ChatGPT

Rate Platform: Redis Streams Integration – Final Plan

1. Introduction

Tài liệu này tổng hợp mục tiêu, công nghệ, thiết kế và kế hoạch triển khai Redis Streams Integration cho Rate Platform. Nó bao gồm các điểm sửa đổi "Critical Fixes" từ phản biện cuối cùng và lời khuyến nghị để solo dev có thể thực thi step-by-step.

2. Mục tiêu

- **Zero-downtime** khi deploy Strapi và Validator worker.
- **Reliable message bus** cho real-time (Extension) và bulk (Importer) workloads.
- **Production-grade operations**: security, observability, error-tracking, data retention.
- **Developer ergonomics**: nhanh chóng cài đặt, build, deploy với pnpm và rolling-reload.

3. Công nghệ chính

- **Backend**: Strapi v5 (API Gateway / custom endpoints)
- **Message Bus**: Redis Streams (consumer groups, DLQ)
- **Persistence**: PostgreSQL (JSONB, partitioned tables)
- **Identifiers**: UUIDv7 (`pg_uuidv7` extension)
- **Package manager**: pnpm
- **Process manager**: PM2 (cluster + rolling-restart)
- **Monitoring**: Prometheus + Grafana, pgaudit
- **CI/CD**: GitHub Actions / GitLab CI + `pm2 reload`

4. Kiến trúc hệ thống

```
graph LR
  subgraph Clients
    A[Rate-Extension]
    B[Rate-Importer]
  end

  subgraph Strapi API
    C[Strapi (cluster PM2)]
    C1[/api/validation/.../]
    C2[/api/importer/.../]
  end

  subgraph Redis
    D[validation_requests Stream]
    E[extension_responses Stream]
    F[importer_responses Stream]
    G[validation_dlq Stream]
  end

  subgraph Validator
    H[Validator Worker (cluster PM2)]
  end

  A -->|HTTPS JWT/API-Token| C1
  B -->|HTTPS API-Token| C2
  C1 & C2 -->|XADD→D| D
  H -->|XREADGROUP→D| D
  H -->|XACK / XTRIM→D| D
  H -->|XADD→E| E
  H -->|XADD→F| F
  C1 <--|XREADGROUP←E| E
  C2 <--|XREADGROUP←F| F
  H -->|XADD→G| G
  C & H -->|PostgreSQL| I[(validation_requests, validation_responses,
raw_item_errors)]
```

5. Thiết kế chi tiết

5.1 Security & Access Control

- **Redis ACL**: `bash`

```
  # redis-acl.conf
auth default "" on nopass ~* +@all
user strapi on >strapi_password
  ~validation_requests:* ~status:* +@stream +@read +xadd
user validator on >validator_password
  ~validation_requests:* +@stream +@read +@write
user prometheus on >prometheus_password
  ~* +@read +info +ping +xinfo -@dangerous
```

- **JWT & API-Token scopes**:

```
const validateServiceToken = (requiredScopes: string[]) => async (req,
res, next) => {
  const tokenData = await validateApiToken(req.headers['x-api-token']);
  if (!requiredScopes.every(s=>tokenData.scopes.includes(s)))
    return res.status(403).json({ error: 'Insufficient scope' });
  next();
};
```

```
// Usage:
app.post('/api/validation/validate',
validateServiceToken(['create:validation_requests']), handleValidation);
```

5.2 Reliability & Exactly-Once Processing

- **Consumer group init**:

```
async function initGroup() {
  try {
    await
redis.xgroup('CREATE','validation_requests','validators','$','MKSTREAM');
  } catch(e){ if(!e.message.includes('BUSYGROUP')) throw e; }
}
```

- **Worker loop với DLQ & retry**:

```
while(true){
  const msgs = await redis.xreadgroup('GROUP','validators',`worker-${pid}
`,'COUNT',10,'BLOCK',5000,'STREAMS','validation_requests','>');
  for(const [stream, entries] of msgs) for(const [id, fields] of entries)
try{
    await processValidation(fields);
    await redis.xack('validation_requests','validators',id);
  } catch(err) { await handleFailure(id,fields,err); }
}
```

- **Dead-letter + backoff**

5.3 Data Persistence Strategy

- **Hybrid tables**:

```
CREATE TABLE validation_requests(
  id UUID PRIMARY KEY DEFAULT uuid_generate_v7(),
  request_id VARCHAR(64) UNIQUE,
  source VARCHAR(50), data JSONB, status VARCHAR(20) DEFAULT 'pending',
created_at TIMESTAMPTZ DEFAULT NOW()
);
CREATE TABLE validation_responses(
  id UUID PRIMARY KEY DEFAULT uuid_generate_v7(),
  request_id VARCHAR(64) REFERENCES validation_requests(request_id),
result JSONB, processing_time_ms INT, created_at TIMESTAMPTZ DEFAULT NOW()
);
```

- **Error details**:

```
CREATE TABLE raw_item_errors PARTITION BY RANGE (occurred_at);
-- partition func with advisory locks
```

- **Stream trimming**:

```
setInterval(async()=>{
  await redis.xtrim('validation_requests','MAXLEN','~',10000);
  const oldIds = await getProcessedOlderThan(Date.now()-3600000);
  if(oldIds.length) await redis.xdel('validation_requests',...oldIds);
},300000);
```

5.4 Scalability & Bulk Import

- **Back-pressure**: check `XLEN` >50k then `await sleep(1000);`
- **Pipeline** for batch `XADD`
- **Redis Cluster** config:

```
cluster-enabled yes
cluster-config-file nodes.conf
maxmemory 4gb
maxmemory-policy allkeys-lru
```

5.5 Monitoring & Observability

- **Prometheus metrics** (stream length, lag, histograms):

```
const streamLength = new Gauge({...});
setInterval(async()=>{/* XINFO STREAM, XINFO GROUPS */},5000);
app.get('/metrics',(_,res)=>res.set('Content-
Type',register.contentType).end(register.metrics()));
```

- **Health check**:

```
@app.get('/healthz')
async def health_check():
  db_ok = await conn.fetchval('SELECT 1')==1
  redis_ok = await redis.ping()
  disk_pct = os.statvfs('/').f_bavail/os.statvfs('/').f_blocks*100
  status=200 if all([db_ok,redis_ok,disk_pct>15]) else 503
  return JSONResponse({...},status)
```

- **pgaudit**:

```
ALTER SYSTEM SET pgaudit.log='DDL';
ALTER TABLE users SET (pgaudit.log='ALL');
```

5.6 UI Integration & Hybrid Approach

- **Strapi Content-Type** `validation-request` schema for Admin UI
- **Real-time dashboard** via WebSocket in custom admin component

5.7 Zero-Downtime Deployment

- **PM2 ecosystem.config.js**:

```
module.exports={apps:[{name:'strapi',script:'npm',args:'start',instances:
2,exec_mode:'cluster',wait_ready:true,listen_timeout:3000,kill_timeout:
5000,health_check:{interval:30,path:'/api/health',timeout:
5000},env_production:{NODE_ENV:'production',PM2_GRACEFUL_LISTEN_TIMEOUT:
1000,PM2_GRACEFUL_TIMEOUT:5000}}]};
process.send('ready');
```

- **Validator**: graceful SIGINT handling

5.8 Backup & Disaster Recovery

- **Postgres** continuous archiving + WAL to S3
- **Redis** RDB snapshots → S3
- **Config** backup script for `/etc/strapi`, `/etc/redis`, `ecosystem.config.js`

5.9 Testing Strategy

- **E2E tests** for pipeline:

```
describe('Validation Pipeline',()=>{
  beforeAll(initTestStream);
  test('1k concurrent',async()=>{
    await publishBulkValidation(requests);
    await waitForStreamEmpty('test_validation_requests',60000);
    expect(Date.now()-start).toBeLessThan(30000);
  });
});
```

5.10 Cost Estimation & Complexity

Component	Monthly Cost (USD)	Notes
VPS (4CPU, 8GB)	40–80	DigitalOcean/Vultr
PostgreSQL managed	15–50	or self-host
Redis managed	20–40	or self-host
S3 backup	5–10	100GB
Monitoring	0–20	Grafana Cloud free tier
Total	**80–200**	

Complexity warning: Week 2 tasks có thể quá nặng; tách thành 2 phase: 2a) Redis Streams + Health; 2b) Monitoring + pgaudit.

6. Kế hoạch triển khai & Roadmap

Tuần	Mục tiêu	Công việc chính
W1	Foundation	pnpm migration + lockfile;<br>PM2 cluster setup;<br>Error tracking tables + uuidv7;<br>Backup script templates.
W2a	Redis Streams & Health	Implement Streams + consumer group + DLQ;<br>Stream trimming;<br>Lightweight `/healthz` endpoint.
W2b	Monitoring & Auditing	Prometheus rules + `/metrics`;<br>pgaudit config;<br>Prom client instrumentation.
W3	Bulk & Partitioning	Generate realistic test data;<br>Bulk publisher + back-pressure;<br>Partition-by-month trigger + advisory lock;<br>Debezium alternative (LISTEN/NOTIFY).
W4	Staging → Production rollout	CI/CD pipeline: `pnpm install`→build→`pm2 reload`;<br>Smoke tests & rollback docs;<br>WebSocket/SSE for dashboard.
Phase 2	Advanced features	Horizontal scaling (Redis Cluster, more workers);<br>Archive old data to S3;<br>Advanced queue management;
Analytics dashboard.		

7. Risk Matrix

Risk	Probability	Impact	Mitigation
Redis OOM	Medium	High	`maxmemory-policy`; monitor usage; alert on XLEN lag.
PM2 reload failure	Low	High	Test in staging; have rollback script.
Stream backlog	Medium	Medium	Back-pressure; auto-scale workers.
Data loss	Low	Critical	Backup strategy; periodic restore tests.

Tóm lại, với plan này, Rate Platform sẽ đạt được performance của Custom API, reliability của proper queue system, convenience của Strapi UI và production readiness từ ngày đầu.

Bạn có thể copy nguyên file này vào Cursor để bắt đầu triển khai!