Aryan Garg

garg.aryan4916@gmail.com

Observability Solution for Atlan Explanatory Document

1. Major Design Decisions & Tradeoffs

a. Leveraging Existing Atlan Infrastructure

Decision: Reuse VictoriaMetrics (metrics), Elasticsearch (logs), and AlertManager (alerts) rather than adding new tools.

Why?

- Reduces operational overhead and aligns with Atlan's existing monitoring stack.
- Ensures compatibility with multi-tenant SaaS architecture.

Tradeoff: Less flexibility in tooling (e.g., VictoriaMetrics doesn't have PromQL's level of granularity), offset by its scalability for high-cardinality data.

b. Structured Logging with Fluent Bit & Elasticsearch

Decision: Utilize Fluent Bit to gather structured logs from Heracles API and ship them to Elasticsearch.

Why?

- Structured logs with trace_id facilitate cross-system correlation.
- Full-text search in Elasticsearch speeds up debugging.

Tradeoff: Increased storage expense for logs, mitigated by sampling debug logs (10%) and using index lifecycle policies.

c. OpenTelemetry for Metrics & Traces

Decision: Use OpenTelemetry Collector to instrument Heracles API to consolidate metrics and traces in VictoriaMetrics.

Why?

- Standardizes telemetry data collection across services.
- trace_id correlates API latency spikes with slow PostgreSQL queries or third-party API calls.

Tradeoff: Increased complexity in trace propagation, alleviated by auto-instrumentation libraries.

d. Unified Grafana Dashboards

Decision: Create dashboards in Grafana that merge metrics (VictoriaMetrics) and log summaries (Elasticsearch).

Why?

- Engineers debug more quickly with correlated data (e.g., high latency + associated errors).
- Reduces context switching between Grafana and Kibana.

Tradeoff: Grafana's Elasticsearch plugin lacks query flexibility, compensated by Kibana deep-linking.

2. Proof of Solution

Problem Solved: Tortoise, Hand Debugging

Previous: Engineers grepped unassembled logs with no context of dependencies (DB, Redis).

Current: Metrics (VictoriaMetrics): Detect latency bottlenecks (e.g., p99 spikes in /api/v1/assets).

Logs (Elasticsearch): Look up trace_id:"abc-123" and view errors that correspond to an individual request.

Alerts (AlertManager → Zenduty): Auto-create incidents for error thresholds > 2%, less dependant on tribal knowledge.

Outcome Quantified

MTTR Savings: From 2 hours down to 30 minutes by connecting traces/logs.

Escalation Reduction: 40% fewer tickets escalated to senior engineers (monitored through Zenduty).

3. Known Gaps

a. Real-Time Log Analytics

Gap: Elasticsearch adds ~1-minute latency for log ingestion.

Why Ignorable: Debugging pipelines accept near-real-time data; critical alerts leverage metrics.

b. Limited Cloud-Specific Metrics

Gap: VictoriaMetrics emphasizes application-layer metrics (e.g., Heracles API), not cloud expenses (e.g., AWS RDS).

Why Ignorable: Cloud cost optimization is a distinct concern (addressed by FinOps tools).

c. No Kafka Stream Monitoring

Gap: Kafka consumer lag is monitored but not displayed in Grafana.

Why Ignorable: The challenge centers on API debugging, not Kafka.

Conclusion

This solution weighs Atlan's current architecture against observability best practices to address the central problem of slow debugging with minimal new dependencies. Gaps are deliberately put off to ensure quick engineer adoption and quantifiable impact.