Google

Data freshness monitoring in complex data pipelines

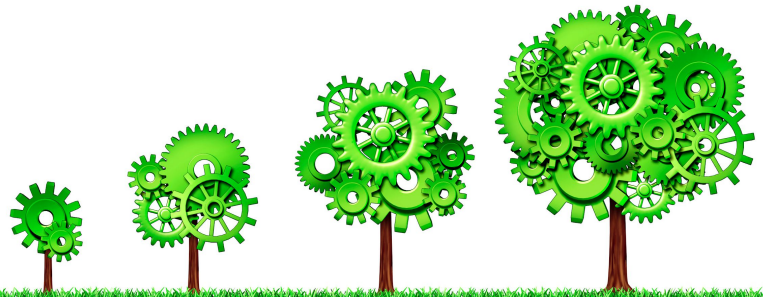
SREcon21 | USENIX

Alexey Skorikov

Site Reliability Engineering

Common challenges many organisations face

- Growing complexity in data pipelines and organizations
  - More nodes and more interactions in business data flows
  - More teams touching pipelines makes coordination challenging
- Lack of clarity and awareness in end to end data dependencies
- Growing technical debt as a result of rapid growth

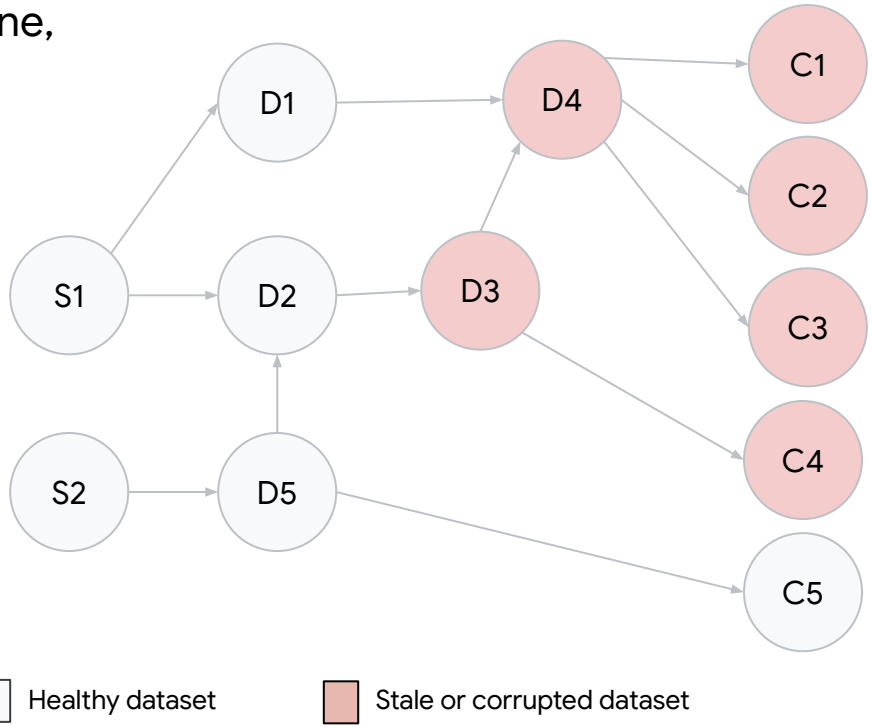


Implications on data integrity

- Lack of operational readiness to deal with specific failure modes and their downstream effects
- Outages due to slow bleed, misconfigured data interfaces
- Longer times for incident investigation, mitigation and resolution

Risk multiplies with each hop downstream

A part of data pipeline, batch or streaming



Data sources

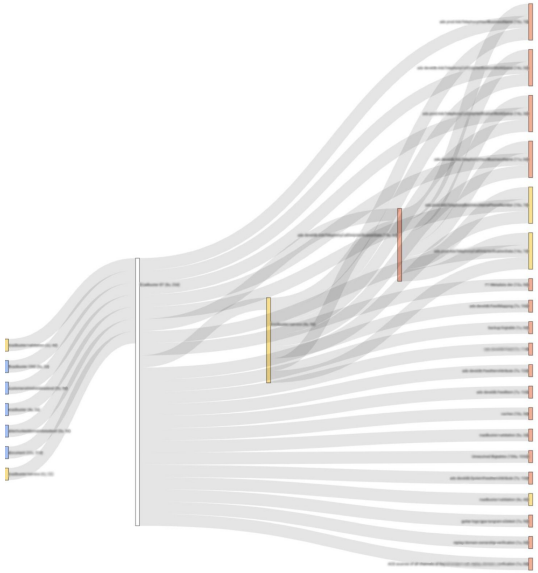
Points of consumption

Cornerstone 1

Understanding critical business data flows, upstream & downstream dependencies

Mapping data dependencies automatically

data
access
patterns



Recording data access events

Levels

- Job binary
  - Process

Persistent storage types

- File storages
- Databases

Examples of recorded metadata

- Modification time, access time
- Writer, reader
- Logical size

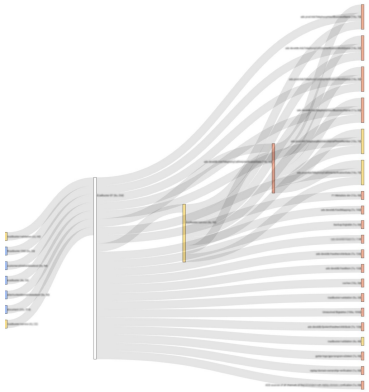
Provision forms

- Sampled flow time series
- Knowledge base

Sources

- Access logs scrapers
  - Storage systems: job-storage
  - Processing systems: job-storage
  - API endpoints: job-job

Mapping a data dependency graph



Dimensions

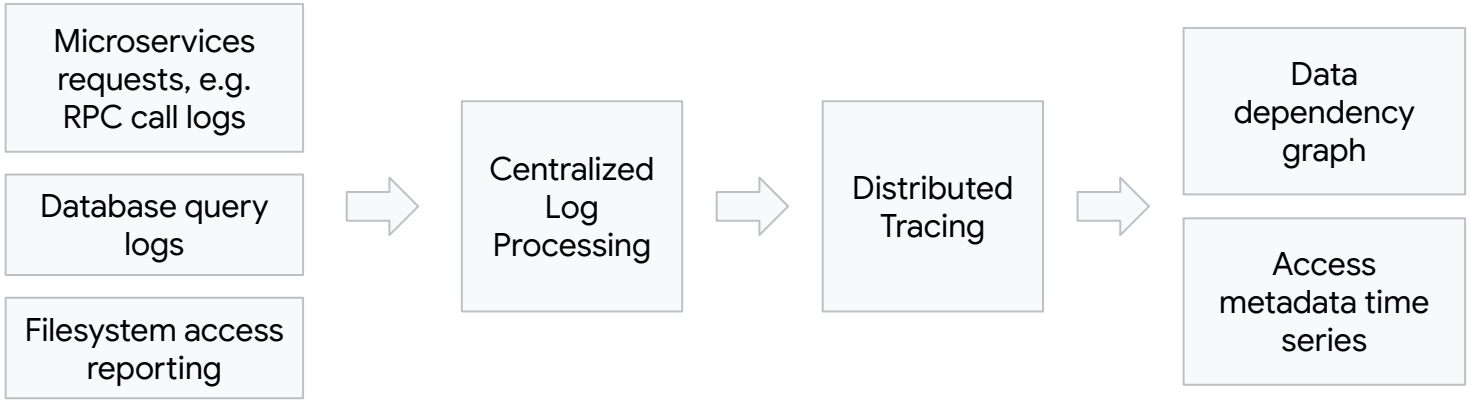
- Time
- Service
- Environment (e.g. prod, staging)
- Instance (e.g. region, datacenter, rack)
- Role: producer, consumer
- Data owner

Logical grouping

- Datasets
  - Groups of data entities with a common purpose
- Systems
  - Binaries, compute functions, etc.

Instrumenting centralized data access recording

- Centralized logging example: Logstash
- Distributed tracing example: Jeager
- Filesystem level events capturing example: ionotify



Continuous data annotation

How

- Automatic inference
- Regular teams' review

Examples

- Criticality
- Ownership (team contacts)
- Data recovery objectives
- Data retention plans

Plethora of DI solutions to build on top

Early detection of slow bleeds

horizontal monitoring to catch data staleness and corruption

Risk analysis

holistic view of impact on downstream consumers

Business continuity

data integrity and recovery plans

Incident response aid

traceability of data dependencies and ownership

Data validation

real time data validation and consistency checks

Plethora of DI solutions to build on top

Early detection of slow bleeds

horizontal monitoring to catch data staleness and corruption

Risk analysis

holistic view of impact on downstream consumers

Business continuity

data integrity and recovery plans

Incident response aid

traceability of data dependencies and ownership

Data validation

real time data validation and consistency checks

Cornerstone 2

Holistic data monitoring to eliminate slow bleeds caused by data staleness or corruption

Slow bleed problem

Misconfiguration between upstream and downstream systems, bad migration or schema update, permission issues, network issues, resource starvation, concurrent writes, change in input data

↓

Producer systems fail to update datasets or generate corrupted data

↓

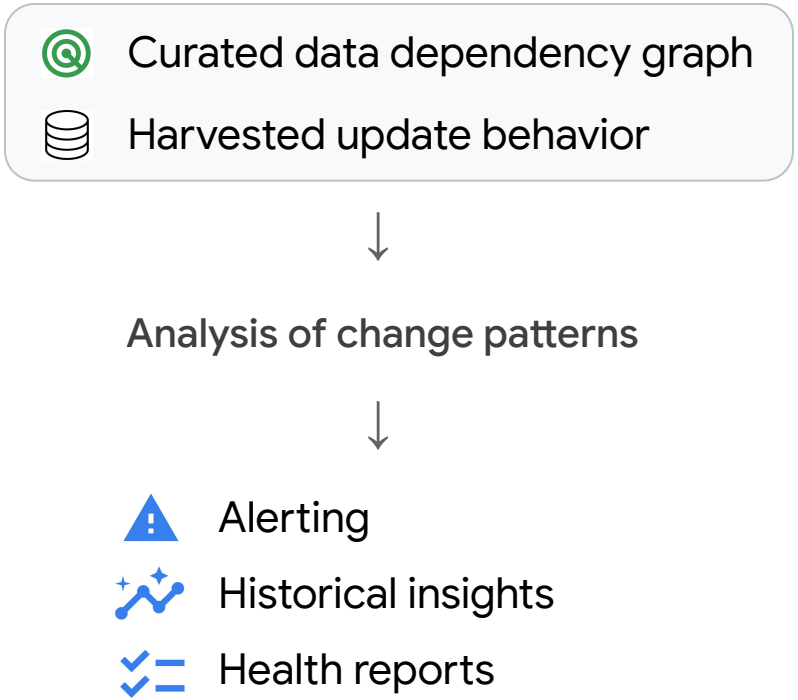
Consumer systems downstream keep reading **stale or corrupted data**

Slow bleeding may go **unnoticed** for weeks or months. Daily incremental impact is hard to detect.

↓

$$$ Large accumulated business impact

Freshness monitoring at scale



Comprehensive solution at scale

Minimum effort for adoption

Cost-effective additional layer of defence

Types of detectable data anomalies

Data staleness

- Dataset's last modification time > expected age threshold
- Age threshold auto-adjusted based on historic performance
  - Helps make a distinction between static and dynamic data

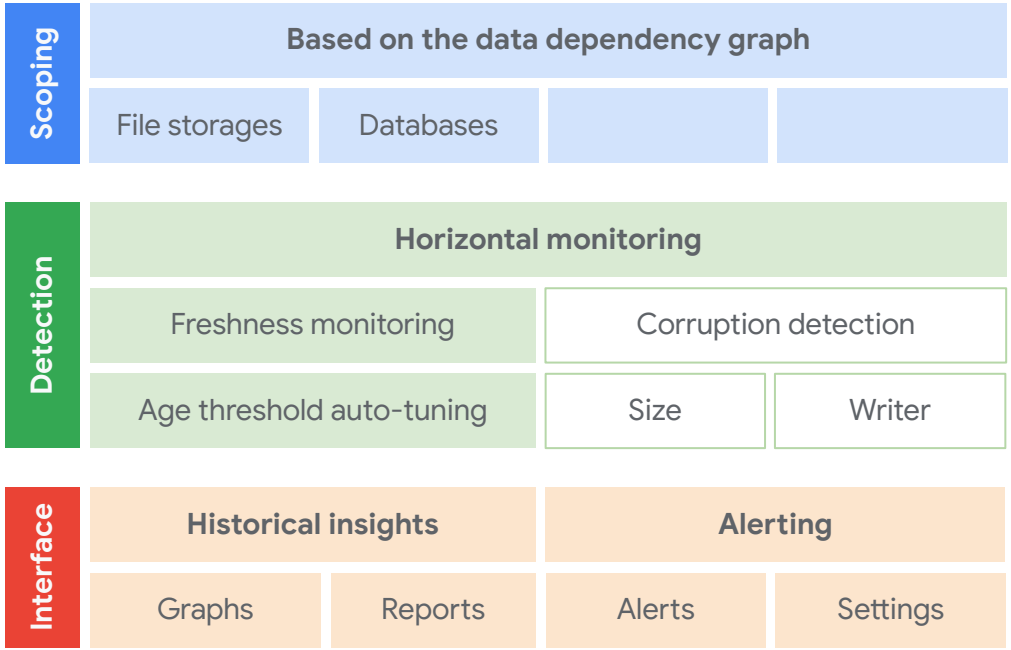
Data corruption

- Direct, e.g. data consistency checks
- Inferred
  - Explosion: unexpected spikes or dips in logical size or rate of change
  - Rogue concurrency: unexpected writers changing data

Key features

- No direct access to data needed for anomaly detection
- Minimal configuration cost, blanket coverage
- Newly discovered datasets get covered by default
- Health picture for all critical datasets
- Actionable alerting from consumer and producer perspectives

One of Google's solutions: breakdown



Lessons learned

The monitoring approach helped to

- ☑ Detect real data staleness issues
- ☑ Identify obsolete datasets for tech debt to cut DI risks surprize
- ☑ Raise awareness of upstream risks in data consumers

Maximizing signal-to-noise ratio of alerts

- ❖ Basic data annotation is key to infer data criticality
- ❖ Auto-tune alert thresholds based on data change patterns
- ❖ Recognise copies and shards of datasets
- ❖ Exclude obsolete data from alerting based on access times
- ❖ Exclude static data from alerting based on heuristics

Thank you.