FLIP-XXX: Flink Operational Events System

- **Status**: Proposed
- **Author(s)**: Kartikey Pant
- **Created**: 2025-06-27
- **JIRA**: here (<- link to https://issues.apache.org/jira/browse/FLINK-XXXX)
- **Released**: <Flink Version>

1. Abstract

Diagnosing failures and understanding operational behavior in Flink requires deep, contextual insights that metrics alone cannot provide. This FLIP proposes a foundational **Flink Operational Events System** to emit structured, actionable events about the job lifecycle. The goal is to provide definitive, machine-readable answers to common operational questions, drastically reducing the reliance on manual log analysis.

The primary goal is to establish a configurable, pluggable mechanism, similar to Flink's `MetricReporter` system. Version 1 (V1) will implement a lightweight, asynchronous dispatch mechanism to ensure stability and protect Flink's core from potential reporter-induced latencies. It will deliver immediate value by emitting events that address critical diagnostic scenarios, such as identifying the root cause of job submission failures and pinpointing the specific subtasks that cause checkpoint timeouts. This system lays the essential groundwork for advanced external observability platforms and sets the stage for future enhancements.

2. Motivation

Operating Flink at scale requires efficient tools for root cause analysis. While Flink's metrics are effective for reactive monitoring, they fall short when trying to correlate discrete state changes to understand complex issues. This proposal addresses critical gaps by providing structured events that answer key operational questions:

- **Diagnosing Job Submission and Configuration Issues:** When a user's job fails to start, the reason is often hidden in logs. There is no centralized, auditable record of submission attempts or the exact configuration a job was started with.
- **Rapid Root Cause Analysis of Failures:** A job can fail for many reasons. An event that clearly distinguishes between a failure due to an exhausted restart-strategy versus a single catastrophic error is critical. Similarly, when a checkpoint times out, knowing *which subtask* was the culprit is the most important piece of diagnostic information.
- **Enabling Robust Automation and CI/CD:** Modern platform teams rely on automation for job upgrades. These workflows require definitive signals to know when an operation, like taking a savepoint, has completed successfully or failed.

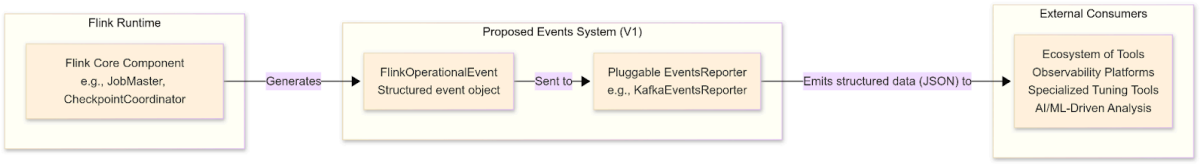
- **Standardizing Observability:** A structured event system offers a robust, versioned alternative to fragile log scraping , lowering the barrier for building powerful, customized alerting and monitoring solutions.

3. Proposed Changes: Flink EventsReporter System (V1)

We propose a new system modeled on the `MetricReporter` framework , focusing on a streamlined V1 with an asynchronous design for stability. The architecture consists of Flink components generating events, a new Events System dispatching them, and external tools consuming them.

3.1. Core Architecture and Dispatch

The architecture is designed to be safe and have minimal performance impact by decoupling Flink's core threads from reporter execution.



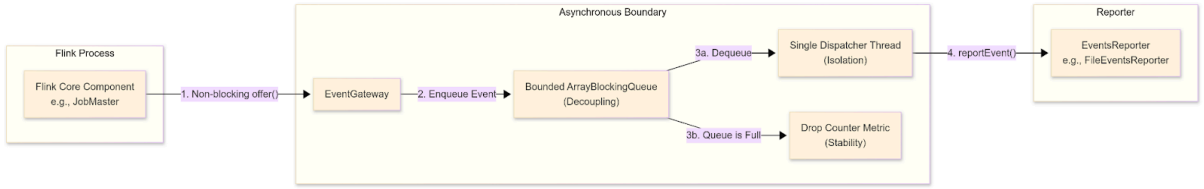
3.1.1. Event Generation and EventGateway

Core Flink components, specifically `JobMaster` for job status changes and `CheckpointCoordinator` for checkpoint events, will create `FlinkOperationalEvent` instances. An internal `org.apache.flink.runtime.events.internal.EventGateway` interface will handle submission. Events are placed onto a queue via a non-blocking `offer()` call, ensuring minimal impact on Flink's core threads.

3.1.2. Minimal EventDispatchService (V1)

This JobManager-local service decouples Flink's core threads from reporter execution.

- **Queue:** A bounded `ArrayBlockingQueue` with a configurable size (default 1024).
- **Dispatcher:** A single thread consumes events and calls `EventsReporter#reportEvent`. This choice emphasizes simplicity for V1.
- **Backpressure:** If the queue is full, the `EventGateway` increments a 'dropped' counter metric and logs the drop (rate-limited), discarding the event without blocking Flink's core threads. For V1, Flink's stability is consciously prioritized over guaranteed event delivery.
- **Scope:** Advanced features like thread pools are out of scope for V1.



3.1.3. FlinkOperationalEvent Object

Events will be a common Java object with a header and a payload. The header will contain essential contextual information like `jobId`, `jobName`, and `executionAttemptId`.

3.1.4. Structured & Versioned Format

Reporters will serialize events, primarily to JSON. V1 provides documented JSON structures and commits to maintaining backward compatibility within a minor Flink version. An `eventSchemaVersion` field will be included in the event header, though formal schema enforcement is a post-V1 feature.

3.2. V1 Event Catalog

V1 will include events to solve common diagnostic problems. Each event will share a common header containing `jobId`, `jobName`, `eventType`, `eventTimestamp`, etc..

1. **`JobSubmissionResult`**: Fired upon a job submission attempt to provide a definitive success or rejection message.
2. **`JobConfigurationSnapshot`**: Fired when a job starts, capturing its configuration for auditing.
3. **`JobStatusChanged`**: Tracks job lifecycle transitions.
    - When a job `FAILS` due to exhausting its restart strategy, the payload will be enriched with that context.
4. **`CheckpointCompleted`**: Reports successful checkpoint details like `durationMs` and `sizeBytes`.
5. **`CheckpointFailed`**: Reports failed checkpoints.
    - When a checkpoint fails due to a timeout, the payload will list the specific non-acknowledging subtasks.
6. **`SavepointCompleted / SavepointFailed`**: Fired upon completion of a savepoint operation to enable automation.

3.3. V1 Reporter Implementations

V1 introduces the public `org.apache.flink.runtime.events.reporter.EventsReporter` interface (`open`, `reportEvent`, `close`). The `reportEvent` method must be non-blocking. V1 includes:

- **`FileEventsReporter`**: Writes events as JSON to a local file. It is simple, reliable, and provides immediate usability.

- **`NoOpEventsReporter`**: A null reporter for disabling event emission or for testing.
- **`OTLPEventsReporter`**: A reporter for OpenTelemetry. This is important to ensure Flink integrates with modern observability ecosystems.

3.4. Configuration, Monitoring, and Security

- **Configuration:** The system will be configured via `flink-conf.yaml` under `observability.events.*` keys. This includes settings for the reporter implementation, queue size, and reporter-specific parameters.
- **Monitoring:** The system will expose its own metrics for monitoring, such as `eventsSubmittedCount` and `eventsDroppedCount`.
- **Security:** Centralized masking is deferred to a future version. V1 events are chosen for lower sensitivity. Any stack traces will be hashed to avoid exposing sensitive details.

4. New Public APIs

- `org.apache.flink.runtime.events.reporter.EventsReporter` interface.
- `org.apache.flink.runtime.events.reporter.FileEventsReporter` class.
- `org.apache.flink.runtime.events.reporter.NoOpEventsReporter` class.
- `org.apache.flink.runtime.events.reporter.OTLPEventsReporter` class.
- Configuration keys under `observability.events.*`.
- Documented V1 JSON event structures.

5. Compatibility, Deprecation, and Migration Plan

- **Compatibility:** This feature is purely additive and will be disabled by default, ensuring no impact on existing users.
- **Deprecation:** No components are deprecated by this FLIP.
- **Migration:** No migration is needed for existing users.
- **Performance:** There will be some performance overhead when the system is enabled, which is minimized by the asynchronous design.
- **Consumers:** Event consumers must rely on the V1 documented JSON structures.

6. Test Plan (V1)

- **Unit Tests:** Will cover event creation, the `EventGateway`, the `EventDispatchService` (including queue-full/drop behavior), serialization logic, configuration parsing, and all V1 reporter implementations.
- **Integration Tests:** End-to-end tests will validate the V1 reporters, covering job lifecycle scenarios (including submission, savepoints, and restart-strategy failures), and verify correct back-pressure behavior under a slow reporter scenario.

- **Performance Tests:** Will measure the overhead (CPU, memory, latency) of the event system to validate the non-blocking design goals, especially under high event load.
- **Documentation Conformance Tests:** Tests will be implemented to ensure the generated JSON for all V1 events precisely matches the documented structures.

7. Rejected Alternatives

- **Relying Solely on Kubernetes Operator:** External orchestrators react to symptoms (e.g., pod restarts, high CPU). This event system provides the internal, causal context *why* those symptoms occurred, which is data the operator does not have. The two systems are complementary.
- **Solely using metrics:** Metrics lack the rich, descriptive context of events needed for diagnosing the root cause of specific state transitions.
- **Log scraping:** This is fragile, expensive, and lacks standardization.
- **Synchronous dispatch:** Rejected as too risky to the stability of Flink's core components.
- **Custom binary formats:** Rejected in favor of the interoperability of JSON.

8. Future Work (Post V1)

This V1 proposal is a focused but powerful foundation. Future work can build upon it:

- **Expanded Events:** Introduce more granular events like `JobGraphSnapshot` and `TaskStatusChanged`.
- **Advanced Core Features:** Implement configurable backpressure strategies and thread pools for the dispatcher.
- **More Native Reporters:** Provide optimized reporters for Kafka and Pulsar.
- **Schema Management:** Introduce formal schemas and integration with a schema registry.
- **Security:** Implement centralized masking and sanitization features.
- **TM Event Path:** Define a secure and efficient architecture for events generated on the TaskManagers.