Comprehensive Redis Flow Description — Live vs Historical

This document outlines how Redis keys and queues are created, moved, scheduled, or deleted across the **News**, **SEC Reports**, and **Transcripts** pipelines. Each section clearly separates the **Live** and **Historical** workflows, linking Redis actions to their respective scripts and logic.

---

1. News (Benzinga)

a) WebSocket (`bz_websocket.py`) — Live

- Connects to Benzinga WebSocket stream.
- For each real-time news item:
  - Uses `set_news()`:
    - Stores raw JSON:
      - `news:benzinga:live:raw:{id}.{updated}`
    - Pushes key to:
      - `news:benzinga:queues:raw`
    - Publishes to Redis (optional):
      - `news:benzinga:raw:channel`

b) REST API (`bz_restAPI.py`) — Historical

- Fetches historical news using batch REST queries.
- Uses `set_news_batch()`:
  - Stores each news item as:
    - `news:hist:raw:{id}.{updated}`
  - Pushes key to:
    - `news:queues:raw`
  - On success:
    - Sets: `batch:news:{from}-{to}:fetch_complete`

News Processing (`NewsProcessor.py`) — Live + Historical

- Inherits `BaseProcessor`, monitors:
  - `news:queues:raw` (both live and hist share this queue)
- Each raw key is:
  1. Fetched from Redis
  2. Validated for allowed symbols
  3. Cleaned and normalized
  4. Augmented using `EventReturnsManager.process_event_metadata()`
  5. Stored to:
     - `news:benzinga:{live|hist}:processed:{id}.{updated}`
  6. Pushed to:
     - `news:benzinga:queues:processed`
  7. If live:
     - Publishes to: `news:benzinga:live:processed`

Returns Processing (`ReturnsProcessor.py`) — Live + Historical

- For **Live**: Subscribes to:
  - `news:benzinga:live:processed`
- For **Historical**: Scans and processes in batches:
  - `news:hist:processed:*`
- For each processed item:
  1. Calculates:
     - Hourly, Session, and Daily returns using `market_session.py`
  2. Annotates `metadata.returns_schedule` and actual return values
  3. Writes to:
     - `news:withreturns:{id}` *(if complete)* or
     - `news:withoutreturns:{id}` *(if pending)*
  4. If incomplete:
     - Adds to `news:pending_returns` (ZSET)
- Periodic checks pull from the ZSET:
  - When returns become available, `withoutreturns` → `withreturns` + delete original

---

2. SEC Reports

a) WebSocket (`sec_websocket.py`) — Live

- Receives live 8-K, 10-Q, 10-K filings.
- Uses `set_filing()`:
  - Writes:
    - `reports:live:raw:{accessionNo}.{filedAt}`
  - Pushes to:
    - `reports:queues:raw`

b) REST API (`sec_restAPI.py`) — Historical

- Fetches historical filings.
- Uses `set_filing()` inside batch logic:
  - Stores under:
    - `reports:hist:raw:{accessionNo}.{filedAt}`
  - Pushes to:
    - `reports:queues:raw`
  - Completion flag:
    - `batch:reports:{from}-{to}:fetch_complete`

Report Processing (`ReportProcessor.py`) — Live + Historical

- Inherits `BaseProcessor`, listens to:
  - `reports:queues:raw`

- Each report:
  1. Validated and converted to `UnifiedReport`
  2. Enriched using `EventReturnsManager.process_event_metadata()`
  3. Stored under:
     - `reports:{live|hist}:processed:{accessionNo}.{filedAt}`
  4. Pushed to:
     - `reports:queues:processed`
  5. If live:
     - Publishes to: `reports:live:processed`

Returns Processing (`ReturnsProcessor.py`) — Live + Historical

- For **Live**: Subscribes to:
  - `reports:live:processed`
- For **Historical**: Scans and processes in batches:
  - `reports:hist:processed:*`
- Same logic as news:
  - Updates `reports:withreturns` or `reports:withoutreturns`
  - Adds to:
    - `reports:pending_returns` (ZSET)

---

3. Transcripts

a) REST API (`EarningsCallTranscripts.py`) — Live & Historical

- Fetches and standardizes transcripts to `UnifiedTranscript`
- Calls `store_transcript_in_redis()`:
  - Sets:
    - `transcripts:{live|hist}:raw:{symbol}_{timestamp}`
  - Publishes to:
    - `admin:transcripts:notifications`
  - Also:
    - Pushed to standard `transcripts:queues:raw` for BaseProcessor compatibility

---

Transcript Processing (`TranscriptProcessor.py`) — Live + Historical

- **Combined Approach**:
  - Uses **both** ZSET scheduling and queue processing
  - Runs a scheduling thread for ZSET monitoring in parallel with BaseProcessor
  - Controlled by `ENABLE_LIVE_DATA` feature flag
- **Scheduling Thread**:
  - Monitors `admin:transcripts:schedule` ZSET with scores as timestamps
  - Processes up to 5 due transcripts at once
  - Performs daily date transition checks
  - Sleep timeout limited by `MAX_TRANSCRIPT_SLEEP_SECONDS` (300 seconds)

  - Listens for notifications on `admin:transcripts:notifications` channel
- For each due item in ZSET:
  1. Checks if already processed via `admin:transcripts:processed` SET
  2. Verifies symbol is in tradable universe
  3. Fetches transcript via `EarningsCallProcessor` for exact date
  4. Falls back to current date if no transcript found
  5. Validates fields and standardizes format for BaseProcessor
  6. Enriches with `EventReturnsManager.process_event_metadata()`
  7. Stores in both:
     - `transcripts:{live|hist}:processed:{symbol}_{timestamp}`
     - `transcripts:withreturns:{id}` or `transcripts:withoutreturns:{id}`
- **State Management**:
  - Removes from `admin:transcripts:schedule` ZSET when completed (via `zrem`)
  - Adds to `admin:transcripts:processed` SET for deduplication (via `sadd`)
  - Reschedules via ZSET if not ready yet (5-minute intervals) (via `zadd`)
  - Has error recovery with 30-minute retry on failures
  - Cleans up raw keys after successful processing
  - Publishes notification messages to:
    * `admin:transcripts:notifications` with `processed:{event_key}` or `rescheduled:{event_key}` prefixes

Reconciliation Logic (`reconcile.py`)

- Periodic or triggered reconciliation of missed items.
- Scans:
  - `news:withreturns:*`
  - `reports:withreturns:*`
  - `transcripts:withreturns:*`
- Verifies if node exists in Neo4j:
  - If yes → deletes from Redis
  - If no → re-pushes to `queues:processed`
- Fallbacks:
  - `metadata.created` + `symbol` used to match Redis and DB state
- Optional TTL cleanup logic may remove items after stale duration.

Event Returns (`EventReturnsManager.py`)

- Used in all 3 pipelines.
- Computes:
  - `returns_schedule = {hourly, session, daily}`
  - Each with start/end timestamps
- Logic adapts to:
  - Post-market vs pre-market vs in-session events
- Returns stored in `metadata`:
  - Under `event`, `returns_schedule`, and `{hourly|session|daily}_return`

Redis Structures Summary

Queues:

- `news:queues:raw`
- `news:benzinga:queues:raw`
- `news:benzinga:queues:processed`
- `reports:queues:raw`
- `reports:queues:processed`
- `transcripts:queues:raw` (used for BaseProcessor compatibility)
- `transcripts:queues:processed` (used for BaseProcessor compatibility)

Pub/Sub Channels:

- `news:benzinga:live:processed`
- `reports:live:processed`
- `admin:transcripts:notifications`

ZSETs:

- `news:pending_returns`
- `reports:pending_returns`
- `admin:transcripts:schedule`

Key Patterns:

Raw

- `news:benzinga:live:raw:{id}.{updated}`
- `news:hist:raw:{id}.{updated}`
- `reports:live:raw:{accessionNo}.{filedAt}`
- `reports:hist:raw:{accessionNo}.{filedAt}`
- `transcripts:live:raw:{symbol}_{timestamp}`
- `transcripts:hist:raw:{symbol}_{timestamp}`

Processed

- `news:benzinga:live:processed:{id}.{updated}`
- `news:hist:processed:{id}.{updated}`
- `reports:live:processed:{accessionNo}.{filedAt}`
- `reports:hist:processed:{accessionNo}.{filedAt}`
- `transcripts:live:processed:{symbol}_{timestamp}`
- `transcripts:hist:processed:{symbol}_{timestamp}`

Final

- `news:withreturns:{id}` / `news:withoutreturns:{id}`
- `reports:withreturns:{id}` / `reports:withoutreturns:{id}`
- `transcripts:withreturns:{id}` / `transcripts:withoutreturns:{id}`

Set Membership

- admin:transcripts:processed

Admin

- admin:tradable_universe:stock_universe
- admin:tradable_universe:symbols
- admin:news:last_message_time
- admin:reports:last_message_time
- admin:news:shutdown_state
- admin:reports:shutdown_state
- admin:backfill:news_restart_gap
- admin:backfill:reports_restart_gap
