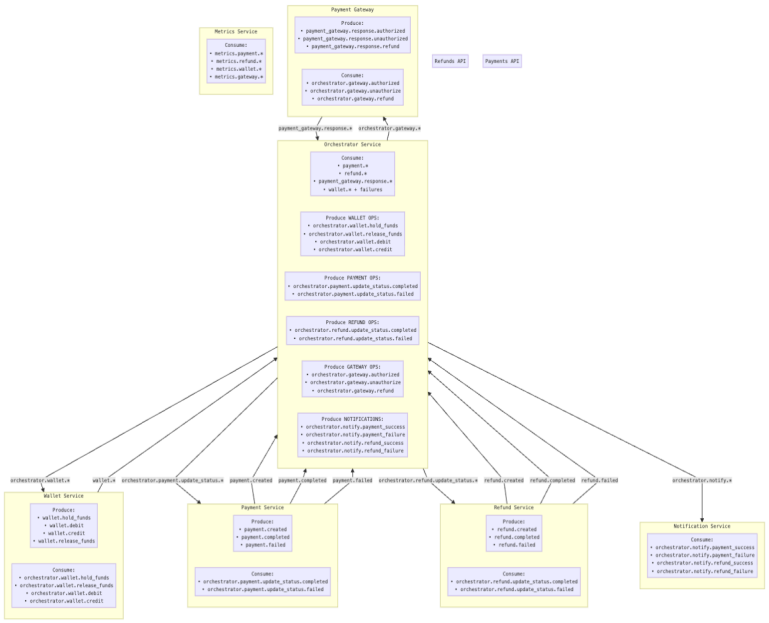
Especificación de Diseño de Eventos

Catálogo de eventos



Estructura de colas y convenciones de nombres

- **Convención general:** {domain}.{action}
- **Convención SAGA Pattern:** orchestrator.{domain}.{action}
- **Convención Métrica:** metric.{domain}.{action}

Tipos de topic/queue:

- **Command queues (point-to-point):** para comandos que deben procesarse por un único consumidor (ej. orchestrator.notify.payment_success dirigido al Wallet Service para evitar competencias).

Esquemas de eventos JSON

PAYMENT SERVICE – Eventos

payment.created

```
{
  "event_id": "uuid-v4",
  "event_type": "payment.created",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
```

```
  "payload": {
    "payment_id": "pay_123",
    "amount": 1000,
    "currency": "ARS",
    "customer_id": "cus_777",
    "status": "PENDING"
  },
  "metadata": {
    "source": "payment-service",
    "idempotency_key": "key-123",
    "trace_id": "trace-123",
    "message_group_id": "payment_pay_123"
  }
}
```

payment.completed

```
{
  "event_id": "uuid-v4",
  "event_type": "payment.completed",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "payment_id": "pay_123",
    "status": "COMPLETED"
  },
  "metadata": {
    "source": "payment-service",
    "trace_id": "trace-123",
    "message_group_id": "payment_pay_123"
  }
}
```

payment.failed

```
{
  "event_id": "uuid-v4",
  "event_type": "payment.failed",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "payment_id": "pay_123",
    "status": "FAILED",
    "error_code": "gateway_declined",
    "error_message": "Payment was declined by gateway"
```

```
  },
  "metadata": {
    "source": "payment-service",
    "trace_id": "trace-123",
    "message_group_id": "payment_pay_123"
  }
}
```

REFUND SERVICE – Eventos

refund.created

```
{
  "event_id": "uuid-v4",
  "event_type": "refund.created",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "refund_id": "ref_123",
    "payment_id": "pay_123",
    "amount": 500,
    "currency": "ARS",
    "status": "PENDING"
  },
  "metadata": {
    "source": "refund-service",
    "trace_id": "trace-456",
    "message_group_id": "refund_ref_123"
  }
}
```

refund.completed

```
{
  "event_id": "uuid-v4",
  "event_type": "refund.completed",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "refund_id": "ref_123",
    "status": "COMPLETED"
  },
  "metadata": {
    "source": "refund-service",
    "trace_id": "trace-456",
```

```
    "message_group_id": "refund_ref_123"
  }
}
```


refund.failed


```
{
  "event_id": "uuid-v4",
  "event_type": "refund.failed",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "refund_id": "ref_123",
    "status": "FAILED",
    "error_code": "gateway_error",
    "error_message": "Refund failed on provider"
  },
  "metadata": {
    "source": "refund-service",
    "trace_id": "trace-456",
    "message_group_id": "refund_ref_123"
  }
}
```


WALLET SERVICE – Eventos

wallet.hold_funds


```
{
  "event_id": "uuid-v4",
  "event_type": "wallet.hold_funds",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "wallet_id": "wal_123",
    "payment_id": "pay_123",
    "amount": 1000,
    "status": "HELD"
  },
  "metadata": {
    "source": "wallet-service",
    "trace_id": "trace-789",
    "message_group_id": "wallet_wal_123"
  }
}
```

wallet.debit


```
{
  "event_id": "uuid-v4",
  "event_type": "wallet.debit",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "wallet_id": "wal_123",
    "payment_id": "pay_123",
    "amount": 1000,
    "status": "DEBITED"
  },
  "metadata": {
    "source": "wallet-service",
    "trace_id": "trace-789",
    "message_group_id": "wallet_wal_123"
  }
}
```


wallet.credit


```
{
  "event_id": "uuid-v4",
  "event_type": "wallet.credit",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "wallet_id": "wal_123",
    "refund_id": "ref_123",
    "amount": 500,
    "status": "CREDITED"
  },
  "metadata": {
    "source": "wallet-service",
    "trace_id": "trace-789",
    "message_group_id": "wallet_wal_123"
  }
}
```


wallet.release_funds


```
{
```

```
  "event_id": "uuid-v4",
  "event_type": "wallet.release_funds",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "wallet_id": "wal_123",
    "payment_id": "pay_123",
    "amount": 1000,
    "status": "RELEASED"
  },
  "metadata": {
    "source": "wallet-service",
    "trace_id": "trace-789",
    "message_group_id": "wallet_wal_123"
  }
}
```

PAYMENT GATEWAY – Eventos

payment_gateway.response


```
{
  "event_id": "uuid-v4",
  "event_type": "payment_gateway.response",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "payment_id": "pay_123",
    "gateway_status": "AUTHORIZED",
    "raw_response": {}
  },
  "metadata": {
    "source": "payment-gateway",
    "trace_id": "trace-gw",
    "message_group_id": "payment_pay_123"
  }
}
```

ORCHESTRATOR – Comandos

orchestrator.wallet.hold_funds

```
{
  "event_id": "uuid-v4",
  "event_type": "orchestrator.wallet.hold_funds",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "payment_id": "pay_123",
    "amount": 1000
  },
  "metadata": {
    "source": "orchestrator",
    "trace_id": "trace-orch",
    "message_group_id": "payment_pay_123"
  }
}
```

(Los otros comandos del orchestrator siguen el mismo formato cambiando event_type y payload.)

NOTIFICATION SERVICE – Eventos

orchestrator.notify.payment_success

```
{
  "event_id": "uuid-v4",
  "event_type": "orchestrator.notify.payment_success",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "payment_id": "pay_123",
    "customer_id": "cus_777"
  },
  "metadata": {
    "source": "orchestrator",
    "trace_id": "trace-notif",
    "message_group_id": "notify_pay_123"
  }
}
```

orchestrator.notify.payment_failure

```
{
  "event_id": "uuid-v4",
```

```
  "event_type": "orchestrator.notify.payment_failure",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "payment_id": "pay_123",
    "customer_id": "cus_777",
    "reason": "gateway_declined"
  },
  "metadata": {
    "source": "orchestrator",
    "trace_id": "trace-notif",
    "message_group_id": "notify_pay_123"
  }
}
```


orchestrator.notify.refund_success


```
{
  "event_id": "uuid-v4",
  "event_type": "orchestrator.notify.refund_success",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "refund_id": "ref_123",
    "customer_id": "cus_777"
  },
  "metadata": {
    "source": "orchestrator",
    "trace_id": "trace-notif",
    "message_group_id": "notify_ref_123"
  }
}
```


orchestrator.notify.refund_failure


```
{
  "event_id": "uuid-v4",
  "event_type": "orchestrator.notify.refund_failure",
  "event_version": "1.0",
  "timestamp": "ISO-8601",
  "payload": {
    "refund_id": "ref_123",
    "customer_id": "cus_777",
    "reason": "provider_error"
  },
```

```
  "metadata": {
    "source": "orchestrator",
    "trace_id": "trace-notif",
    "message_group_id": "notify_ref_123"
  }
}
```

Consideraciones operacionales y de diseño

- **Idempotencia:** Todas las APIs públicas aceptan idempotent_key. Servicios propietarios deben guardar el mapping idempotent_key => entity_id.

- **Retries y backoff:**
    - DLQs 3 reintentos con backoff exponencial para comandos a servicios externos).
    - Errores permanentes deben generar compensaciones.

- **Errores y observabilidad:**
    - Propagar trace_id en todos los eventos para trazabilidad distribuida.
    - Metrics Service consume eventos metrics.*.

- **Seguridad:** todas las operaciones con proveedores externos, el gateway solo mapea y soporte la tokenización.

- **Schema evolution:** versionar eventos con campo version y mantener compatibilidad hacia atrás.
- **SAGA Pattern:** Orchestrator actúa como coordinador central que envía comandos y escucha resultados para avanzar/compensar.

Ejemplo: pago con wallet + gateway

1. payment.created (Payment Service) → Orchestrator escucha.

2. Orchestrator orchestrator.wallet.hold_funds → Wallet Service procesa y emite wallet.hold_funds.completed o .failed.

3. Si wallet produce hold_funds, Orchestrator lo consume y envía orchestrator.gateway.authorize → Payment Gateway procesa y finalmente emite payment_gateway.response

4. Si authorized, Orchestrator envía orchestrator.wallet.debit → Wallet emite wallet.debit_funds.

5. Al completarse, Orchestrator emite orchestrator.payment.update_status.completed → Payment Service marca COMPLETED y emite payment.completed.

6. Si en cualquier paso ocurre fallo, Orchestrator realiza compensaciones: por ejemplo, si gateway devuelve unauthorized después de hold_funds.completed, Orchestrator envía orchestrator.wallet.release_funds.

Orden de eventos y garantías de entrega

Implementará colas **SQS FIFO** debido a que garantizan:

- Los mensajes se procesan exactamente en el orden en que fueron enviados **dentro del mismo** MessageGroupId. = payment_id o refund_id **generando sync**
- **Exactly-once delivery:** No duplica mensajes (garantía lógica). setear MessageDeduplicationId = ${event_type}.${entity_id}
- VisibilityTimeout > tiempo máximo de procesamiento para evitar mensajes reprocesados, mensajes invocados por múltiples consumers
- **Retry automático incorporado:** SQS reintentará indefinidamente mientras el mensaje no se elimine.
    - Eliminar manualmente el mensaje cuando el procesamiento esté completo.
- Configurar una cola DLQ asociada
    - La clave de la garantía: maxReceiveCount = 3–5
    - Si tu consumidor falla N veces → el mensaje va a la DLQ, **confirmando que no se perdió**.
- **Monitoreo y alarmas** en base al a métricas de CloudWatch
    - ApproximateNumberOfMessagesVisible > X
    - ApproximateNumberOfMessagesDelayed > X
    - ApproximateNumberOfMessagesNotVisible > X
    - DLQ > X umbral

Stack tecnológico

Simplificarlo el stack con las siguientes reglas:

- Utilizar librerías de memory cache para garantizar idempotencia en las puertas de entrada.
- **Wallet** utilizará un RDS
    - Operaciones que cumplen con ACID para manejo de saldos
    - Ledger Layer
- Los otros servicios podrían DynamoDB ya que no requieren fuerte consistencia ni operaciones atómicas.
- **Payments Gateway** debe implementar librerías de circuit breaker para:
    - Soportar múltiples proveedores de pago
    - Definir reintentos por proveedor o latencia para poder switchear a otro que mejor performance y garantizar operatividad.
- Colas y mensajería SNS/SQS.