Axiom — Observability & Reliability Design

1. Purpose and Scope

Axiom is a distributed financial technology platform supporting brokers and quantitative engineers across multiple AWS regions and on-premises environments.
Reliability, latency, and uptime directly affect broker productivity and trading operations.

This document outlines the observability and reliability approach I recommend for Axiom, based on practices that have worked well in similar hybrid environments.
It enables engineering and operations teams to:

- Detect service degradation before users experience issues
- Trace end-to-end broker workflows from desktop to backend
- Govern service health via SLIs, SLOs, and error budgets
- Automate alerts, scaling, and anomaly detection while maintaining cost efficiency

2. Observability Architecture

2.1 Core Stack

Layer	Tool	Purpose
Instrumentation	OpenTelemetry SDK + Collector	Unified telemetry for metrics, logs, and traces
Metrics	Prometheus (per region/on-prem) + Thanos	Metric collection, global aggregation, and long-term retention
Logs	Fluent Bit → Logstash → Elasticsearch → Kibana	Centralized structured logging and analysis
Traces	OpenTelemetry Collector → Jaeger (Elasticsearch backend)	Distributed tracing and dependency visualization
Dashboards / Alerts	Grafana + Alert manager	Visualization, SLO dashboards, and alert routing
Cloud Integrations	CloudWatch Exporter	Metrics from ALB, SNS/SQS, DynamoDB, S3, Route 53
Synthetic Monitoring	Blackbox Exporter	End-to-end probes simulating broker interactions

This stack is built around components that most SRE and DevOps teams already know well, which keeps onboarding and maintenance simple.
It provides unified, scalable visibility for both on-prem and cloud workloads without introducing niche tooling.

2.2 Data Flow Overview

1. **Application Telemetry** – Each microservice (Edge, Layout, Access Control, Deployment, File Server) emits OpenTelemetry metrics, logs, and traces with shared trace IDs.
   In practice, this flow ensures we can trace any broker action—from the desktop client all the way to backend queues—with a single trace ID.
2. **Collection & Processing** – Prometheus scrapes metrics; Fluent Bit forwards logs to Logstash; the OTel Collector exports spans to Jaeger; CloudWatch Exporter gathers AWS service data.
3. **Aggregation & Storage** – Prometheus nodes push compressed blocks to S3 via Thanos Sidecar. Thanos Querier and Store Gateway merge all regions into a unified dataset. Logs and traces use Elasticsearch hot/warm/cold tiers.
4. **Visualization & Alerting** – Grafana consumes Prometheus, Elasticsearch, and Jaeger data sources; Alert manager routes alerts to Slack, Teams, or PagerDuty.

2.3 Deployment Model

- **AWS EKS:** Prometheus Operator, Fluent Bit Daemon Set, OTel Collector, Thanos Sidecar.
- **On-Prem Kubernetes:** Identical agent setup for portability.
- **VMs / Docker:** Lightweight Prometheus Agent and standalone Collector.
- **Security:** mTLS between collectors, IAM-based exporter roles, and S3 encryption at rest.

2.4 Data Model Standards

Logs: timestamp, level, service, env, region, trace_id, span_id, tenant, request_id, message
Metrics: service, route, region, env, version (bounded cardinality)
Traces: service.name, http.route, sqs.queue, region, tenant, error.code, latency_ms

---

3. SLIs, SLOs, and Error Budgets

#	Service / Journey	SLI (Measurement)	SLO Target	Source
1	Broker “Open Layout”	Availability = (2xx ÷ total requests)	≥ 99.5 %	Prometheus + Blackbox
2	Layout Load Latency	P95 trace duration	< 300 ms (in-region) / < 600 ms cross-region	Jaeger + Prometheus
3	Layout Deployment Timeliness	% deploys ≤ 5 min	≥ 99 %	Prometheus + CloudWatch
4	Access Control Latency	P95 authorize request	< 100 ms	Prometheus
5	Event Bus Health	Age of oldest SQS message	< 60 s normal / < 180 s peak	CloudWatch Exporter

These SLOs balance what’s realistically achievable by engineering with what’s meaningful to brokers — availability and latency directly impact their productivity.

Error-Budget Policy

- 99.5 % SLO ⇒ 3 h 22 m monthly error budget.
- Hard gate: burn-rate ≥ 14× (1 h) or 6× (6 h) → block deployments.
- Soft gate: > 75 % budget used → pause non-critical releases.
- Auto-rollback if breach > 10 min.
- Grafana burn-rate panels track consumption and SLO compliance.

I’ve seen this burn-rate based gating work well to prevent reactive firefighting and keep releases data-driven.

---

4. Alerting & Incident Workflow

Alerting should surface only what needs human attention.
This workflow keeps that principal front and centre.

4.1 Paging Policy

Critical (PagerDuty):

- SLO burn-rate breach
- SQS backlog > 120 s
- ALB success < 99 %, 5xx > 0.5 %
- Cluster unavailable

Warning (Slack / Teams):

- Pod restarts, CPU > 90 %, disk > 85 %, DynamoDB throttles

4.2 Escalation & Noise Control

- Escalation: SRE → Service Owner → Duty Manager
- Group alerts by service, region, env
- Upstream failures suppress dependent alerts (inhibition)
- Weekly reviews remove stale alerts and adjust thresholds

4.3 Runbooks

Incident Type	Immediate Action
Edge Service 5xx Errors	Check Prometheus error-rate metrics and correlate with recent deployments; rollback via GitLab if deploy-related.
SQS Backlog Growing	Verify SQS consumer lag metrics and service health; scale consumers horizontally using Kubernetes HPA or ECS autoscaling.
High Latency on Layout Service	Check p95 latency traces in Jaeger; inspect upstream Edge and downstream File Server dependency health.
Access Control Failures	Verify token service and LDAP connector availability; restart affected pods and invalidate stale tokens if necessary.

---

5. Reliability, Scaling & Cost Optimization

5.1 Logging (ELK)

- Structured JSON only, INFO sampled 10 %.
- ILM: 7 d hot → 30 d warm → 90 d cold → S3 archive.
- Remove DEBUG logs and mask PII fields.

5.2 Metrics (Prometheus + Thanos)

- Scrape interval 15 s (app) / 60 s (infra).
- Retain 15 months via Thanos down sampling (5 m / 1 h).
- S3 lifecycle policies auto-expire stale data.
- Global queries and HA via Thanos Querier + Store Gateway.

5.3 Traces (OpenTelemetry + Jaeger)

- Tail sampling 5 % baseline / 100 % errors / 25 % slow.
- Retention 7 d hot / 30 d warm storage.
- Jaeger uses Elasticsearch backend linked to Grafana via exemplars.

5.4 Automation & Resilience

Where possible, remediation should happen automatically — human time is too valuable to spend restarting pods or clearing queues.
Over time, this automation helps SREs shift from reactive ops to proactive capacity management.

Auto-Scaling & Self-Healing

- **HPA:** scales microservices by CPU %, memory %, and latency.
- **KEDA:** scales event-driven workloads based on SQS queue depth.
- **VPA:** tunes resource requests based on Prometheus history.
- Failed rollouts trigger **auto-restart** or **Argo Rollout rollback**.

Auto-Remediation

- Alert manager → webhook → Remediation Service executes safe actions.
- Examples:
    - SQS backlog > threshold → scale consumers ×2
    - Node disk > 90% → rotate logs / resize EBS
    - Crash Loop → rollback deployment
- Every action logs to operations. Audit in Elasticsearch.

Predictive Scaling & Anomaly Detection

- Prometheus Adapter + AWS Forecast anticipate load (e.g., market open).
- predict_linear() detects metric drift and notifies SRE via Slack.
- Dynamic thresholds adjust for sustained growth trends.

Chaos & Recovery Validation

- Monthly **LitmusChaos** tests for pod, node, and network failure.
- **Route 53 failover drills** verify regional recovery.
- Recovery objectives: MTTR < 10 min, RPO ≤ 5 min.

Observability-Driven Automation

- Alert manager triggers → Runbook Automation Service.
- Trace anomalies → temporary log-level elevation.
- All automation emits telemetry for observability of itself.

---

6. Dashboards

Dashboard	Purpose
SLO Overview	Displays availability, latency, and error-rate objectives with error-budget burn rate.
Edge Service Dashboard	Shows request rate, success/error ratio, and regional latency percentiles.
Layout Service Dashboard	Visualizes layout fetch latency, cache hit rate, and deployment timings.
Event Bus Dashboard	Tracks SQS queue depth, publish/consume rates, and message age.
Access Control Dashboard	Monitors auth latency, token validation failures, and role mapping errors.
Infrastructure Dashboard	Aggregates node health, pod status, container CPU/memory, and storage utilization.

In my experience, a single well-designed SLO dashboard does more for incident readiness than a dozen metric charts.
Dashboards follow the principle *"one glance, one decision"* and serve SREs, developers, and support teams alike.

---

7. Implementation Outline

Phase 1 (Setup): Deploy Prometheus Operator, Thanos, Fluent Bit, Jaeger, Grafana, and ELK.
Phase 2 (Instrumentation): Integrate Open Telemetry SDK into Edge and Layout services.
Phase 3 (Dashboards & Alerts): Define SLO dashboards, burn-rate alerts, and deploy markers.
Phase 4 (Hardening): Implement chaos testing, ILM tuning, and auto-scaling validation.

---

8. Outcome

This observability design enables Axiom to detect issues early, trace user flows end-to-end, and maintain governed reliability through clear SLIs and SLOs. It ensures efficient data usage, cost control, and automation-driven resilience — giving teams global visibility and confidence in platform stability.