Event Schema Documentation

Event Schema Documentation

> **Project:** {{PROJECT_NAME}} **Version:** {{VERSION}} **Date:** {{DATE}}
> **Author:** {{AUTHOR}} **Status:** Draft | In Review | Approved **Reviewers:** {{REVIEWERS}}

Document History

Version	Date	Author	Changes
0.1	{{DATE}}	{{AUTHOR}}	Initial draft

---

1. Event-Driven Architecture Overview

```
graph LR
    subgraph "Publishers"
        UserService["user-service"]
        OrderService["order-service"]
        PaymentService["payment-service"]
    end

    subgraph "Message Broker"
        Broker["{{Kafka / RabbitMQ / AWS SQS + SNS}}"]
    end

    subgraph "Consumers"
        NotifService["notification-service"]
        AnalyticsService["analytics-service"]
        SearchService["search-service"]
```

```
        AuditService["audit-service"]
    end

    UserService -->|user.* events| Broker
    OrderService -->|order.* events| Broker
    PaymentService -->|payment.* events| Broker

    Broker -->|filtered| NotifService
    Broker -->|all events| AnalyticsService
    Broker -->|entity events| SearchService
    Broker -->|all events| AuditService
```

Event-driven use cases in this system:

- Decoupled notifications (user.created → send welcome email)
- Search index updates (entity.updated → reindex)
- Audit trail (all mutations → audit log)
- Cross-service data sync (order.created → update inventory)

---

2. Message Broker Configuration

Broker: `{{Apache Kafka | RabbitMQ | AWS SQS/SNS | NATS | Upstash Kafka}}` **Version:** `{{3.x}}`
Hosting: `{{Confluent Cloud / self-hosted / AWS MSK}}`

Topic / Queue Naming Convention

```
{{DOMAIN}}.{{ENTITY}}.{{ACTION}}

Examples:
  user.user.created
  order.order.status_changed
  payment.invoice.generated
  notification.email.sent
```

Pattern rules:

- All lowercase, dot-separated
- Domain prefix = service name (without `-service`)
- Entity = singular noun
- Action = past tense verb (created, updated, deleted, completed)

Topic Configuration

Topic	Partitions	Replication	Retention	Compaction
`user.user.*`	6	3	7 days	No
`order.order.*`	12	3	30 days	No
`payment.invoice.*`	6	3	90 days	No
`*.*.deleted`	6	3	30 days	Log compaction

---

3. Event Naming Conventions

Component	Rule	Examples
Full event type	`{domain}.{entity}.{action}`	`user.user.created`
Domain	Lowercase, matches service prefix	`user`, `order`, `payment`
Entity	Singular noun, lowercase with underscores	`user`, `order_item`, `invoice`
Action	Past-tense verb, lowercase with underscores	`created`, `updated`, `status_changed`, `payment_failed`

Do NOT use:

- Present tense (`user.user.create` — wrong)
- Generic names (`user.user.changed` — too vague)
- Abbreviations (`usr.usr.crtd` — unreadable)

---

4. Event Envelope Format (CloudEvents 1.0)

```
{
  "specversion": "1.0",
  "type": "{{DOMAIN}}.{{ENTITY}}.{{ACTION}}",
  "source": "{{SERVICE_NAME}}",
  "id": "evt_01HX7M2K5N3P4Q5R6S7T8V9W0",
  "time": "2024-01-15T10:30:00.000Z",
  "datacontenttype": "application/json",
```

```
  "subject": "{{optional: entity ID}}",
  "data": {
    "{{field}}": "{{value}}"
  }
}
```

Envelope fields:

Field	Type	Required	Description
`specversion`	string	Yes	Always `"1.0"`
`type`	string	Yes	Event type (see naming convention)
`source`	string	Yes	Emitting service name
`id`	string	Yes	Unique event ID (ULID format)
`time`	string	Yes	ISO 8601 timestamp (UTC)
`datacontenttype`	string	Yes	Always `"application/json"`
`subject`	string	No	Primary entity ID (for routing)
`data`	object	Yes	Domain-specific event payload

TypeScript interface:

```
interface CloudEvent<T = Record<string, unknown>> {
  specversion: '1.0';
  type: string;
  source: string;
  id: string;
  time: string;
  datacontenttype: 'application/json';
  subject?: string;
  data: T;
}
```

---

5. Per-Event Documentation

5.1 User Service Events

user.user.created

Published when a new user account is created.

Property	Value
Publisher	`user-service`
Consumers	`notification-service`, `analytics-service`, `audit-service`
Topic	`user.user.created`
Ordering guarantee	Per user ID (partitioned by subject)
Idempotency key	`id` (event ID) — consumers must deduplicate
Retry behavior	Consumer retries up to 5x before DLQ

Payload schema (JSON Schema):

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": ["userId", "email", "name", "role", "createdAt"],
  "properties": {
    "userId": { "type": "string", "format": "uuid" },
    "email": { "type": "string", "format": "email" },
    "name": { "type": "string", "minLength": 1 },
    "role": { "type": "string", "enum": ["admin", "user", "viewer"] },
    "createdAt": { "type": "string", "format": "date-time" }
  }
}
```

Example event:

```
{
  "specversion": "1.0",
  "type": "user.user.created",
  "source": "user-service",
  "id": "evt_01HX7M2K5N3P4Q5R6S7T8V9W0",
  "time": "2024-01-15T10:30:00.000Z",
  "datacontenttype": "application/json",
  "subject": "usr_01HX7...",
```

```
  "data": {
    "userId": "usr_01HX7...",
    "email": "newuser@example.com",
    "name": "Jane Doe",
    "role": "user",
    "createdAt": "2024-01-15T10:30:00.000Z"
  }
}
```

---

user.user.updated

Published when user profile data changes.

Property	Value
Publisher	`user-service`
Consumers	`search-service`, `notification-service`, `analytics-service`
Topic	`user.user.updated`
Ordering guarantee	Per user ID
Idempotency key	`id` (event ID)

Payload schema:

```
{
  "type": "object",
  "required": ["userId", "updatedFields", "updatedAt"],
  "properties": {
    "userId": { "type": "string" },
    "updatedFields": {
      "type": "array",
      "items": { "type": "string" },
      "description": "List of field names that changed"
    },
    "before": { "type": "object", "description": "Previous values (only changed fields)" },
    "after": { "type": "object", "description": "New values (only changed fields)" },
    "updatedAt": { "type": "string", "format": "date-time" }
  }
}
```

Example event:

```
{
  "specversion": "1.0",
  "type": "user.user.updated",
  "source": "user-service",
  "id": "evt_01HX8...",
  "time": "2024-01-16T08:00:00.000Z",
  "datacontenttype": "application/json",
  "subject": "usr_01HX7...",
  "data": {
    "userId": "usr_01HX7...",
    "updatedFields": ["name"],
    "before": { "name": "Jane Doe" },
    "after": { "name": "Jane Smith" },
    "updatedAt": "2024-01-16T08:00:00.000Z"
  }
}
```


---

user.user.deleted

Published when a user account is soft-deleted.

Property	Value
Publisher	`user-service`
Consumers	`order-service`, `notification-service`, `analytics-service`
Payload	`{ userId, deletedAt, reason }`
Ordering guarantee	Per user ID

Example event:


```
{
  "specversion": "1.0",
  "type": "user.user.deleted",
  "source": "user-service",
  "id": "evt_01HX9...",
  "time": "2024-01-17T12:00:00.000Z",
  "datacontenttype": "application/json",
  "subject": "usr_01HX7...",
  "data": {
    "userId": "usr_01HX7...",
```

```
    "deletedAt": "2024-01-17T12:00:00.000Z",
    "reason": "user_requested"
  }
}
```

5.2 Order Service Events

order.order.created

Property	Value
Publisher	`order-service`
Consumers	`payment-service`, `notification-service`, `inventory-service`
Topic	`order.order.created`
Ordering guarantee	Per order ID

Payload schema:

```
{
  "type": "object",
  "required": ["orderId", "userId", "items", "total", "currency", "createdAt"],
  "properties": {
    "orderId": { "type": "string" },
    "userId": { "type": "string" },
    "items": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "productId": { "type": "string" },
          "quantity": { "type": "integer" },
          "unitPrice": { "type": "number" }
        }
      }
    },
    "total": { "type": "number" },
    "currency": { "type": "string", "pattern": "^[A-Z]{3}$" },
    "createdAt": { "type": "string", "format": "date-time" }
  }
```

```
}
```

5.3 {{DOMAIN}} Service Events

{{domain}}.{{entity}}.{{action}}

Property	Value
Publisher	`{{service-name}}`
Consumers	`{{consumer-a, consumer-b}}`
Topic	`{{domain.entity.action}}`
Ordering guarantee	`{{Per entity ID
Idempotency key	`{{id}}`

Payload schema: `TODO: Define JSON Schema`

Example event: `TODO: Add example`

6. Dead Letter Queue Handling

DLQ naming: `{{topic}}.dlq` — e.g., `user.user.created.dlq`

DLQ workflow:

```
Event Published
    ↓
Consumer processes
    ↓ Fails
Retry (exp. backoff: 1s, 2s, 4s, 8s, 16s — max 5 retries)
    ↓ All retries exhausted
Move to DLQ
    ↓
Alert fires: PagerDuty P3
    ↓
On-call investigates
    ↓
Option A: Fix consumer bug → replay from DLQ
Option B: Skip message (data was invalid) → log + discard
```

DLQ message format:

```
{
  "originalEvent": { /* original CloudEvent */ },
  "failureReason": "Consumer threw: Cannot read property 'id' of undefined",
  "attemptCount": 5,
  "firstFailedAt": "2024-01-15T10:30:00Z",
  "lastFailedAt": "2024-01-15T10:32:00Z",
  "consumerGroup": "notification-service-consumer"
}
```

DLQ retention: 14 days. **DLQ alert threshold:** > 10 messages in DLQ within 5 minutes.

---

7. Event Versioning Strategy

Strategy: Backward-compatible field addition + major version in event type.

Rules:

1. Adding optional fields: allowed without version bump
2. Removing fields: NOT allowed (use deprecation first, remove after all consumers updated)
3. Changing field types: NOT allowed (breaking change)
4. Adding required fields: requires version bump
5. Major breaking change: new event type `user.user.created.v2`

Deprecation process:

```
1. Mark field as deprecated in schema docs
2. Notify all consumer teams
3. Wait 2 sprint cycles (4 weeks minimum)
4. Remove field from schema
5. Update documentation
```

Schema registry: `{{Confluent Schema Registry | AWS Glue Schema Registry | Manual docs}}`
Validation: Consumer validates incoming events against pinned schema version.

---

8. Event Replay Capability

Replay supported: `{{Yes — Kafka log retention | No — events are ephemeral}}`

Replay scenarios:

- Bug in consumer → fix bug → replay affected time window
- New consumer onboarded → replay historical events to build initial state
- Data migration → replay events to new storage

Replay procedure:

1. Identify topic and time range to replay
2. Coordinate with all consumer teams (replay may cause duplicate side effects)
3. Ensure consumers are idempotent before replay
4. Set consumer offset to target timestamp: `kafka-consumer-groups --reset-offsets --to-datetime`
5. Restart consumer with temporary consumer group to avoid affecting production offset
6. Verify replayed state is correct
7. Switch production consumer to new state

Retention periods by topic: See topic configuration table in Section 2.

---

9. Monitoring & Observability

Metric	Alert Threshold	Severity	Channel
Consumer lag (per topic)	> 10,000 messages	P2	Slack `#alerts`
DLQ depth	> 10 messages / 5min	P3	Slack `#alerts`
Producer error rate	> 1% / 5min	P1	PagerDuty
Consumer error rate	> 5% / 5min	P2	PagerDuty
Event processing latency P99	> 5 seconds	P3	Slack `#alerts`

Dashboard: `{{https://monitoring.domain.com/dashboards/events}}` **Distributed tracing:** All events carry `traceparent` header (OpenTelemetry W3C Trace Context).

---

10. Testing Event-Driven Flows

Unit Tests

```
// Test producer: verify event shape
it('should publish user.created event with correct schema', async () => {
  await userService.create(createUserDto);

  expect(eventBus.publish).toHaveBeenCalledWith(
    expect.objectContaining({
      type: 'user.user.created',
      source: 'user-service',
      data: expect.objectContaining({
        userId: expect.any(String),
        email: createUserDto.email,
      }),
    })
  );
});

// Test consumer: verify handler idempotency
it('should not send welcome email twice for duplicate event', async () => {
  const event = buildUserCreatedEvent();
  await handler.handle(event);
  await handler.handle(event); // duplicate
  expect(emailService.send).toHaveBeenCalledTimes(1);
});
```

Integration Tests

```
// Use real broker in integration tests (testcontainers)
const kafka = await new KafkaContainer('confluentinc/cp-kafka:7.5.0').start();
```

E2E Tests

Test full event chain: API action → event published → consumer processes → side effect visible.

```
POST /users → poll for welcome email (SendGrid sandbox) → assert received within 5s
```

Approval

Role	Name	Date	Signature
Author			
Backend Lead			
Platform / Infrastructure Lead			
Architect			

Revision #9
Created 2026-02-23 12:05:31 UTC by John
Updated 2026-05-25 07:33:39 UTC by John