Gluing it all together

Building a platform around Prometheus

Who am I?

- Former developer and DevOps engineer (whatever it means)
- Now head of our monitoring platform
- Live in Brno (Czechia)

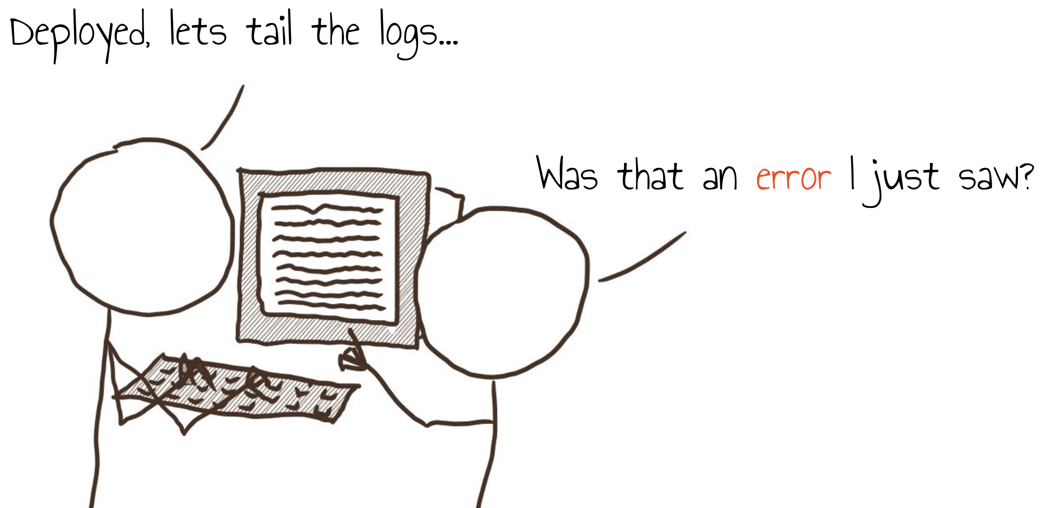
I work with Prometheus since v1.6 (2017)

Where do I work?

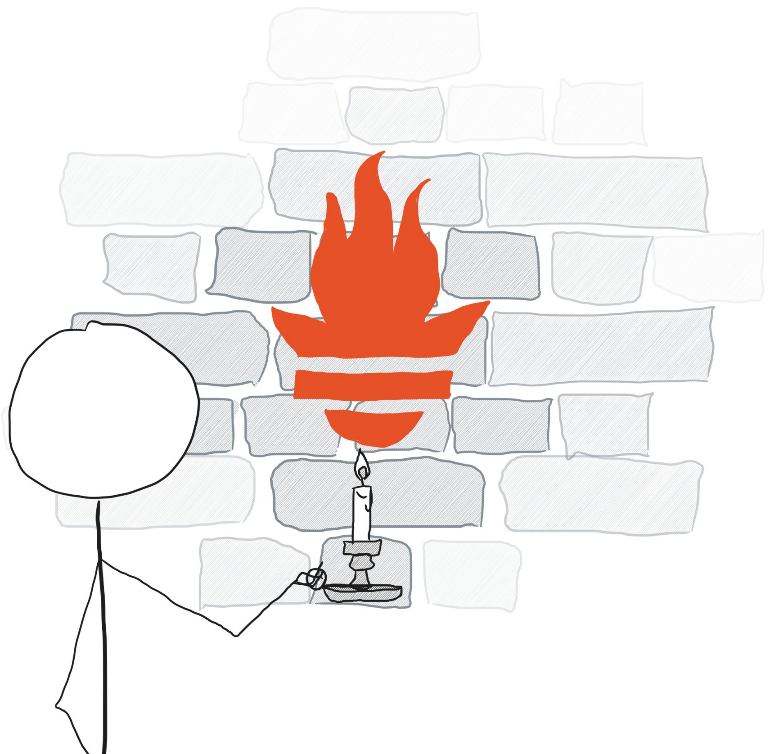
SEZNAM.CZ

- Around 800 developers
- 3 physical data centers (2 own)
- Own internal cloud platform
- 14k physical machines
- 150 k8s clusters
- 25k OpenStack instances

How it started...


Deployed, lets tail the logs...
Was that an error I just saw?

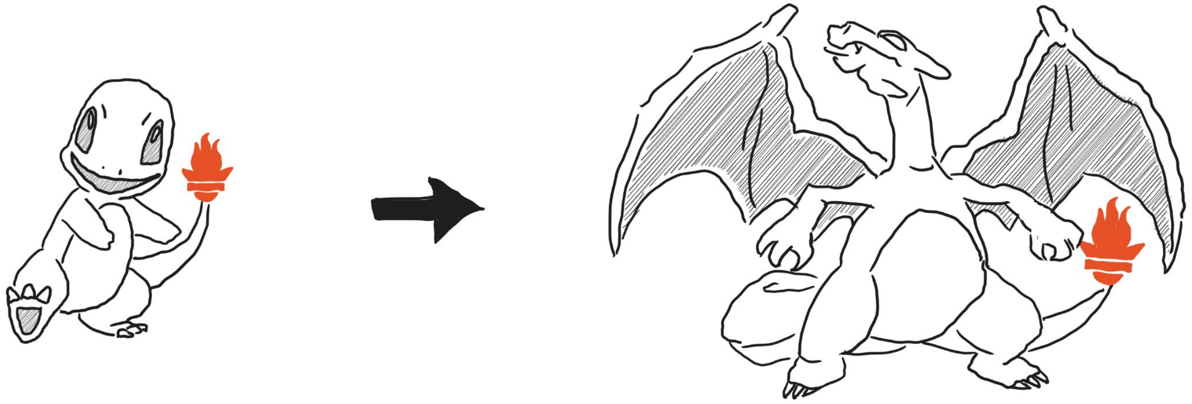
Prometheus occured



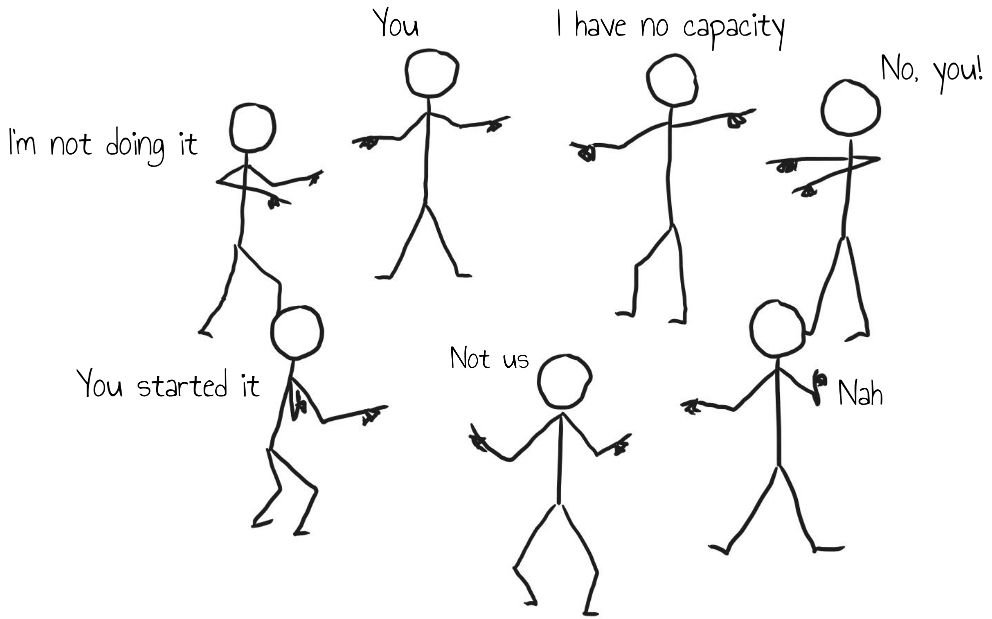
The BOOM

And you get one!
And you get one!
And you get one!
Everyone gets one!

Evolving the setup



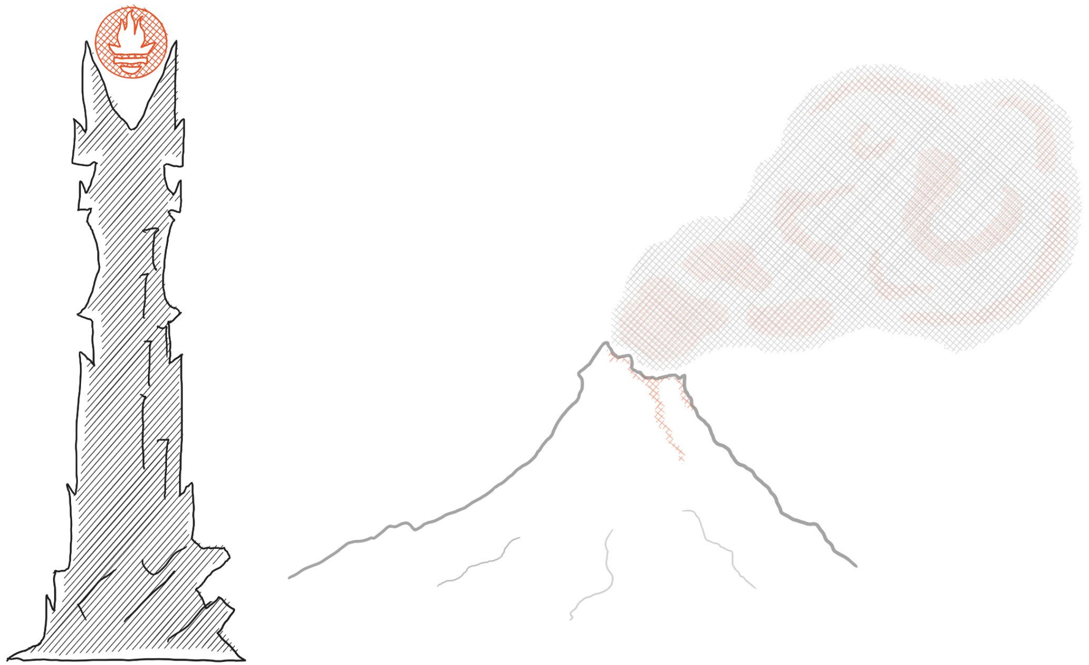
Who will maintain it?



With great power comes...

- Keeping up the track with new features
- Understanding of the system
- Each setup bit different
- Debugging issues
- Upgrades
- Scaling
- On-call

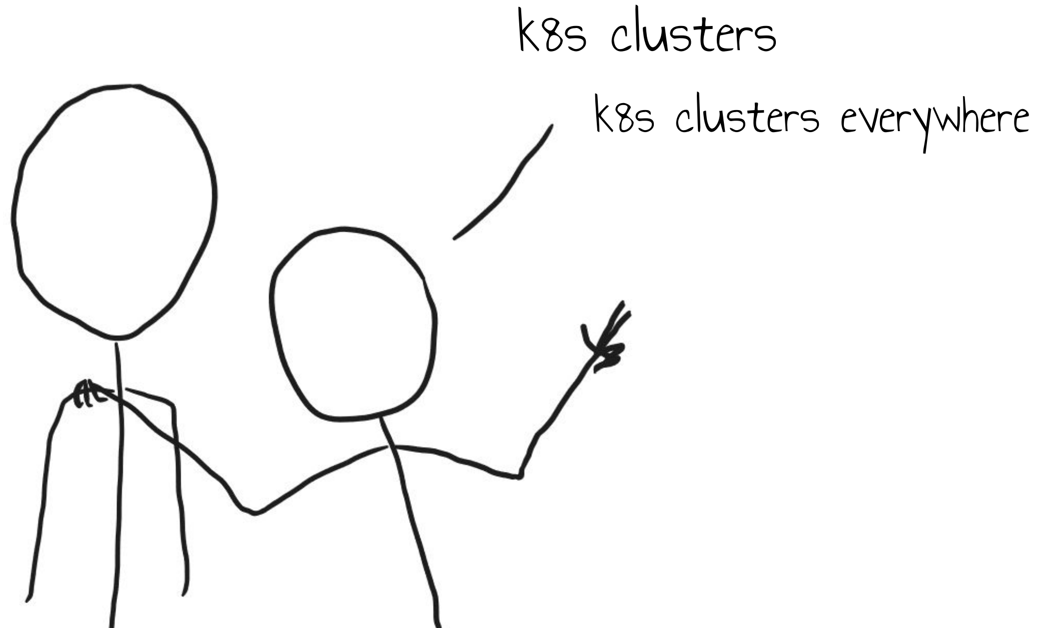
One platform to ~~rule~~ serve them all!



What we wanted to achieve

- Devs just needs to expose the metrics (happy path)
- Easy to configure
- Simple interface
- Separation of tenants to minimize noisy neighbors
- Billing of the tenants
- Resilience for overloading
- Authentication and authorization

Who are the users?



Service discovery in k8s?

Annotations

+ Simple for users
+ Uses bare k8s resources
+ Easy on Prometheus (one scrape job)
- No advanced scrape configuration (relabeling, authentication, ...)
- Hard-wired label schema

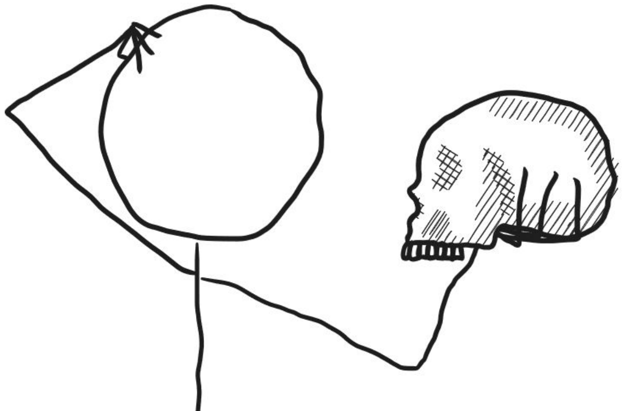
Prometheus operator

+ De-facto standard used in charts etc
+ Plenty of documentation and upstream resources
+ Highly customisable
- Requires special k8s CRD
- Possibly heavy on Prometheus (scrape job per monitor)

To alert locally or not to?

The best practice says: **"Always alert as close to the app as possible"** , to avoid any possible further issues with the infrastructure so the alerting is reliable as possible.

But at the same time everyone embraces agents remote writing to a central storage where local alerting is not possible like no best practises exist...



Use remote write?

Thanos (without remote write) queries the most recent data from its origin. This might lead to overloading of the Prometheis and needs to fan out to all the instances where the data lies repeatedly until the data gets to the central storage. On the other hand, in case of central storage failure, most recent data can be still accessed and in case of temporary failure of a Prometheus you will know what data is missing and get warning.

Remote write ships the data to central storage right away and it's queried from the central storage. But in case of issues with the remote write, you may end up blind (if using agents) or without global view (lots of Prometheis spread everywhere).

Since the scale should have been quite big, the Thanos approach seemed that might hit a limit with bigger queries fanning out everywhere. We decided to go with the remote write.

Managed Prometheis

Should developers manage their own Prometheus instances?

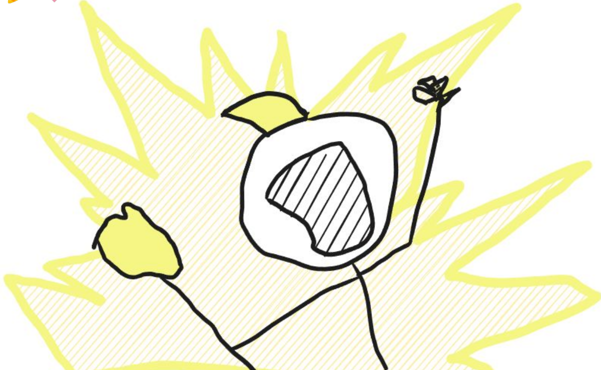
nope.

It's still no easy task if you have to upgrade, watch new features, adjust the setup etc even with prometheus-operator.

So we needed to provision Prometheus for each tenant in each k8s cluster it has a workload in and configure it to monitor all tenant namespaces in the cluster.`

Managing other parts?

- Alertmanager? – Yes!
- Grafana? – Yes!
- Pushgateway? – No! (but eventually yes... 💩)
- Black box exporter? – Yes (but limited capabilities)
- PromLens? – Yes! (update: not needed anymore 🎉)
- Exporters? – No!



Querying other tenants data?

We definitely like the "democratize metrics" approach, but on the other hand, we need to be able to tell who is querying which metrics and how much, and be able to block it so it doesn't overload other tenants, etc.

So we wanted to separate the tenants, but allow anyone freely access metrics of the others. Possibly restrict the access in rare cases.

Other Seznam cloud platform metrics?

Special case of accessing other tenants metrics are other Seznam internal cloud platform services (MySQL, s3 like storage, OpenStack, k8s etc). Developers need an insight to these services so they can also alert on it.

Previously, everyone had to figure out where to federate these metrics from and they were copied across the infrastructure. We wanted to avoid that.

Each service has its own tenant and provides the metrics useful for users there with a catalogue of the provided metrics.

Central configuration management?

Centralized repository for rules, alerts routing and other related configurations supporting Jsonnet for easy sharing of alerts and parts of configuration between tenants. Also we can provide libraries with alerts for users to alert on the services they use.

|The central repository allows us to provide a managed CI that validates, tests (promtool, amtool, promruval, PromQL unit test, amtool, custom checks) and deploys all the configuration with permissions for tenants to approve their changes.

So how is it going?

The happy path

1. User requests a new tenant (issue)
2. Creates a namespace and labels it with a name of its tenant
3. Deploys the app and monitor resource optionally with rule resources

That's it, Prometheus for the tenant will automatically be spinned up and start to scrape the app as configured by the monitor resources and evaluate the rules if deployed.

All the data are sent to the central storage using remote write and alerts to the central alertmanager.

Unified interface

```
monitoring.foo.bar/<tenant>/prometheus
monitoring.foo.bar/<tenant>/alertmanager
monitoring.foo.bar/<tenant>/grafana
monitoring.foo.bar/<tenant>/pushgateway
monitoring.foo.bar/<tenant>/promlens
monitoring.foo.bar/<tenant>/k8s-prometheus/<cluster>
```

Our setup

- Central proxy that manages all routing, authentication and authorization
- Custom operator creates and configures Prometheus pair for each tenant in k8s clusters based on namespace labels
- Prometheus operator that runs the Prometheis and provides the service discovery using monitors and loads local cluster-scope rules
- Prometheis sends the data to the central storage (mimir) and alerts to the central alertmanager (mimir)
- We use mimir query federation to allow access to metrics of other tenants
- Each tenant has own Grafana instance (so we can upgrade individually)
- One central git repo with all the configuration

Size of the setup

70 tenants

180M series

7,5k rules

8M samples/s

200 queries/s

750 Prometheis

So.. everything is 🌈 and 🦄 !

But even the unicorns have to 💩 sometimes

Let's talk about what went wrong

Remote Write issues

- Replaying of data after an outage delays newest data
  (newer data after an outage is more important than replaying of the old)
- Data that should be replayed will be lost on Prometheus restart
- Partial inconsistencies in query results without knowing
  (if one prometheus out of many stops sending data, you won't notice)
- Need to delay queries to avoid inconsistencies
  (sending data takes its time and may lead to inconsistencies, worst are classic histograms)
- Hard to debug
  (no returned status codes in Prometheus metrics, hard to find out which data were lost)
- Data are sent twice from a Prometheus HA pair
  (mimir accepts data only from one of the HA pair)

Remote Write issues
(Prioritization of new data and deduplication)

The remote write v2 unfortunately does not address most of these issues (yet).

Idea to implement sidecar proxy that would do the "deduplication" and possibly prioritization of the writes that would replay data after the outage as Out Of Order samples rather than delaying the newest data.

Remote Write issues

(Missing data)

Make sure you have some notion of all the running Prometheis that sends data over the remote write so you can alert using absent on it.

Also if you want to verify that all the data were replayed successfully after an outage and nothing was lost, it's merely impossible.

Meta monitoring and remote write

If you choose to use agents without local alerts, you should run at least some minimal Prometheus instance to monitor the agents if the remote write is working.

Alering just on the absence of data from the prometheus is fragile and slow (staleness) and if you lose the data, you won't be able to investigate the issue.

Scaling Prometheus

If you decide to go with full Prometheus instead of agents, you will have to size the Prometheus to match the metrics load of each user.

Make sure to automate scaling of the Prometheis resources (VPA) and volumes (volume autoscaler) otherwise you might end up communicating with the users about how they plan to grow and doing all this manually.

Scaling Prometheus

(sharding)

If you decide to go with full Prometheus instead of agents, you might need to shard it at some point. Sharding using hashing is easy to automate (prometheus-operator supports it now) but harder for user to find where their data are located. Semantic sharding on the other hand can be hard to implement to be effective.

Both comes with an issue with alerting, when you have to know where to deploy your alerts (semantic makes this more predictable).

Issues caused by users

- Leaking of unique values to labels
- Sending too many alerts to OpsGeine leading to rate limiting
- Failing rules (many to many joins in Prometheus rules etc)
- Sudden blast of metrics without notice

Make sure to apply various limits to prevent such cases so users do the capacity planning. But make sure to have a way to notify the users automatically so they know they are hitting the limits!

Calculate the price

Even if it is an internal platform, I'd recommend reporting usage of the main resources (samples scraped, sent to remote storage, stored per hour, bytes analyzed by queries) per tenant.

It's hard to pursue anyone in company to optimize their metrics, rules or dashboards, until you show them the cost.

What's next

- Make remote write more resilient and optimized
- Provide external monitoring using BBE
- Standardization of maintenance metrics across company to be used for inhibiting
- Switch to hybrid mode combining agents and Prometheis?

That's all folks!