📘 RSSAgg Architecture & Design Document (v1.0)

MarketSwarm Intelligence Pipeline — RSS Aggregator

---

1. Purpose

RSSAgg is responsible for:

Stage 1 — Metadata Ingestion

- Fetching categorized RSS feeds
- Extracting titles, links, images, abstracts
- Storing normalized metadata in Redis
- Publishing lightweight events into the *intel bus* for downstream consumption

Stage 2 — Full Article Extraction

- Listening to UIDs from the intel stream
- Downloading full article content
- Extracting images, metadata, and semantic features
- Storing enriched articles for use by Vexy AI and other MarketSwarm components

---

2. System Context & Responsibilities

```
   External RSS Feeds ───────▶ Stage 1: Metadata Ingestor ───────────▶
redis:intel:rss:content (STREAM)
                                                                   │
                                                                   ▼
                                                              Stage 2: Full
```

```
Article Worker
                                                          |
                                                          ▼

redis:intel:rss:article:{uid}
                                                          |
                                                          ▼
                                             Vexy AI (semantic
processing)
```
RSSAgg also produces **public RSS XML files** for publication on your website or external services (e.g., dlvr.it → X.com).

---

3. Core Components

3.1 Shell Scripts

Purpose: launch, test, debug, and manage service components.

Scripts include:

- ms-rssagg.sh — full service (main + heartbeat + orchestrator)
- ms-rssagg-ingest-test.sh — run metadata ingestion only
- ms-rssagg-worker-test.sh — run article extraction only
- ms-rssagg-publisher-test.sh — generate XML feeds only

These allow isolated testing of each subsystem.

3.2 Entry Point (main.py + setup.py)

Responsibilities

1. Load truth.json from **system-redis**

2. Validate component configuration
3. Prepare workspaces:
    - Redis domain keys
    - TTLs
    - directories for XML feeds
4. Start:
    - Heartbeat publisher
    - Orchestrator (Stage 1 loop + Feed Publisher loop)

Key outputs

- Redis structures guaranteed to exist before orchestrator runs
- Atomic publish of heartbeat
- Logs for supervision and recovery (mesh + healer services)

3.3 Orchestrator (orchestrator.py)

Responsibilities

- Coordinate concurrent tasks:
    1. Metadata ingestion loop (async)
    2. Feed generation loop (async)
- Use schedule derived from truth.json
- Log ingestion and publishing status
- Ensure Redis connectivity and handle retries

Two Tasks Running in Parallel

```
start_workflow()          # Stage 1 metadata ingest loop
schedule_feed_generation()# Public RSS feed XML generator
```

Inputs

- feeds.json (categories + sources)
- truth.json (publish_dir, access points)
- environment (paths, redis hosts)

3.4 Stage 1: Ingestor (ingestor.py)

Responsibilities

- Fetch XML from each RSS feed
- Normalize entries (title, url, image, abstract)
- Deduplicate based on UID hash
- Write metadata into Redis:
    - rss:item:{uid}
    - rss:index (ZSET sorted by published date)
    - rss:seen (SET for dedupe)
    - intel:rss:content (STREAM)

Key Guarantees

- Lightweight data (fast)
- Low latency
- No full article fetching here
- Exactly-once delivery of UIDs to Stage 2 worker

Output RECORD into intel bus

```
{
  "uid": "...",
  "category": "...",
  "title": "...",
  "abstract": "...",
  "url": "original source article"
}
```

3.5 Stage 2: Full Article Worker

(Will be built later — designed now)

Responsibilities

Consumer of intel:rss:content stream:

1. Download full HTML of article
2. Extract:
   - readable text
   - metadata (author, tags, canonical URL)
   - images
   - sentiment
   - keywords
   - topics
3. Store enriched article in:

```
rss:article:{uid}       # HASH
rss:raw_html:{uid}      # STRING
rss:images:{uid}        # SET
```

4. Optionally republish a **processed event** to downstream:
   - intel:rss:enriched

Design notes

- This is independent and horizontally scalable
- Can be triggered manually or can run continuously
- Future versions may handle:
  - paywall detection

    - summarization
    - embedding generation

3.6 Publisher (publisher.py)

Responsibilities

- Generate RSS XML feeds for **each category**
- Write them into a filesystem directory for:
    - your website
    - external syndication (dlvr.it for X.com)

Sources of truth

- All items come from rss:item:{uid}
- Sorting from rss:index
- Category filtering via hash field category

Output

- {publish_dir}/{category}.xml
- Written atomically for safety

---

4. Redis Architecture (Intel Bus)

Keys used by Stage 1 (metadata ingest)

```
rss:item:{uid}   (HASH)
rss:index        (ZSET)
```

```
rss:queue         (STREAM – downstream for now)
rss:seen          (SET)
intel:rss:content (STREAM – NEW)
```

Keys used by Stage 2 (full articles)

```
rss:article:{uid}      (HASH)
rss:raw_html:{uid}     (STRING)
rss:images:{uid}       (SET)
intel:rss:enriched     (STREAM – optional)
```

TTL Policies

- Lightweight metadata: 24h-48h
- Full articles: 7–30 days (configurable)
- Seen-Uids: 7 days
- Queues: maxlen trim policy

---

5. Filesystem Architecture

Under the service root:

```
/feeds/                         # public RSS XML files
/workspace/
     /articles/                 # optional snapshots of text
     /html/                     # raw HTML
     /images/                   # downloaded images
```

The service only *requires* /feeds/.

Workspace is optional for debugging or future offline tasks.

---

6. Truth.json Integration

From truth.json, RSSAgg consumes:

Buses

```
Intel: redis://127.0.0.1:6381
System: redis://127.0.0.1:6379   (heartbeat)
```

Access Points

```
publish_to:
   intel-redis: intel:rss:content (STREAM)
   system-redis: rss_agg:heartbeat
```

Workflow

```
interval_sec
publish_dir
```

RSSAgg trusts truth.json as the canonical configuration. No hardcoded redis hosts.

---

7. Error Handling & Observability

Ingestor

- Logs fetch failures
- Ignores malformed entries
- Tracks successes per category

Publisher

- Skips empty categories
- Logs exception per feed

Orchestrator

- restarts tasks automatically
- tracks last_run status in Redis:

```
rss_agg:status
```

Main

- heartbeat for monitoring
- status output for mesh and healer

---

8. Future Enhancements (not part of v1.0)

1. **Full article download worker**
2. **Semantic enrichment with embeddings**
3. **Cross-feed deduplication**
4. **Publisher linking derived articles instead of source URLs**
5. **Scoring and ranking (quality/trust filters)**
6. **Front-end API for viewing content from Redis**
7. **WordPress adapter (optional)**