Exception Monitoring at Scale
About Us

Jorge Creixell
Platform Lead at SoundCloud
@jcreixell

Marc Tuduri
Senior Engineer at SoundCloud
@marctuduri

- Opinions are our own -
Why Monitor Errors?
Why Monitor Errors?

**Metrics and Alerting**
Quantitative, allow to determine whether a system operates within acceptable thresholds.

**Logging**
General purpose audit trail of events of interest.

**Error Monitoring**
**Qualitative**, specialized indexing and aggregation of errors for gaining insights into the source of a problem *(request context, backtraces, etc)*. Useful during investigations and incident response.
Our Journey
Early Days: Monolithic Architecture
Early Days: Monolithic Architecture
Micro-Service Architecture
Micro-Service Architecture
Limitations

Thundering herd issues
A single bad deploy for a microservice would create a spike of errors, exhausting the entire quota for the month.

Self hosted solutions would need to be overprovisioned, be ready to auto-scale very fast or subject to rate limits, risking losing important signals.

Third-Party Vendor
Crossing internet boundaries, security and sensitive data locality concerns.
Early Alternatives

**Log Tailing**
Extremely verbose logs, **slow**. Like finding a needle in a needle in haystack.

Issues with log truncation/splitting and out of order processing.

**Log Ingestion and Indexing**
Initially discarded due to massive **storage** requirements.
TAKING A STEP BACK
What Do We Really Need?

Requirements

- Complete Index of Errors
- Highly Scalable (traffic, instances, clusters)
- Low resource consumption
- Cluster-Local (cloud native)
- Occurrence Sampling and Request Context (reproducible errors)

Non-Requirements:

- Data durability
- Detailed metrics (already provided by Prometheus)
- Client-Based Errors
Periskop
Design

Pull Based Model

Client Library Aggregates and Samples Errors in Memory

The collector builds a unique key with the exception’s message and a hash of the stack trace.

Periskop Server Scrapes and Further Aggregates Errors Across Instances

Multiple levels of aggregation possible (federation)
Trade-Offs

Pull Based Model

+ **Very Efficient Use of Resources**
  Sampling and aggregation provide a very low memory footprint. I/O reduced to the minimum.

+ **Scales to Very Large Number of Errors and Instantes**

+ **Decentralized**
  Hierarchical collection across multiple data centers possible (federation).

- **Not Suitable for Short Lived Processes**
  Fork-based application servers, batch jobs.

- **Problematic for Crash-Looping Processes**
  Panics, OOMs

- **Less flexibility for Aggregation Strategies**
Main Features
Periskop UI

Services and errors navigation

Select a service

- api-auth
- api-braze
- api-curators
- api-deck
- api-feeds
- api-insights
- api-mobile
- api-mobile-creators
- api-partners
- api-public
Periskop UI

Error search and filtering

Summary

Key
com.twitter.finagle.CancelledRequestException@a1e1fa28

Count
3700727

Severity
error

First Occurrence
7 days ago

Latest Occurrences 1/100

Occurred at
3 minutes ago

Curl
Periskop UI

Mark errors as resolved
Client libraries

Current client implementation of Periskop in the following languages

- Go
- Scala
- Python
- Ruby
func main() {
    c := periskop.NewErrorCollector()

    // Without context
    c.Report(faultyFunc())

    // With HTTP context
    c.ReportWithHTTPContext(faultyFunc(), &periskop.HTTPContext{
        RequestMethod:   "GET",
        RequestURL:      "http://example.com",
        RequestHeaders:  map[string]string{"Cache-Control": "no-cache"},
        RequestBody:     nil,
    })

    // Call the exporter and HTTP handler to expose the
    // errors in /-/exceptions endpoints
    e := periskop.NewErrorExporter(&c)
    h := periskop.NewHandler(e)
    http.Handle("/-/exceptions", h)
    http.ListenAndServe(":8080", nil)
}
```python
if __name__ == "__main__":
    collector = ExceptionCollector()
    try:
        faulty_func()
    except Exception as exception:
        collector.report(exception)
```
Plugable Service Discovery
Prometheus Based

Same SD mechanism as Prometheus

Many types supported

Same Configuration Format

```yaml
services:
  - name: mock-target
dns_sd_configs:
  - names:
    - localhost
    refresh_interval: 10s
type: A
port: 7778
scraper:
  endpoint: "/errors"
  refresh_interval: 10s
```
Push capabilities
Using pushgateway service

For the the cases of fork-based application servers or batch jobs.

Use as sidecar container
Roadmap and Future
Roadmap

- Built-in federation (Hierarchical Collection)
- Time Series Visualization
- More Integrations (Backstage, Grafana?)
- More Languages and Frameworks Supported
- Labelling of Errors
Fun Facts
Periskop: The Name

Inspired by a Very Interesting Office Device
Early Contributors Exchanged Roles

Initial Goal: Solve an existing problem while learning something new

Go Backend Initially built by Android and iOS Engineers

Front-End in Typescript/React by Backend Engineers

Usage of Self Allocated Time (SAT) (thanks SoundCloud!)
Conclusion
Key Takeaways

Periskop is a FOSS exception monitoring system for the cloud

Pull based systems offer good scalability characteristics with some trade-offs

Focus on your needs, optimize your resources

Never stop building new things, learning and contributing
Contribute

https://periskop.io
THANK YOU