

# Case Event Handler Service - Architecture Report

## Executive Summary

The **Case Event Handler** is a critical microservice in Datadog's Case Management system that processes domain events asynchronously via Kafka. It implements a multi-flavor architecture with 17 specialized handlers, each responsible for specific aspects of case processing including integrations, notifications, search indexing, analytics, and automation.

## Service Overview

### Purpose

The Case Event Handler serves as the event processing backbone of the case management system, ensuring that case-related events trigger appropriate downstream actions across multiple domains and external systems.

### Key Characteristics

- **Event-Driven Architecture:** Processes domain events asynchronously via Kafka
- **Multi-Flavor Design:** 17 specialized handler flavors for different functional domains
- **High Availability:** Fault-tolerant with retry mechanisms and circuit breakers
- **Extensible:** Plugin-like architecture for easy addition of new handlers

## Core Architecture

### Event Processing Pipeline

Kafka Topics → Event Consumption → Event Decoding → Handler Routing → Processing → Acknowledgment

1. **Event Consumption:** Subscribes to Kafka topics for domain events
2. **Event Decoding:** Deserializes protobuf domain events
3. **Handler Routing:** Routes events to the configured handler flavor
4. **Processing:** Executes business logic specific to the handler
5. **Acknowledgment:** Confirms successful processing or handles errors

### Processing Modes

The service supports two distinct processing approaches:

#### Individual Processing (Processor)

- Processes events one at a time
- Immediate processing and acknowledgment
- Used by most handlers for real-time processing

## Batch Processing (`ProcessorBatch`)

- Accumulates events into batches
- Processes multiple events together for efficiency
- Used by EVP Indexer for high-throughput scenarios

## Response System

All handlers return a `HandleResp` with four possible outcomes:

- **Success** (`SuccessHandleResp`): Event processed successfully, continue
- **Skip** (`SkipHandleResp`): Event not applicable to this handler, continue
- **Retry** (`RetryErrHandleResp`): Retryable error, crash and restart for fresh state
- **Discard** (`DiscardErrHandleResp`): Non-retryable error, skip event and continue

## Handler Flavors

### 1. Timeline Handler (`timeline`)

**Purpose:** Converts domain events into timeline cells for case UI visualization.

**Functionality:** - Creates timeline entries for case lifecycle events - Handles case links, attachments, and integration events - Supports multiple cell types (lifecycle, attributes, integrations) - Provides chronological view of case activity

**Key Events:** Case creation, status changes, assignments, attribute updates, attachments

---

### 2. Notification Handler (`notification`)

**Purpose:** Sends notifications based on case events and watcher configurations.

**Functionality:** - Processes watcher subscriptions and notification rules - Integrates with Datadog Notification Platform - Supports email, Slack, and other notification channels - Handles notification preferences and filtering

**Key Events:** Case events that match watcher criteria and notification rules

---

### 3. Search Indexer Handler (`search-indexer`)

**Purpose:** Maintains Elasticsearch search index for case discovery.

**Functionality:** - Updates search index with case data changes - Enables complex case queries and filtering - Supports faceted search and full-text search - Handles incremental updates for performance

**Key Events:** All case attribute changes, status updates, assignments

---

#### **4. Jira Handler (jira)**

**Purpose:** Bidirectional synchronization with Jira issues.

**Functionality:** - Creates Jira issues for new cases - Syncs case updates to Jira (title, description, priority, status, assignee) - Maps case fields to Jira custom fields - Handles workflow state synchronization

**Key Events:** Case creation, attribute changes, comments, status transitions

---

#### **5. ServiceNow Sync Handler (snow-sync)**

**Purpose:** Bidirectional synchronization with ServiceNow tickets.

**Functionality:** - Maps case fields to ServiceNow ticket fields - Handles incremental synchronization - Supports custom field mappings - Manages workflow state transitions

**Key Events:** Case creation, status changes, comments, attribute updates

---

#### **6. Paging Handler (paging)**

**Purpose:** Creates pages (alerts) to on-call teams based on case conditions.

**Functionality:** - Evaluates escalation queries against case attributes - Integrates with Datadog On-Call service - Supports dynamic team resolution from case attributes - Auto-assigns cases to on-call members - Resolves pages when cases are closed

**Key Events:** Case creation, status changes, attribute changes, assignments

---

#### **7. Page Linking Handler (page-linking)**

**Purpose:** Links existing pages to cases when correlations are identified.

**Functionality:** - Creates timeline entries for page-case relationships - Manages page-to-case linking metadata - Supports correlation-based automatic linking

**Key Events:** Page linking events, correlation events

---

## **8. Analytic Handler (`analytic`)**

**Purpose:** Generates analytics data for case management metrics.

**Functionality:** - Tracks status transitions for dashboard metrics - Generates time-series data for case lifecycle analysis - Supports SLA tracking and performance metrics - Creates aggregated statistics

**Key Events:** Status changes, case creation, incident linking

---

## **9. EVP Indexer Handler (`evp-indexer`)**

**Purpose:** Sends case events to Event Platform for observability.

**Processing Mode:** Batch processing for high throughput

**Functionality:** - JSON serialization of events for Event Platform - Batch processing for efficiency - Supports event correlation and monitoring - Enables observability across case lifecycle

**Key Events:** Most domain events

---

## **10. EVP Indexer Feed Handler (`evp-indexer-feed`)**

**Purpose:** Alternative Event Platform integration for specific use cases.

**Functionality:** - Direct integration with Event Management v2 client - Individual event processing - Specialized event formatting and routing

**Key Events:** Domain events for event management workflows

---

## **11. Notification Rules Handler (`notification-rules`)**

**Purpose:** Processes project-level notification rules.

**Functionality:** - Evaluates notification rules against case events - Generates notification events for matched rules - Supports complex rule conditions and filtering - Enables project-specific notification policies

**Key Events:** Most case events based on rule configurations

---

## **12. Rule Evaluation Handler (`rule-evaluation`)**

**Purpose:** Evaluates and executes automation rules.

**Functionality:** - Processes automation rules defined in projects - Generates workflow events and notification events - Supports rule-based case automation - Enables custom business logic execution

**Key Events:** Case events matching automation rule conditions

---

### **13. Attributes Indexer Handler (`attributes-indexer`)**

**Purpose:** Maintains specialized indexing for case attributes.

**Functionality:** - Optimized attribute search and filtering - Supports custom attribute types and values - Enables attribute-based queries and reporting

**Key Events:** Attribute change events

---

### **14. Settings Handler (`settings`)**

**Purpose:** Handles settings-related case processing.

**Functionality:** - Processes case assignment events - Manages project-specific configurations - Handles settings-driven case routing

**Key Events:** Case assignment events, configuration changes

---

### **15. Incident Escalation Handler (`incident-escalation`)**

**Purpose:** Manages escalation of cases to incidents.

**Functionality:** - Creates incidents from high-priority cases - Manages incident-case relationships - Supports custom escalation criteria - Handles incident lifecycle management

**Key Events:** Case events triggering incident creation

---

### **16. Incident Followup Sync Handler (`incident-followup-sync`)**

**Purpose:** Synchronizes incident followup data.

**Functionality:** - Manages incident resolution workflows - Syncs followup actions and status updates - Handles post-incident case processing

**Key Events:** Incident-related events, followup events

---

## 17. Additional Specialized Handlers

The system supports additional handlers for specific integrations and use cases, following the same architectural patterns.

## Design Patterns and Principles

### 1. Event Sourcing

- All case changes are captured as immutable domain events
- Events provide audit trail and support replay capabilities
- Enables temporal queries and historical analysis

### 2. CQRS (Command Query Responsibility Segregation)

- Handlers maintain specialized read models
- Optimized data structures for specific query patterns
- Separate concerns between command and query processing

### 3. Idempotency

- All handlers support idempotent processing using event IDs
- Duplicate event detection and handling
- Ensures exactly-once processing semantics

### 4. Circuit Breaker Pattern

- Handlers distinguish between retryable and non-retryable errors
- Graceful degradation during dependency failures
- Automatic recovery and retry mechanisms

### 5. Plugin Architecture

- Loosely coupled handler implementations
- Easy addition of new handler flavors
- Standardized interfaces and contracts

## Error Handling and Resilience

### Retry Strategy

- **Retryable Errors:** Service crashes and restarts for fresh state
- **Non-Retryable Errors:** Events are discarded to prevent blocking
- **Exponential Backoff:** Built-in retry delays for transient failures

### Maintenance Window Support

- Graceful handling of maintenance periods
- Event skipping during maintenance windows

- Automatic resumption after maintenance completion

## **Monitoring and Observability**

- Comprehensive metrics and alerting
- Distributed tracing support
- Event processing telemetry

## **Integration Ecosystem**

The Case Event Handler integrates with numerous systems:

### **Datadog Core Services**

- **Timeline Service:** Case timeline visualization
- **Notification Platform:** Multi-channel notifications
- **Search Service:** Case discovery and filtering
- **Analytics Platform:** Metrics and reporting

### **External Systems**

- **Jira:** Issue tracking integration
- **ServiceNow:** ITSM integration
- **On-Call Services:** Incident response integration

### **Infrastructure Services**

- **Kafka:** Event streaming platform
- **Elasticsearch:** Search and analytics
- **PostgreSQL:** Persistent data storage

## **Deployment and Operations**

### **Service Deployment**

- Each flavor deployed as separate Kubernetes service
- Independent scaling based on processing requirements
- Flavor-specific configuration and resource allocation

### **Configuration Management**

- Environment-specific configuration files
- Feature flags for controlled rollouts
- Runtime configuration updates

## **Monitoring and Alerting**

- Handler-specific metrics and dashboards
- SLA monitoring and alerting
- Performance and error rate tracking

## **Conclusion**

The Case Event Handler represents a sophisticated, event-driven microservice architecture that successfully handles complex case management workflows at scale. Its multi-flavor design provides excellent separation of concerns while maintaining operational efficiency and system reliability.

The service's plugin-like architecture enables rapid development of new integrations and features while maintaining system stability and performance. Its robust error handling and monitoring capabilities ensure high availability and operational visibility.

This architecture serves as an excellent example of how complex business domains can be decomposed into manageable, specialized components while maintaining system coherence and reliability.

---

*Report Generated: December 2024 Architecture Version: Current State Analysis*