Scale your Prometheus monitoring with Thanos

Who am I ?

Julien Leloup

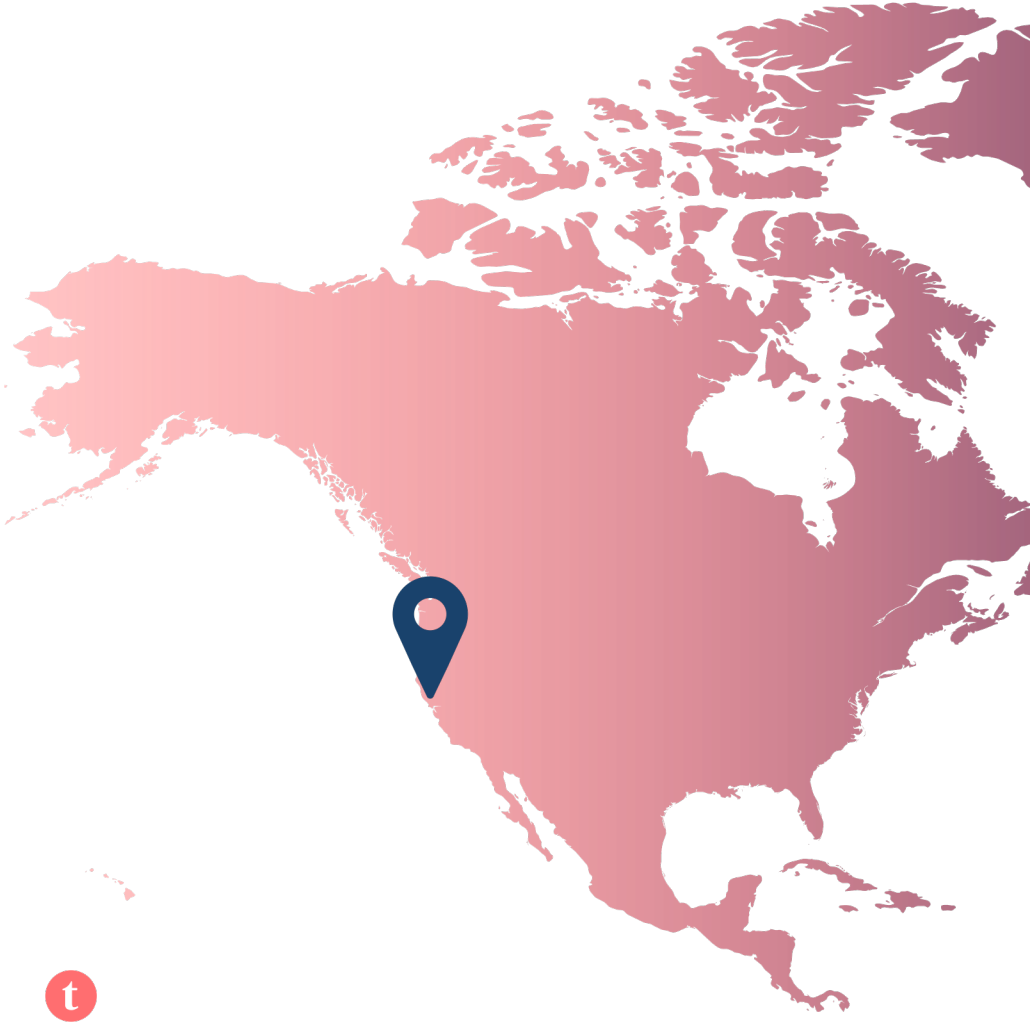
- Site Reliability Engineer @ Talend since 2016
- Working in the SRE Runtime team where we deal with Kubernetes, monitoring and Logging
- Specifically focused on monitoring topics

Some context

Sizing, expectations & choices

Talend & monitoring

- 4 production environments
- AWS & Azure
- Between 1500 & 2500 Kubernetes pods
- Up to 200 instances (EC2 or AzureVM)



Introducing Prometheus @ Talend

Beginning of 2018:

- Datadog is too expensive
- Need to monitor all environments including dev
- Introduce monitoring in infrastructure as code

Options:

- SaaS based ?
- Telegraph & InfluxDB ?
- Prometheus ?

Decision: go for Prometheus

- Open source & **well documented**
- Lots of traction
- Well suited with **Kubernetes**

Limitations of Prometheus

Challenges to be matched

Prometheus characteristics

Local storage only

Fast & efficient
No native data replication
Not suitable for long term storage
Default: 15 days of data

Scalability

Split Prometheus in smaller pieces
Incresed complexity in the architecture

High Availability

Two Prometheus replicas scraping the same targets
Alerts deduplication in Alertmanager
But no deduplication during PromQL processing

Remote Read & Write

Integrations with other tools
Remote Write: push data in a compatible storage
Remote Read: access local data

Prometheus challenges

Challenges	Workaround	Long term solution
No long term storage	Prometheus vertical scaling if needed	Prometheus remote_write to a supported backend ?
Scalability & HA = more complex read path	Use one Grafana datasource per Prometheus	Prometheus server federation ?
High availability = weird Grafana experience	Live with it: does not prevent metrics to be displayed & does not impact alerting	Deduplication during PromQL queries ? Or in a single storage ?

Thanos as a Prometheus overlay

How to solve most of the challenges

Introducing Thanos

Open source “overlay” for Prometheus

- Built as a micro-service with self service components
- Loose coupling with Prometheus
- Single entrypoint to access all Prometheus metrics with deduplication
- Provide long term storage based on S3/Azure/GCS/Minio
- Downsampling

Thanos official website

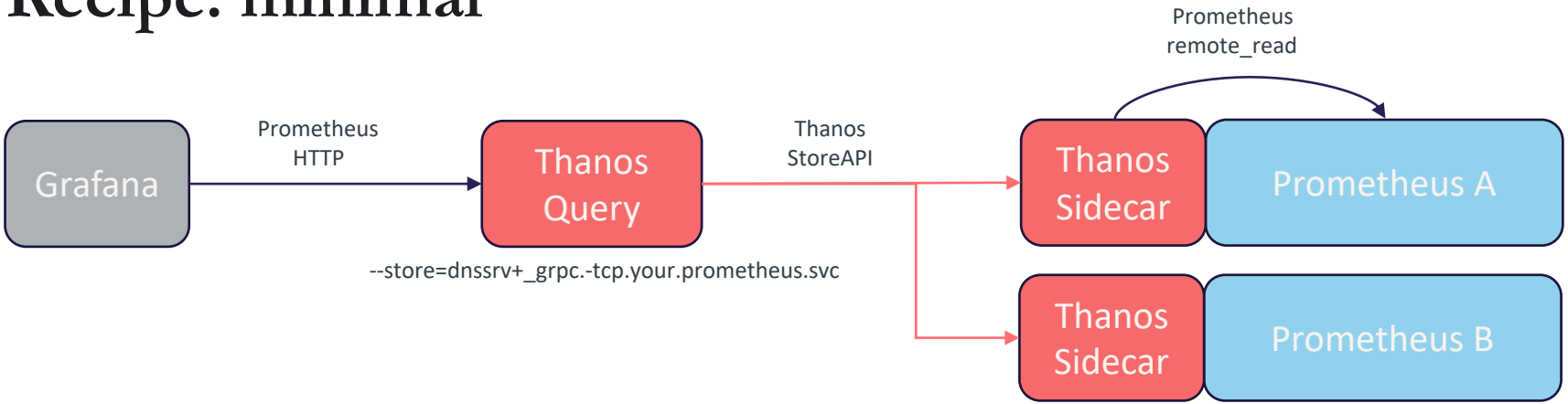
Externally:

- Exposes Prometheus HTTP API
- Compatible with all clients using PromQL
- Grafana datasource: type Prometheus

Internally:

- Uses gRPC for cross-service communications
- Strong segregation between Prometheus API & Thanos API

Recipe: minimal



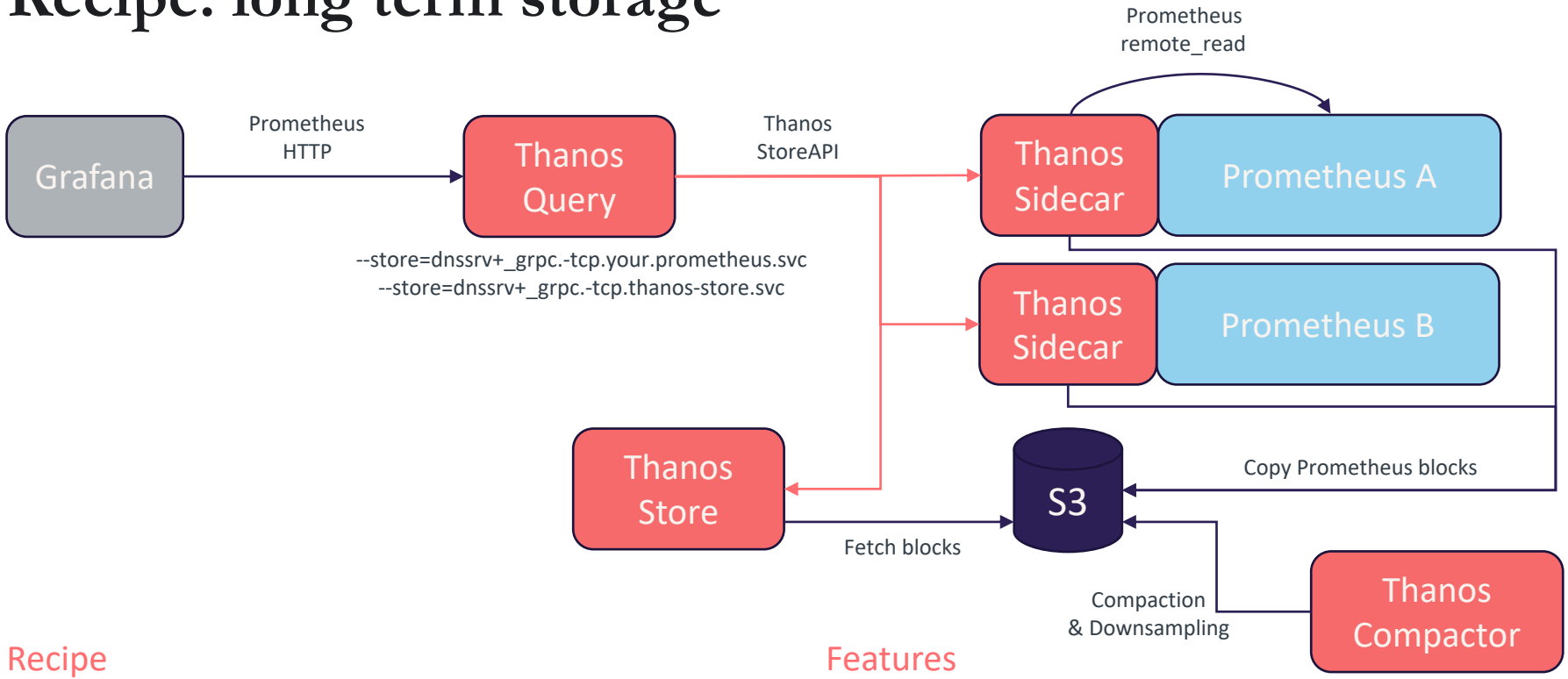
Recipe

- Add a Thanos Sidecar to your Prometheus
- Deploy Thanos Query

Features

- Single entrypoint for all metrics
- Simple architecture

Recipe: long term storage



Recipe

- Minimal setup + Store Gateway
- Recommended: use Thanos Compactor

Features

- Provide long term storage

Other Thanos components

Thanos Ruler

Prometheus rules on metrics from Thanos Query
Useful on some edge cases like alerts based on absent()

Thanos Query Frontend

Provide caching & parallelization on top of Thanos Query

Thanos Receive

Exposes a remote_write compatible endpoint
Stores data in a local Prometheus TSDB
Used when you can't use a standard Thanos Sidecar

Thanos Tools

Set of extra tools
Help debugging especially the object store

What we learned from running Thanos

Simple to onboard

Start simple & add components when needed

Nice features

Query deduplication is quite useful
Query caching & parallelization helps on performance
Downsampling could be quite beneficial depending on your use case

Hooks well with Prometheu-operator

prometheus-operator can deploy Thanos Sidecar
PrometheusRule support in Thanos Ruler

Storage is cheap, computation isn't

Lots of metrics in long term storage = lots of Store Gateway memory
Beware also of Thanos Compactor that can eat up a lot of memory