UAS Sistem Terdistribusi – Pub-Sub Log Aggregator

Python 3.11

FastAPI 0.110

Docker Compose

Tests 20 passing

Sistem Pub-Sub Log Aggregator terdistribusi dengan **idempotent consumer**, **deduplication kuat**, dan **kontrol transaksi/konkurensi** berbasis Docker Compose.

Arsitektur

```
┌─────────────────────────────────────────────────────────────┐
│                 Docker Compose Network (internal)            │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  ┌────────────┐     ┌────────────┐     ┌────────────────┐   │
│  │ Publisher  │────▶│   Broker   │◀────│   Aggregator   │   │
│  │ (Python)   │     │  (Redis)   │     │  (FastAPI+     │   │
│  └────────────┘     └────────────┘     │   Workers)     │   │
│                           │            └────────────────┘   │
│                           │                    │            │
│                           ▼                    ▼            │
│                    ┌────────────┐      ┌────────────┐       │
│                    │broker_data │      │  Storage   │       │
│                    │ (volume)   │      │(PostgreSQL)│       │
│                    └────────────┘      └────────────┘       │
│                                               │             │
│                                               ▼             │
│                                        ┌────────────┐       │
│                                        │  pg_data   │       │
│                                        │ (volume)   │       │
│                                        └────────────┘       │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Komponen

- **aggregator**: FastAPI + background workers untuk processing events
- **publisher**: Generator event dengan duplikasi terkontrol (30%+)
- **broker**: Redis 7 sebagai message queue internal
- **storage**: PostgreSQL 16 untuk persistent dedup store

Quick Start

Menjalankan Sistem

```
# Build dan jalankan semua services
docker compose up --build
```

```
# Jalankan dengan publisher demo
docker compose --profile demo up --build

# Atau jalankan di background
docker compose up --build -d
```

Akses Endpoints

Endpoint	Method	Deskripsi
http://localhost:8080/health	GET	Health check
http://localhost:8080/publish	POST	Publish events
http://localhost:8080/events?topic=X	GET	List events by topic
http://localhost:8080/stats	GET	Aggregator statistics
http://localhost:8080/queue/stats	GET	Queue status

Contoh Request

```
# Health check
curl http://localhost:8080/health

# Publish single event
curl -X POST http://localhost:8080/publish \
  -H "Content-Type: application/json" \
  -d '{
    "events": [{
      "topic": "user-events",
      "event_id": "evt-12345",
      "timestamp": "2024-01-01T12:00:00Z",
      "source": "demo",
      "payload": {"action": "login"}
    }]
  }'

# Get statistics
curl http://localhost:8080/stats

# List events by topic
curl "http://localhost:8080/events?topic=user-events"
```

Testing

Unit/Integration Tests (20 tests)

```
# Install dependencies
pip install -r requirements-dev.txt

# Run all tests
pytest tests/ -v

# Run with coverage
pytest tests/ -v --cov=aggregator
```

Test Coverage

- Deduplication tests (3 tests)
- Persistence tests (2 tests)
- Transaction/Concurrency tests (4 tests)
- Schema validation tests (4 tests)
- API endpoint tests (4 tests)
- Batch processing tests (2 tests)
- Stress/Performance test (1 test)

Load Testing (K6)

```
# Install K6: https://k6.io/docs/getting-started/installation/
# macOS: brew install k6

# Run load test
k6 run loadtest/k6-script.js

# Custom options
k6 run --vus 20 --duration 60s loadtest/k6-script.js
```

Features

Idempotency & Deduplication

- Unique constraint `(topic, event_id)` di PostgreSQL
- `INSERT ON CONFLICT DO NOTHING` untuk atomic dedup
- Metrics: `unique_processed`, `duplicate_dropped`

Transaksi & Konkurensi

- ACID compliance dengan PostgreSQL
- Isolation level: READ COMMITTED
- Atomic counter updates: `SET count = count + 1`
- Thread-safe workers dengan independent sessions

Reliability

- At-least-once delivery dengan retry eksponensial
- Crash tolerance via persistent storage
- Container restart policy: `unless-stopped`
- Health checks untuk dependency ordering

Persistensi

- Named volumes: `pg_data`, `broker_data`, `aggregator_data`
- Data aman meski container dihapus
- Redis AOF untuk queue durability

Observability

- `/health` endpoint untuk liveness/readiness
- `/stats` dengan dedup rate percentage
- Structured logging
- `/queue/stats` untuk monitoring

Struktur Proyek

```
UAS/
├── aggregator/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── app/
│       ├── main.py          # FastAPI endpoints
│       ├── consumer.py      # Background workers
│       ├── models.py        # SQLAlchemy models
│       ├── schemas.py       # Pydantic schemas
│       ├── db.py            # Database connection
│       ├── queue.py         # Redis/InMemory queue
│       └── config.py        # Configuration
├── publisher/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── app/
│       └── main.py          # Event generator
├── tests/
│   └── test_api.py          # 20 unit/integration tests
├── loadtest/
│   └── k6-script.js         # K6 load test script
├── scripts/
│   ├── demo.sh
│   ├── quick-test.sh
│   └── test-docker.sh
├── docker-compose.yml       # Main compose file
├── docker-compose.test.yml  # Test compose file
├── requirements-dev.txt     # Development dependencies
├── REPORT.md               # Full report (T1-T10)
└── README.md               # This file
```

Configuration

Environment Variables

Variable	Default	Description
DATABASE_URL	postgresql+psycopg2://user:pass@storage:5432/uasdb	Database connection
REDIS_URL	redis://broker:6379/0	Redis connection
WORKER_COUNT	3	Number of background workers
QUEUE_KEY	event_queue	Redis queue key
DISABLE_WORKERS	0	Disable workers (for testing)
USE_INMEMORY_QUEUE	0	Use in-memory queue (for testing)

Publisher Configuration

Variable	Default	Description
TARGET_URL	http://aggregator:8080/publish	Aggregator URL
TOPIC	demo-topic	Default topic
RATE_PER_SEC	100	Events per second
DUP_RATE	0.35	Duplicate rate (35%)

API Schema

Event JSON

```
{
  "topic": "string (required, 1-255 chars)",
  "event_id": "string (required, unique per topic)",
  "timestamp": "ISO8601 datetime",
  "source": "string (required)",
  "payload": { "any": "json object" }
}
```

Stats Response

```
{
  "received": 1000,
  "unique_processed": 700,
  "duplicate_dropped": 300,
  "dedup_rate_percent": 30.0,
  "topics": ["topic-a", "topic-b"],
  "uptime_seconds": 120.5
}
```

Video Demo

Link: YouTube - UAS Sistem Terdistribusi Demo

References

Tanenbaum, A. S., & Van Steen, M. (2017). *Distributed Systems: Principles and Paradigms* (3rd ed.). Pearson Education.

License