Application Log Monitoring System

Product Requirements Document (PRD)

Version: 1.0
Date: June 26, 2025
Document Owner: Ecosystem Engineering [Workloads, App Dev, Openshift AI]
Status: Draft
Author: Mriganka Paul [Senior Principal Software Engineer]

---

1. Executive Summary

The Application Log Monitoring System empowers OpenShift application administrators to create sophisticated, natural language-based log alerting rules that trigger intelligent notifications. By leveraging advanced pattern recognition and contextual analysis, this product transforms traditional log monitoring from reactive manual searches into proactive, intelligent alerting that reduces MTTR and improves system reliability.

Key Value Propositions

- **Natural Language Alert Creation**: Describe complex log patterns in plain English
- **Contextual Intelligence**: Multi-dimensional alerting based on severity, frequency, content, and temporal patterns
- **Intelligent Paging**: Smart escalation and routing based on alert criticality and on-call schedules [Basic Paging, complex escalation may be ambitious for a blueprint]
- **Openshift-Native**: Deep integration with K8s logging infrastructure and metadata
- **Metrics Integration:** Integrating logs + alerts + metrics + traces with AI technologies (via MCP) could help tremendously to diagnose and solve issues

---

2. Product Overview

2.1 Solution Description

The Application Log Monitoring System enables administrators to create intelligent alerts using natural language expressions such as:

- "Page me if there are more than 10 ERROR logs from the payment service in 5 minutes"
- "Alert on-call if any pod shows 'OutOfMemory' errors during business hours"

Draft

- "Send critical page if database connection failures spike above normal baseline"
- "Notify team lead if authentication service shows unusual error patterns"

The system processes these requests using advanced NLP, applies them to real-time log streams from Openshift clusters, and delivers intelligent notifications through multiple paging channels.

2.2 Target Users

- **Primary**: Openshift application administrators and SREs
- **Secondary**: Development team leads requiring application health monitoring
- **Tertiary**: Platform engineers managing multi-tenant K8s environments

---

3. Goals and Success Metrics [Modify as needed for Blueprint]

3.1 Primary Goals

- **Reduce Mean Time to Resolution (MTTR)**: Decrease incident response time by 50%
- **Minimize Alert Fatigue**: Reduce false positive alerts by 70%
- **Improve System Reliability**: Increase application uptime through proactive monitoring
- **Enhance Developer Productivity**: Enable self-service alerting without deep logging expertise
- **Self-healing / Suggest solution:** Purpose solutions if the error is well-known. (Example: Missing SCC binding to workload's SA.)

3.2 Key Performance Indicators (KPIs)

- **Alert Accuracy**: <3% false positive rate for user-created alerts
- **Response Time**: 95% of critical alerts delivered within 30 seconds
- **Coverage**: Monitor 100% of application pods with zero blind spots
- **MTTR Improvement**: 50% reduction in average incident resolution time

3.3 Success Metrics

- **Technical**: 99.95% alert delivery reliability, <5 second log processing latency
- **Business**: 30% reduction in production incidents, 25% improvement in SLA compliance
- **User Experience**: 90% of alerts created successfully on first attempt

---

Draft

4. User Stories and Use Cases

4.1 Core User Stories

Epic 1: Alert Creation and Configuration

- **US-001**: As an app admin, I want to describe log alert conditions in natural language so that I can create monitoring rules without learning complex query syntax
- **US-002**: As an SRE, I want to set up alerts based on log severity, frequency, and content patterns so that I can catch issues before they impact users
- **US-003**: As a platform engineer, I want to create template alerts that can be applied across multiple applications so that teams have consistent monitoring baselines

Epic 2: Contextual and Intelligent Alerting

- **US-004**: As an app admin, I want alerts that consider deployment events and application state so that I don't get paged during planned maintenance
- **US-005**: As an SRE, I want baseline-aware alerting that adapts to normal application behavior so that I only get notified of genuine anomalies
- **US-006**: As a team lead, I want cross-service correlation in alerts so that I understand the full impact of incidents [Ambititious]

Epic 3: Paging and Escalation [Basic Escalation could be incorporated for a blueprint]

- **US-007**: As an on-call engineer, I want intelligent paging that routes alerts based on severity and my availability so that critical issues reach the right person, or scales the alert to upper level if there’s delay on the response
- **US-008**: As a team manager, I want escalation workflows that automatically promote unacknowledged alerts so that nothing falls through the cracks
- **US-009**: As an app owner, I want alert suppression during deployments and maintenance windows so that I'm not paged for expected issues

---

Draft

5. Functional Requirements

5.1 Alert Creation and Management

5.1.1 Natural Language Processing

Requirement ID	Feature	Description	Priority
FR-001	NLP Alert Parser	Parse natural language into executable log queries	P0
FR-002	Syntax Validation	Validate and provide feedback on alert feasibility	P0
FR-003	Query Preview	Show generated query and test with historical logs	P0
FR-004	Auto-Suggestions	Provide intelligent suggestions during alert creation	P1

5.1.2 Alert Configuration

Requirement ID	Feature	Description	Priority
FR-005	Multi-Condition Alerts	Support complex AND/OR logic in single alerts	P0
FR-006	Temporal Windows	Configure time-based aggregation and thresholds	P0
FR-007	Severity Mapping	Map log levels to alert priorities automatically	P0

Draft

Requirement ID	Feature	Description	Priority
FR-008	Namespace Filtering	Scope alerts to specific K8s namespaces or labels	P0

5.2 Log Processing and Analysis

5.2.1 Real-Time Processing

Requirement ID	Feature	Description	Priority
FR-009	Stream Processing	Process logs in real-time with <5 second latency	P0
FR-010	Pattern Recognition	Identify recurring patterns and anomalies	P0
FR-011	Content Analysis	Parse structured and unstructured log content	P0
FR-012	Correlation Engine	Correlate events across multiple services and pods	P1

5.2.2 Contextual Intelligence

Requirement ID	Feature	Description	Priority
FR-013	Deployment Awareness	Suppress alerts during known deployment windows	P0
FR-014	Baseline Learning	Learn normal behavior patterns to reduce false positives	P1
FR-015	Service Dependency	Understand service relationships for cascade detection	P1

Draft

Requirement ID	Feature	Description	Priority
FR-016	Resource Correlation	Correlate log patterns and workload metrics with resource utilization	P1

5.3 Paging and Notification System

5.3.1 Multi-Channel Paging

Requirement ID	Feature	Description	Priority
FR-018	Slack Notifications	Rich Slack alerts with actionable buttons	P0
FR-020	Custom Webhooks	Support for custom notification endpoints	P1

5.3.2 Intelligent Routing and Escalation [Basic Escalation could be incorporated for a blueprint]

Requirement ID	Feature	Description	Priority
FR-022	Severity-Based Routing	Route alerts based on criticality and team responsibility	P0
FR-023	Auto-Escalation	Automatic escalation for unacknowledged critical alerts	P0
FR-024	Alert Grouping	Group related alerts to prevent notification storms	P1

Draft

5.4 Openshift Integration

5.4.1 Native K8s Features

Requirement ID	Feature	Description	Priority
FR-025	Log Collection	Openshift Logging vector	P0
FR-026	Metadata Enrichment	Enrich logs with pod, service, and deployment metadata	P0
FR-027	RBAC Integration	Respect K8s RBAC for alert access and configuration	P0
FR-028	Multi-Cluster Support	Monitor logs across multiple K8s clusters	P1 [Ambititious]

5.5 Testing and Validation

5.5.1 Alert Testing Framework

Requirement ID	Feature	Description	Priority
FR-029	Historical Testing	Test alerts against historical log data	P0
FR-030	Synthetic Log Injection	Inject test log events to validate alert behavior	P0
FR-031	A/B Testing	Test different alert configurations simultaneously	P1
FR-032	Performance Testing	Validate alert processing under high log volumes	P0

Draft

6. Non-Functional Requirements [Review: Align scope for blueprint]

6.1 Performance Requirements

- **Log Processing**: Handle 100K+ log events per second per cluster
- **Alert Latency**: Critical alerts delivered within 30 seconds of trigger condition
- **Query Performance**: Alert condition evaluation within 1 second
- **Scalability**: Linear scaling with cluster and log volume growth

6.2 Reliability Requirements

- **Availability**: 99.95% uptime for alert processing and delivery
- **Fault Tolerance**: Continue operation with individual component failures

6.3 Security Requirements

- **Authentication**: Integration with enterprise SSO and K8s RBAC [Ambition]
- **Authorization**: Fine-grained permissions for alert configuration and viewing
- **Audit Trail**: Complete audit log of all alert configurations and modifications

6.4 Observability Requirements

- **Monitoring**: Full observability of the monitoring system itself
- **Metrics**: Detailed metrics on alert performance and accuracy
- **Logging**: Comprehensive logging of system operations and decisions
- **Alerting**: Meta-alerts for monitoring system health and performance

---

7. Technical Specifications [To be refined]

7.1 System Architecture

7.1.1 Core Components

Component	**Technology**	**Justification**
Log Ingestion	Openshift Logging Vector	Industry standard K8s log collection
Stream Processing	Apache Kafka + Apache Flink	High-throughput real-time processing

Draft

Component	Technology	Justification
NLP Engine	Transformer-based models	Advanced natural language understanding
Time Series DB	InfluxDB / Prometheus	Optimized for time-series log data
Alert Engine	Custom Go service	High-performance alert evaluation

7.1.2 Frontend and API

Component	Technology	Justification
Web UI	React + TypeScript + PatternFly	Modern, maintainable user interface
API Gateway	RedHat 3 scale	Authentication, rate limiting, routing
Backend API	Go with Gin framework	High-performance API services
Real-time Updates	WebSocket + Server-Sent Events	Live alert status updates

7.1.3 Integration Layer

Component	Technology	Justification
K8s Integration	Openshift Go Client	Native K8s API integration
Paging Services	Slack API	Multi-channel notification delivery
Metrics Collection	Prometheus + Grafana	System observability and monitoring
Configuration Storage	etcd	Distributed configuration management

Draft

7.2 Data Architecture

7.2.1 Data Flow

1. **Log Collection**: Vector agents collect logs from K8s pods
2. **Stream Ingestion**: Logs are sent to Kafka topics partitioned by namespace
3. **Real-time Processing**: Flink processes log streams and evaluates alert conditions
4. **Alert Triggering**: Matching conditions trigger alerts through the alert engine
5. **Notification Delivery**: Alerts are routed through appropriate channels
6. **Feedback Loop**: Alert outcomes are fed back to improve accuracy

8.2 Key User Interfaces

8.2.1 Alert Creation Wizard

1. **Natural Language Input**: Large text area with smart suggestions
2. **Scope Selection**: Namespace, service, and pod filtering options
3. **Severity Configuration**: Map alert conditions to notification urgency
4. **Testing Interface**: Validate alerts against recent log history
5. **Notification Setup**: Configure channels and escalation rules
6. **Exception definition:** Define time-widnows when a deployment is being updated and we want to pause the notifications

8.2.2 Alert Dashboard

- **Alert Status Grid**: Real-time status of all configured alerts
- **Recent Triggers**: Timeline of recent alert activations
- **Performance Metrics**: Alert accuracy and response time statistics
- **Log Explorer**: Integrated log search and analysis tools

8.2.3 Incident Response Interface

- **Alert Details**: Full context including correlated events and metrics

10. Integration Requirements

10.1 Openshift Ecosystem

- **Log Aggregation**: Vector, Filebeat compatibility[Choose as required]
- **Service Mesh**: Openshift Service Mesh and Linkerd(????? May be for ease) log integration
- **Monitoring**: Prometheus metrics and Grafana dashboard integration
- **Security**: OPA (Open Policy Agent) policy integration [May be too much for a blue print- Ambititious]

Draft

10.2 External Systems

- **Communication**: Slack
-

10.3 CI/CD Pipeline Integration

- **GitOps**: Alert configurations stored in Git repositories
- **Deployment Hooks**: Automatic alert suppression during deployments [Ambititious]
- **Testing Integration**: Alert validation in CI/CD pipelines
- **Infrastructure as Code**: Terraform/Ansible and Helm chart deployment

---

11. Risk Assessment and Mitigation

11.1 Technical Risks

Risk	**Impact**	**Probability**	**Mitigation Strategy**
High log volume overwhelming system	High	Medium	Auto-scaling, circuit breakers, backpressure handling
NLP misinterpretation causing false alerts	Medium	Medium	Extensive training data, validation framework, user feedback
Single point of failure in alert delivery	High	Low	Redundant delivery channels, circuit breakers

Draft

14. Appendices

14.1 Example Alert Configurations

Basic Error Rate Alert

```
None
Natural Language: "Page me if the payment service has more than 50 ERROR logs
in 5 minutes"

Generated Query:
{
  "conditions": {
    "logLevel": ["ERROR"],
    "serviceFilters": ["payment-service"],
    "frequency": {
      "threshold": 50,
      "timeWindow": "5m"
    }
  },
  "severity": "high",
  "channels": ["slack"]
}
```

Complex Correlation Alert

```
None
Natural Language: "Critical page if database connection errors AND API response
time spikes above 2 seconds"

Generated Query:
{
  "conditions": {
    "correlationRules": [
      {
        "condition": "contentPatterns",
        "patterns": ["database connection failed", "connection timeout"],
        "frequency": {"threshold": 5, "timeWindow": "2m"}
      },
      {
        "condition": "metricCorrelation",
        "metric": "response_time_p95",
        "threshold": 2000,
        "operator": "gt"
```

```
      }
    ],
    "logic": "AND"
  },
  "severity": "critical",
  "channels": ["webhook", "slack"]
}
```

Metrics + Logs relation

None

```
Natural Language: "Notice me when a workload is reaching its Memory Limit and
returning OOM"

Generated Query:
{
  "conditions": {
    "logLevel": ["ERROR"],
  },
  "correlationRules": [
      {
        "condition": "contentPatterns",
        "patterns": ["OOM", "OutOfMemory"],
        "frequency": {"threshold": 5, "timeWindow": "2m"}
      },
      {
        "condition": "metricCorrelation",
        "metric": "(
               sum by (pod, namespace) (
                 container_memory_usage_bytes{container!="", pod!=""}
               )
               /
               sum by (pod, namespace) (
                 container_spec_memory_limit_bytes{container!="", pod!=""}
               )
             ) * 100
             ",
        "threshold": 95,
        "operator": "gt"
      }
    ],
```

```
    "logic": "AND"
  },
  "severity": "critical",
  "channels": ["webhook", "slack"]
}
```

14.2 Natural Language Alert Patterns for Log Monitoring

Category 1: Basic Threshold Alerts (Count-based)

Error Rate Patterns

1. "Page me if there are more than 10 ERROR logs from the payment service in 5 minutes"
2. "Alert when user-service shows over 50 WARN messages per hour"
3. "Notify if database service has any FATAL errors in the last 10 minutes"
4. "Send critical alert if authentication-api exceeds 25 ERROR logs in 2 minutes"
5. "Page on-call if more than 100 4xx responses from api-gateway in 15 minutes"

Service-Specific Patterns

6. "Alert if checkout-service logs contain more than 5 'payment failed' messages in 10 minutes"
7. "Page me when inventory-service shows over 20 'stock unavailable' errors per hour"
8. "Notify team if email-service has more than 3 'SMTP connection failed' in 5 minutes"
9. "Critical page if redis-cache service logs 'connection refused' more than 2 times in 1 minute"
10. "Alert when message-queue shows over 15 'queue full' warnings in 30 minutes"

Category 2: Content Pattern Matching

Keyword and Phrase Detection

11. "Page me if any pod shows 'OutOfMemory' errors during business hours (9 AM - 6 PM)"
12. "Alert on-call if database connection failures spike above normal baseline"
13. "Notify team lead if authentication service shows 'token expired' more than 50 times in 1 hour"
14. "Critical alert if any service logs 'security breach' or 'unauthorized access'"
15. "Page if SSL certificate errors appear in any ingress controller logs"

Performance-Related Content

16. "Alert when any service logs 'timeout' more than 10 times in 5 minutes"
17. "Page me if 'slow query' appears in database logs more than 5 times per hour"
18. "Notify if 'circuit breaker open' shows up in any microservice logs"
19. "Alert when 'rate limit exceeded' appears more than 20 times in 10 minutes"

Draft

20. "Critical page if 'deadlock detected' appears in any database service"

Category 3: Temporal and Conditional Logic

Time-Based Conditions

21. "Alert only during weekends if backup-service shows any ERROR logs"
22. "Page me between 2 AM - 4 AM if maintenance-job fails"
23. "Notify during business hours if customer-service response time exceeds 5 seconds"
24. "Alert outside business hours if security-scanner finds critical vulnerabilities"
25. "Page immediately if payment processing fails during Black Friday (Nov 24-25)"

Multi-Condition Logic

26. "Critical page if database connection errors AND API response time spikes above 2 seconds"
27. "Alert if memory usage > 85% AND garbage collection logs show 'full GC' more than 3 times in 5 minutes"
28. "Page when disk space < 10% OR I/O errors appear in storage service logs"
29. "Notify if CPU usage > 90% AND application logs show 'thread pool exhausted'"
30. "Alert if both primary and secondary database show connection failures within 1 minute"

Category 4: Deployment and Change-Related

Deployment Awareness

31. "Skip alerts for payment-service during deployment windows (tagged with 'maintenance')"
32. "Alert only if ERROR logs appear 10 minutes after deployment completion"
33. "Page if rollback occurs AND new ERROR patterns emerge in any service"
34. "Notify if post-deployment health checks fail in any critical service"
35. "Critical alert if canary deployment shows 5x more errors than stable version"

Configuration Changes

36. "Alert when configuration reload fails in any service"
37. "Page if 'config validation error' appears after any ConfigMap update"
38. "Notify if feature flag changes cause increase in error rates by >200%"
39. "Alert when environment variable changes trigger service restarts >3 times in 1 hour"

Category 5: Cross-Service and Correlation

Service Dependency Patterns

40. "Page if payment-service AND order-service show errors simultaneously"
41. "Alert when downstream service failures cascade to more than 2 dependent services"
42. "Notify if external API failures affect more than 3 internal microservices"
43. "Critical page if user-authentication fails AND affects checkout, profile, AND support services"

Business Impact Correlation

44. "Alert if shopping cart abandonment logs increase by >50% during peak hours"
45. "Page when user registration failures spike during marketing campaign periods"

Draft

46. "Notify if search service errors correlate with >25% drop in product page views"

Category 6: Resource and Infrastructure

Resource Exhaustion

47. "Critical page if any pod shows 'OOMKilled' status in logs"
48. "Alert when persistent volume claims show 'disk full' errors"
49. "Page if container restart count exceeds 5 times in 10 minutes for any critical service"
50. "Notify if node resource pressure causes pod evictions in production namespace"

15. High Level Development Plan

Strategic approach: Build the core and expand incrementally

Phase0: Foundation and proof of concept

- Goal: Validate the concept with minimal viable alerting
- Components to build
    - Simple Log Ingestion - Openshift Logging Vector + Kafka pipeline
    - Basic Alter Engine - Simple rule based alerting(no NLP engine)
    - Single notification channel: Slack integration only
    - Minimal UI: Command Line or simple web form for alert creation
- Deliverables
    - Working log pipeline from openshift pods to kafka
    - Basic threshold based alerts (count, keyword matching)
    - Slack notification working
    - Single hard coded alert rule validation
- Success Criteria
    - Can detect “ERROR” logs exceeding count threshold
    - Can send slack notification within 30 seconds
    - No data loss in log pipeline

Phase1: MVP - Core Alerting Engine

- Goal: Production ready basic alerting without NLP complexity
- Milestone 1.1: Log processing pipeline
    - Components
        - Log collection - Openshift logging vector
        - Stream processing - Kafka + Kafka streams (simpler than Flink initially)
        - Basic Data storage: In memory state store for alerts
    - Technical Focus

Draft

        - Reliable log ingestion from k8s pods
        - Kafka topic design(partition by namespace/service)
        - Basic log parsing and enrichment with k8s metadata
        - Simple alert rule engine (JSON-based rules)
    - Validation
        - Handle 10k logs/second without loss
        - Enrich logs with pod/service/namespace metadata
        - Basic filtering and counting operations
- Milestone 1.2: Alert Rule Engine
    - Components
        - Rule definition: JSON based alert rules (pre NLP)
        - Evaluation Engine: Real time rule evaluation
        - State management: Track alert states and cooldowns
        - Testing framework: Validate rules against historical data
- Milestone 1.3: Notification System
    - Components
        - Slack Integration: Rich notification with context
        - Alert Routing: Basic severity based routing
        - Delivery Reliability: Retry logic and failure handling

Phase2: Intelligence and User Experience

- Goal: Add natural language processing and improved UX
- Milestone 2.1 : Natural Language Processing
    - Components
        - NLP parser: Convert natural language to JSON rules
        - Validation Engine: Validate feasibility of natural language requests
        - Query preview: Show generated rules and test historical data
    - Approach
        - Start with template based parsing (pattern matching)
        - Use libraries like spaCy for entity extraction
        - Build rule templates for common patterns
        - **Progressive enhancement toward ML based parsing**
    - Example Patterns
        - 0"Page me if [service] has more than [X] [LOG_LEVEL] logs in [time]"
        - "Alert when [pattern] appears in [service] logs"
        - "Notify if [metric] exceeds [threshold]"
- Milestone 2.2: Web UI development
    - Components
        - Alert creation wizard: Natural language input with suggestions
        - Alert Dashboard: Status monitoring and management
        - Testing interface: Historical log validation
- Milestone 2.3 Historical Testing and Validation

Draft

- Components
  - Synthetic testing: Inject test events
  - Performance validation: Load testing with high log volume
  - Log Replay systemL Test alert against historical data

Phase3: Production Hardening [TBD]

Phase4: Advanced Features [TBD]

Draft