Log Aggregation for Cloud-Native Applications

Koyyada Vishnu Vardhan

Prof. Dr.-Ing. Jens Ehlers

June 2025

Agenda

1. Why Logging Matters
2. Log Lifecycle in Kubernetes
3. Log Aggregation Stacks
4. Experimental Setup
5. Performance Comparison
6. Recommendations

Importance of Logs

- **Observability**: Crucial for observing application behavior in real time
- **Debugging**: Helps trace API calls and identify issues
- **Performance**: Measures system performance metrics
- **Security**: Essential for auditing and compliance
- **Traceability**: Enables system-wide tracking in microservices

Key Insight

Without proper logging, root cause analysis becomes guesswork

How Logs Flow in Kubernetes

Applications write logs to `stdout` or `stderr`.

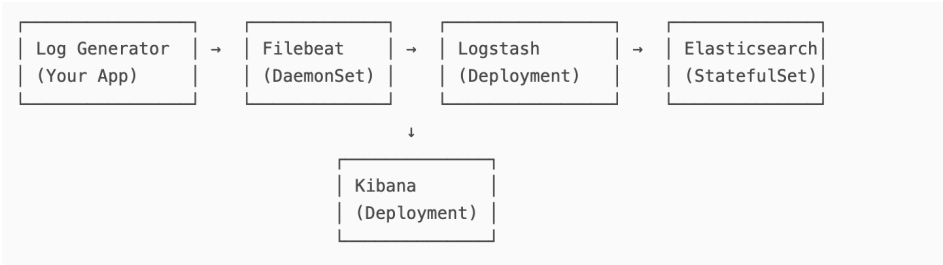
- Container runtime (e.g., containerd, CRI-O) captures the output
- Logs stored as plain text files on the node:
  - `/var/log/containers/`
  - `/var/log/pods/`
- Kubelet reads these logs and handles rotation
- `kubectl logs <pod>` reads from these local log files

Critical Limitation

- Kubernetes **does not** aggregate or persist logs
- If a pod crashes or is rescheduled, logs on the old node are **lost**

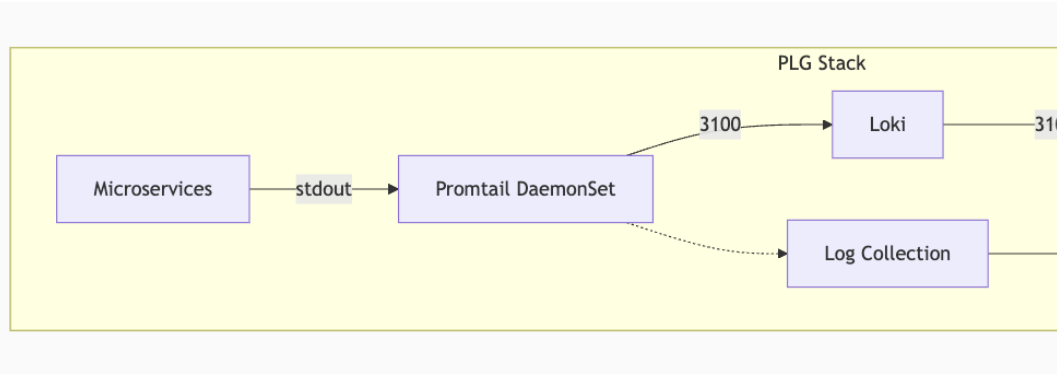
ELK Stack Components

- **Elasticsearch**:
  - Distributed search and analytics engine
  - Stores data in JSON documents
- **Logstash**:
  - Processes logs with custom pipelines
  - JSON parsing and field transformation
- **Kibana**: Visualization interface
- **Beats**:
  - Lightweight data shippers
  - Platform for specialized data collection
    - *Filebeat*: Log file collection
    - *Heartbeat*: Uptime monitoring



PLG Stack Components

- **Promtail**:
  - Lightweight log collector (DaemonSet)
  - Discovers and tags Kubernetes pod logs
  - Adds metadata (pod labels, namespace)
  - Pushes to Loki via gRPC/protobuf
- **Loki**:
  - Horizontally-scalable log store
  - Indexes by labels (not content)
  - Gzip compression for efficiency
  - Compatible with Prometheus labels
- **Grafana**:
  - Unified query interface (LogQL)
  - Correlate logs with metrics
  - Alerting and dashboard features



Test Environment

- **Cluster**: Kind
- **Nodes**: 1 control-plane, 3 workers
- **Namespaces**: Separate for ELK and PLG deployments
- Avg Log Generations stats by App :
  - Total: 471 logs (232.83/min)
  - Runtime: 121.4 seconds
  - Levels:
    - INFO: 339 (72.0%)
    - ERROR: 56 (11.9%)
    - WARNING: 76 (16.1%)
  - Size: 184.0 KB (400b/log)
- Log patterns:
  - Normal operations (INFO)
  - Error conditions (ERROR)
  - Debug bursts

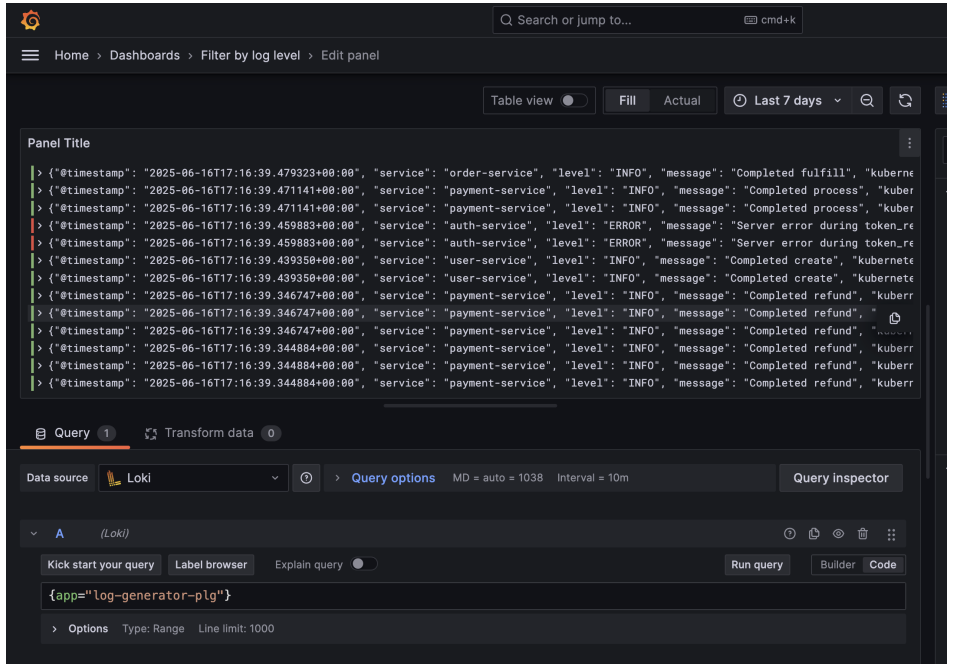


Figure: Sample Logs Generated by app

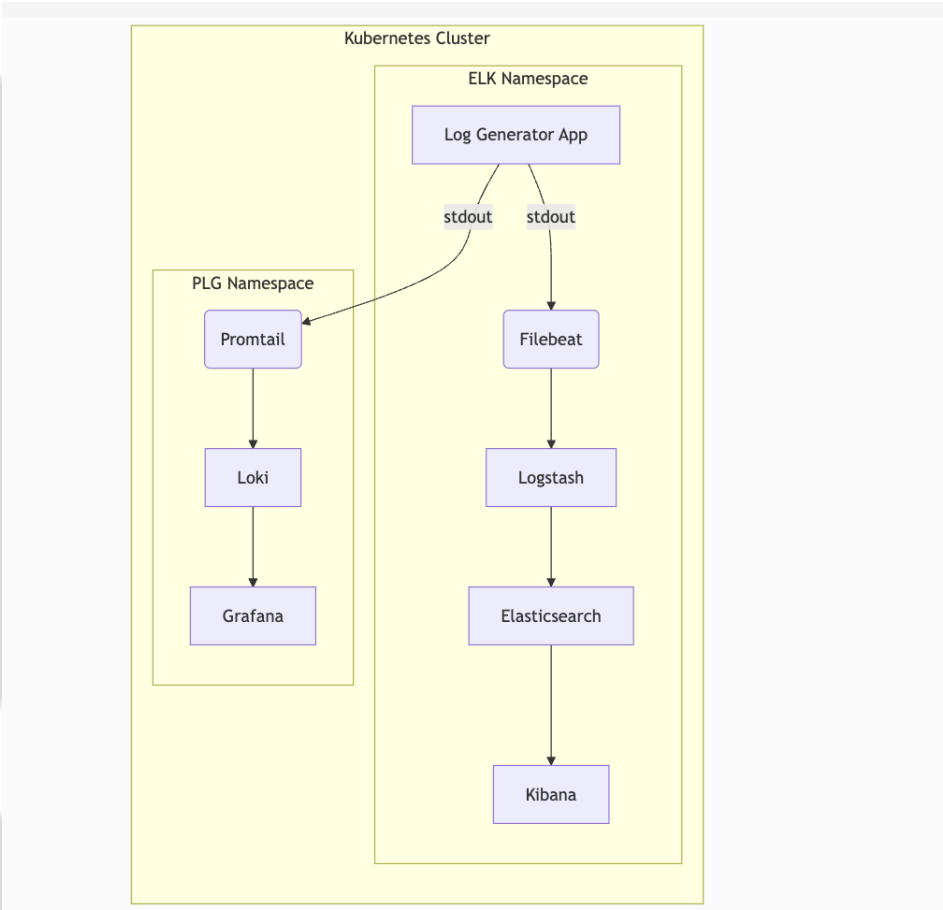
Logging Stack Deployment Architecture

ELK Stack Deployment

- **Application Pods**:
  - 3 replicas of log-generator
  - JSON logs to stdout
- **ELK Components**:
  - Filebeat DaemonSet (per node)
  - Logstash: 1 pod
  - Elasticsearch: 1 pod
  - Kibana: 1 pod
- **Services**:
  - Logstash:5044 (NodePort)
  - ES:9200, Kibana:5601

PLG Stack Deployment

- **Application Pods**:
  - 3 replicas log-generator-plg
  - Disabled Logstash output
- **PLG Components**:
  - Promtail DaemonSet (per node)
  - Loki: 1 pod
  - Grafana: 1 pod



Common Infrastructure

- Separate `logging` and `logging-plg` namespaces

ELK Stack Deployment in Kind

Core Components

- **Filebeat**:
  - DaemonSet with hostPath volumes
  - Collects from /var/log/containers
  - Adds Kubernetes metadata
  - Outputs to Logstash:5044
- **Logstash**:
  - JSON parsing pipeline
  - Field renaming and cleanup
  - Daily index routing
  - NodePort service (5044)

Key Configurations

- Dedicated `logging` namespace
- Security disabled (xpack.security.enabled: false)

Elasticsearch & Kibana

- **Elasticsearch**:
  - Single-node discovery
  - HTTP service on 9200
  - Daily index rotation
  - 1GB heap size
- **Kibana**:
  - Pre-configured ES connection
  - NodePort on 5601
  - Visualization dashboards

Data Processing

- JSON log parsing with validation
- Timestamp extraction
- Kubernetes field normalization
- Index pattern: `logs-%{+YYYY.MM.dd}`

PLG Stack Deployment in Kind

Core Components

- **Promtail**:
  - DaemonSet collecting pod logs
  - Scrapes /var/log/pods and Docker containers
  - Adds Kubernetes metadata labels
  - Pushes via HTTP to Loki
- **Loki**:
  - Single-binary mode for testing
  - Filesystem storage backend
  - Label-based indexing only
  - 7-day log retention

Key Features

- Dedicated `logging-plg` namespace
- CRI-O/Docker log format support
- No authentication in dev mode
- Auto log rotation (24h index period)

Grafana Integration

- **Pre-configured**:
  - Loki datasource auto-added
  - LogQL query support
  - NodePort (3000) access
- **Correlation**:
  - Unified logs and metrics
  - Alerting capabilities
  - Custom dashboard support

Data Flow

1. Apps → stdout logs
2. Promtail → Collects + labels
3. Loki → Stores + indexes
4. Grafana → Visualizes

```
loki.logging-plg:3100
grafana.logging-plg:3000
```

Query Performance

Metric	ELK	PLG
Avg Query Time	65ms	11ms
Max Ingestion Rate	2,500 logs/s	8,000 logs/s
Simple Query Latency	1.1s $\pm$ 0.3s	320ms $\pm$ 80ms
Complex Query Latency	3.8s $\pm$ 1.2s	1.2s $\pm$ 0.4s

Table: Query performance comparison

- PLG shows 10$\times$ faster query processing (34ms vs 340ms)
- Particularly efficient for simple queries and recent logs

Resource Utilization

Metric	ELK	PLG
Memory Usage	2235MiB	281MiB
CPU Usage (avg)	2.3 cores	1.1 cores
Storage/Day	12GB	8GB
Compression Ratio	1.5×	3-5×

Table: Resource utilization comparison

- PLG uses 8× less memory than ELK
- More efficient storage with better compression

When to Choose Which Stack

Choose ELK when:

- Need advanced analytics
- Full-text search required
- Complex transformations needed
- Rich visualization essential
- Budget allows for resources

Choose PLG when:

- High-throughput needed
- Operational monitoring focus
- Resource efficiency critical scaling required
- Simplified operations preferred

Cloud-Native Advantage

PLG's efficiency provides substantial cost benefits in cloud environments

Thank You!

- Contact: `vishnu.koyyada@fh-kiel.de`
- Slides: `https://github.com/vishnukoyyada/CloudProject`