# Observability, Logging & Alert Catalog

## Purpose and scope

This document defines the observability strategy for PineCone Pro’s ERP/IMS and specifies the events, metrics and alerts required to operate the system reliably. Observability is the ability to understand the internal state of a distributed system based on the data it produces【206002500321820†L55-L79】. By capturing and correlating logs, metrics and traces, operations teams can detect anomalies, troubleshoot root‑causes and make data‑driven improvements. This catalog covers:

* **Logging** – structured, centralized and correlated logs with consistent levels and retention policies.
* **Metrics** – quantitative measurements that describe service behaviour, business KPIs and infrastructure health, grouped into golden signals and domain‑specific indicators【572740891080354†L82-L97】.
* **Tracing** – end‑to‑end tracking of requests across services using distributed tracing to understand latency and dependencies【206002500321820†L323-L333】.
* **Alerting** – design principles, severity tiers and detailed alert definitions for each module based on service‑level objectives (SLOs).

## Observability framework

### Logging best practices

* **Structured logs:** Adopt JSON‑formatted structured logging so that logs are easily parsable and queryable. Include key fields such as timestamp, service name, severity level, message, entity identifiers (e.g., order ID), error codes, and correlation/trace IDs【206002500321820†L153-L173】. Avoid generic messages – be specific about what happened and why【206002500321820†L175-L176】.
* **Log levels:** Use standardized log levels: **TRACE** for verbose diagnostics, **DEBUG** for debugging, **INFO** for routine operations, **WARN** for potential issues, **ERROR** for recoverable errors and **FATAL** for critical failures【206002500321820†L178-L193】. Configure lower levels (TRACE/DEBUG) only in development; limit production logs to INFO and above to manage volume【206002500321820†L194-L196】.
* **Centralized logging:** Send logs from all microservices to a centralized logging system (e.g., Elastic Stack) for aggregation and search【206002500321820†L198-L204】. Use Fluentd or another agent to collect logs and forward them to Elasticsearch where Kibana provides dashboards and search【572740891080354†L251-L260】.
* **Correlation IDs:** Generate a unique correlation ID at the entry point (API gateway) and propagate it through downstream service calls. Include this ID in every log entry so a single transaction can be traced across services【206002500321820†L212-L265】. When an order traverses multiple services, each log should share the same correlation\_id and include relevant context (customer ID, order total, error code, etc.)【206002500321820†L211-L256】.
* **Sensitive data & retention:** Redact personally identifiable information (PII) and payment data from logs. Define retention policies aligned with compliance requirements (e.g. 90 days for application logs, 365 days for audit logs). Archive older logs to a long‑term storage bucket for forensic analysis.

### Metrics best practices

#### Key performance indicators

Adopt the **four golden signals** recommended by Google’s Site Reliability Engineering book: **latency**, **traffic**, **errors** and **saturation**【572740891080354†L82-L97】. These should be measured for each service to provide a high‑level picture of system health:

| Golden signal | Description | Example PromQL expression【572740891080354†L82-L97】 |
| --- | --- | --- |
| **Latency** | Time to serve a request; track average, 95th and 99th percentile latencies | histogram\_quantile(0.95, sum(rate(http\_request\_duration\_seconds\_bucket[5m])) by (le, service)) |
| **Traffic** | Request rate or demand on the system | sum(rate(http\_requests\_total[5m])) by (service) |
| **Errors** | Rate of failed requests | sum(rate(http\_requests\_total{status=~"5.."}[5m])) / sum(rate(http\_requests\_total[5m])) |
| **Saturation** | Resource utilization (CPU, memory, disk, network) | sum(container\_memory\_usage\_bytes) / sum(container\_memory\_limit\_bytes) |

In addition to golden signals, track service‑specific KPIs such as request rate, error rate, latency percentiles, resource utilization, service dependencies and throughput【206002500321820†L268-L283】. Identify KPIs that align with business goals (e.g. inventory accuracy, order fulfilment SLAs).

#### Naming conventions and storage

* **Metric naming:** Use a consistent naming scheme <service\_name>\_<metric\_type>\_<unit>【206002500321820†L284-L289】. For example: inventory\_service\_error\_rate\_percent, payment\_service\_latency\_milliseconds, orders\_requests\_per\_second.
* **Label usage:** Apply labels (dimensions) such as service, endpoint, status\_code, region to enable flexible filtering and aggregation. Avoid high‑cardinality labels like individual user IDs.
* **Time‑series database:** Store metrics in a specialized time‑series database (e.g., Prometheus) for efficient querying and aggregation【206002500321820†L294-L299】. Use Prometheus’s pull‑based architecture to scrape metrics endpoints regularly【572740891080354†L36-L59】.
* **Dashboards:** Visualize metrics in Grafana or a similar tool. Design dashboards to tell a story, highlight what matters (use thresholds and colours), include context and links to runbooks, and make panels actionable【572740891080354†L104-L118】.

#### Instrumentation

* **Prometheus exporters:** Instrument each service to expose metrics on a /metrics endpoint. For JavaScript/TypeScript services, use the Prometheus client library; for .NET services, use prometheus-net. The metrics should cover request counts, latencies, database calls, cache hits, queue lengths and custom domain metrics like cycle‑count accuracy or purchase order lead time.
* **Application metrics:** Each module should collect business KPIs, for example:
  + **PIM:** number of product creations/updates per day, data load times, number of failed SKU validations.
  + **Inventory & Warehouse:** inventory discrepancies, cycle count accuracy, pick/pack/ship durations, cross‑dock transfer latency.
  + **Orders & RMA:** orders processed per minute, error rate, RMA issuance rate, refund processing time.
  + **Purchasing & Vendor:** purchase orders created, approval cycle times, vendor on‑time delivery rate.
  + **Shipping & Hazmat:** shipments per carrier, failed label generation events, hazmat compliance errors.
  + **Tax & Reporting:** tax API call latency, error rate, number of jurisdictions handled.
  + **Accounting Sync:** synchronization latency, number of failed journal entries, backlog of pending transactions.
* **Aggregation & visualization:** Use Prometheus queries and Grafana dashboards to aggregate metrics across instances and display at‑a‑glance service health【206002500321820†L307-L311】.
* **Alert thresholds:** Define alert thresholds on key metrics to detect anomalies before they impact users【206002500321820†L320-L321】. For example, set an alert if the 99th‑percentile latency exceeds 1 second for more than 5 minutes, or if the error rate is above 5%【572740891080354†L171-L183】.

### Tracing best practices

Distributed tracing captures the journey of a request as it passes through multiple microservices. It helps to pinpoint latency bottlenecks and understand service dependencies【206002500321820†L323-L333】.

* **OpenTelemetry:** Use OpenTelemetry for vendor‑neutral tracing. Automatic instrumentation can capture incoming HTTP requests, database calls and external requests with minimal setup【206002500321820†L337-L356】. Manual instrumentation should be added for critical business operations to record attributes like order ID, customer ID and amount【206002500321820†L358-L382】.
* **Trace context propagation:** Ensure trace context (trace ID, span ID, sampled flag) is propagated via HTTP headers (traceparent, X‑B3‑TraceId) across service boundaries【206002500321820†L407-L419】. This maintains a complete picture of each request’s journey.
* **Sampling strategies:** In high‑volume systems, capture a percentage of traces rather than all. Use rate‑based sampling, adaptive sampling that increases sampling during low traffic, tail‑based sampling that focuses on slow outliers, and error‑based sampling that always records failed requests【572740891080354†L241-L249】. Adjust sampling rates per service according to business criticality.
* **Correlation with logs:** Include trace IDs and span IDs in logs to connect log entries with traces【206002500321820†L211-L256】. This correlation helps cross‑reference logs and traces during troubleshooting.

## Alerting strategy

Effective alerting transforms monitoring from a passive dashboard to an active tool. The design principles below should guide alert configuration:

1. **Actionable:** Every alert should require human intervention; ignore non‑actionable noise【572740891080354†L155-L165】.
2. **Precise:** Alerts must clearly identify what is wrong and where to look【572740891080354†L160-L162】.
3. **Context‑rich:** Include metadata like affected service, endpoint, error rate, correlation ID and links to logs/dashboards【572740891080354†L160-L163】.
4. **Prioritized:** Differentiate critical versus non‑critical issues using severity levels【572740891080354†L160-L163】. Use P1, P2 and P3 tiers【572740891080354†L187-L200】.
5. **Tested:** Regularly test alerts through chaos engineering to ensure they fire appropriately【572740891080354†L155-L165】.

### Alert severity tiers

| Severity | Description | Examples【572740891080354†L187-L200】 | Response |
| --- | --- | --- | --- |
| **P1 (Critical)** | Immediate attention; wakes people up. Indicates a service outage, data loss or security breach. | Service outage, unbounded queue growth, failed tax service causing order processing halts. | On‑call engineer investigates immediately, follow incident management procedure. |
| **P2 (High)** | Requires attention during business hours. Indicates degraded performance, approaching capacity limits or a non‑critical component failure【572740891080354†L187-L199】. | Error rate > 5% for 5 minutes【572740891080354†L171-L183】, inventory accuracy drops below 98%, slow purchase order approval queue. | Engineering team investigates within 1 hour; plan remediation. |
| **P3 (Low)** | Non‑urgent; next business day response. Warns of potential issues or technical debt. | Low stock threshold reached (reorder required), increasing latency trend but within SLO, minor integration failures. | Add to backlog; schedule improvements. |

### Alert definitions by module

The tables below define the primary alerts to be implemented per module. Each alert lists the metric or event, threshold, severity, and recommended response/runbook.

#### Product Information Management (PIM)

| Alert | Metric / Event | Threshold | Severity | Response |
| --- | --- | --- | --- | --- |
| **Product sync failure** | Error rate in PIM service | Error rate > 5% for 5 min【572740891080354†L171-L183】 | P2 | Investigate integration with B2B portal and update product queue; roll back last deployment if necessary. |
| **Missing data fields** | Validation failures when creating/updating products | > 50 failed records in 10 min | P3 | Notify data steward; review import files and correct missing attributes. |
| **Slow product import** | Average latency of product import endpoint | > 2 s for 95th percentile for 5 min | P2 | Check database performance; scale PIM microservice; inspect trace for bottlenecks. |

#### Inventory & Warehouse

| Alert | Metric / Event | Threshold | Severity | Response |
| --- | --- | --- | --- | --- |
| **High inventory error rate** | inventory\_service\_error\_rate\_percent【206002500321820†L274-L283】 | > 5% for 5 min | P2 | Investigate failing inventory transactions; review logs with correlation ID for root cause. |
| **Cycle count accuracy below target** | Cycle count variance (1 – accuracy) | Accuracy < 98% for 2 consecutive cycles | P2 | Trigger a recount; audit bin counts; verify scanning equipment. |
| **Stock out / low stock** | Quantity on hand vs reorder point | Quantity < reorder point | P3 | Generate purchase order; notify purchasing team. |
| **RF scan latency high** | 99th percentile latency for RF scanning API | > 1 s for 5 min | P2 | Review wireless network; scale scanning API; check device firmware. |

#### Orders & RMA

| Alert | Metric / Event | Threshold | Severity | Response |
| --- | --- | --- | --- | --- |
| **Order processing error rate** | order\_service\_error\_rate\_percent | > 5% for 5 min【572740891080354†L171-L183】 | P1 | Immediately check payment gateway and inventory verification; roll back deployment if due to recent change. |
| **Order latency breach** | 99th‑percentile latency for order API | > 2 s for 5 min | P2 | Analyze traces to identify slow spans; scale service or database; check for contention. |
| **RMA request surge** | RMA requests per hour | > 2× rolling 7‑day average | P2 | Investigate quality issues; review manufacturing/packing processes. |
| **Refund processing delay** | Average time to refund after RMA closure | > 48 hours | P3 | Notify finance; review accounting sync and payment gateway. |

#### Purchasing & Vendor Management

| Alert | Metric / Event | Threshold | Severity | Response |
| --- | --- | --- | --- | --- |
| **PO approval backlog** | Number of purchase orders pending approval | > 20 POs pending > 2 days | P3 | Remind purchasing approvers; adjust approval routing; evaluate procurement workflow. |
| **Vendor lead time exceeded** | Average vendor lead time vs agreed SLA | > 10% above vendor SLA for 3 consecutive orders | P2 | Contact vendor; update vendor scorecard; adjust reorder points. |
| **ASN mismatch** | Inbound shipments received without matching ASNs | > 5 shipments/day | P2 | Investigate vendor compliance; coordinate receiving team; update vendor guidelines. |

#### Shipping & Hazmat

| Alert | Metric / Event | Threshold | Severity | Response |
| --- | --- | --- | --- | --- |
| **Label generation failure** | Error rate when generating shipping labels | > 5% for 5 min | P2 | Check carrier API; verify API credentials; switch to backup carrier. |
| **Hazmat compliance error** | Hazmat documentation validation failures | Any failure | P1 | Block shipment; review hazardous materials classification and packaging; ensure compliance with DOT/IATA regulations【79366541072420†L13-L27】. |
| **Shipping latency** | 95th‑percentile time from order ready to shipment | > 24 h for 5% of orders | P2 | Investigate warehouse throughput; check cross‑dock/3PL delays; adjust staffing. |

#### Tax & Reporting

| Alert | Metric / Event | Threshold | Severity | Response |
| --- | --- | --- | --- | --- |
| **Tax API failure** | Tax service availability | Unavailable or error rate > 1% for 5 min | P1 | Switch to backup tax provider; retry with exponential backoff; notify finance. |
| **Tax calculation latency** | Average latency of tax calculation requests | > 200 ms for 5 min | P2 | Check tax service network; optimize tax calculation client; contact provider. |
| **Reporting backlog** | Number of reports not generated | > 10 pending reports for > 1 day | P3 | Investigate data warehouse ETL; review scheduler; increase resources. |

#### Accounting Sync

| Alert | Metric / Event | Threshold | Severity | Response |
| --- | --- | --- | --- | --- |
| **GL sync failure** | Number of failed journal entries | > 10 failures in 1 hour | P2 | Check accounting connector; inspect logs for validation errors; retry sync. |
| **Sync latency high** | 95th‑percentile latency to sync transactions to accounting | > 5 minutes for 5 min | P2 | Check message queue backlog; verify network connectivity; scale sync worker. |
| **Unreconciled transactions** | Transactions pending reconciliation | > 50 pending for > 1 day | P3 | Alert finance team; investigate mismatch between ERP and accounting system. |

## Implementation guidelines

1. **Tooling:** Use Prometheus and its AlertManager for metrics scraping and alerting; Grafana for dashboards; the EFK stack for logs【572740891080354†L251-L260】; and OpenTelemetry for tracing【206002500321820†L337-L356】.
2. **Documentation:** Maintain runbooks for each alert, with steps to diagnose and resolve issues. Link runbooks directly from alert annotations【572740891080354†L171-L183】.
3. **Integration:** Feed alerts into the organisation’s incident management system (e.g., PagerDuty, Slack) with severity routing rules (P1 triggers immediate paging).
4. **Review and tuning:** Regularly review alert thresholds and dashboard panels. Use post‑mortems and chaos engineering exercises to refine what is measured and how alerts fire.
5. **Continuous improvement:** Observability is iterative. Expand metrics and tracing instrumentation as the system evolves. Use the collected data to refine SLOs and capacity planning.

## Conclusion

By implementing structured logging, comprehensive metrics and distributed tracing, and by adopting thoughtful alert design, PineCone Pro gains deep visibility into its ERP/IMS. This observability strategy facilitates rapid diagnosis of issues, proactive capacity management, and continuous improvement of service quality. Aligning alerts with SLOs ensures that the team focuses on what matters most to business outcomes, thereby delivering a stable and performant system that meets customer expectations.