

Corporate Actions Data Processing Pipeline

Technical Documentation for Handover

1. Executive Summary

The Corporate Actions Data Processing Pipeline is a serverless Azure-based system that automatically processes corporate action email notifications from custodians (primarily Goldman Sachs), extracts structured data using LLM, maintains versioned records in Cosmos DB, and exposes the data through REST APIs for frontend consumption.

Business Problem Solved: Investment operations teams at Hurricane Capital received 500+ corporate action emails monthly (dividends, mergers, stock splits, rights issues) requiring manual processing. This was error-prone, time-consuming, and lacked audit trail for version history.

Key Stakeholders:

- Investment Operations Team (primary users)
- Operations Managers (dashboard consumers)
- Compliance Team (audit trail requirements)

Deployment: Azure Cloud (serverless architecture)

2. Data Ingestion

2.1 Email Ingestion via Azure Logic Apps

Data Source: Corporate email inbox (Exchange Online/Office 365)

- Sender: Goldman Sachs Asset Servicing, other custodians
- Content: Corporate action notifications with structured tables

Trigger Configuration:

- Trigger: "When a new email arrives in inbox"
- Filter Conditions: Subject contains "Corporate Action" OR sender domain matches custodian list
- Actions: Extract email body, download attachments (PDF/HTML), convert to .eml format

Ingestion Flow:

1. Logic App monitors inbox continuously
2. New email matching criteria triggers workflow
3. Email content and attachments extracted
4. Uploaded to Azure Blob Storage `raw-emails` container
5. Event Grid notification triggers processing pipeline

2.2 Azure Blob Storage Containers

Container	Purpose	Lifecycle
<code>raw-emails</code>	Original .eml files from Logic Apps	Retained 1 year
<code>processed-data</code>	Intermediate JSON files (raw_emails.json)	Retained 90 days
<code>processed-emails</code>	Final normalized JSON for Azure Functions	Retained 1 year
<code>json-processing-errors</code>	Failed files for manual review	Retained indefinitely
<code>final-data</code>	Backup copies for disaster recovery	Retained 2 years

2.3 File Formats Handled

- `.eml`: Email message format from Logic Apps
- `.html`: Email body content (Goldman Sachs format)
- `.pdf`: Attached corporate action documents
- `.json`: Processed structured data

2.4 Volume

- 500+ emails per month
- Peak: 50+ emails per day during earnings season
- Average email size: 50KB (HTML) to 2MB (with PDF attachments)

3. Data Cleaning & Preprocessing

3.1 Email Parsing

Email Content Extraction:

- Parse .eml file to extract headers, body, and attachments
- Convert HTML body to text while preserving table structure
- Extract embedded images if present (rare)

Header Fields Extracted:

- `email_id`: Unique message identifier (used for version tracking)
- `received_date`: Timestamp when email arrived
- `subject`: Email subject line
- `sender`: Sender email address

3.2 HTML Table Parsing

Goldman Sachs Email Format:

- Corporate action details presented in HTML tables
- Multiple tables per email (event details, account details, terms)
- Nested tables for complex events (rights issues, mergers)

Parsing Approach:

- BeautifulSoup for HTML parsing
- Identify table structure by header row patterns
- Extract key-value pairs from two-column tables
- Extract row data from multi-row tables (accounts, terms)

3.3 Schema Validation

Pydantic Models:

- `EmailData`: email_id, received_date, subject, body, accounts, events
- `EventData`: event_id, event_type, security, dates, terms, status
- `AccountData`: account_number, position, amounts, deadline

Validation Rules:

- Required fields: event_id, event_type, security.isin
- Date format normalization (see Issues section)

- Currency code validation (ISO 4217)
 - Numeric field parsing (handle commas, parentheses for negatives)
-

4. Data Processing & Transformation

4.1 LLM Processing Pipeline

Step 1: Azure Document Intelligence (for PDF attachments)

- Prebuilt document model for structured extraction
- Table detection and parsing
- Key-value pair extraction

Step 2: LangChain Structured Output

- Chain: Document → Prompt Template → Azure OpenAI → Structured JSON
- Pydantic models define expected output schema
- Temperature: 0.1 (for consistent extraction)

Extracted Fields:

- Event Type: CASH_DIVIDEND, STOCK_SPLIT, MERGER, RIGHTS_ISSUE, TENDER_OFFER, etc.
- Security Info: CUSIP, ISIN, ticker, company name
- Key Dates: announcement_date, record_date, pay_date, ex_date, deadline
- Financial Details: rate, ratio, currency, gross_amount, net_amount, tax_withheld
- Account Details: account_number, position, entitled_amount

4.2 Event Identification

Event ID Generation:

- Composite key: {event_type}_{isin}_{record_date}_{announcement_id}
- Used to link multiple emails for same corporate action

Event Matching Logic:

1. Parse incoming email for event identifiers
2. Query Cosmos DB for existing event with same event_id

3. If new → Create document with version=1
4. If existing → Update document, increment version, archive previous state

4.3 Version Management

Version History Schema:

```
{
  "event_id": "CASH_DIV_US14040H1059_20250304_83894241",
  "version": 11,
  "version_history": [
    {
      "version": 10,
      "changed_date": "2025-03-04T19:20:50Z",
      "email_ref": "MSG12345_extracted",
      "state": { ... previous state ... },
      "changes": ["status", "pay_date"]
    },
    ...
  ],
  "latest_update": {
    "email_ref": "MSG12346_extracted",
    "update_date": "2025-03-04T19:22:47Z",
    "changed_fields": ["status"],
    "change_summary": "Status changed from Pending to Payment Released"
  }
}
```

Version Tracking Benefits:

- Complete audit trail for regulatory compliance
- Ability to see event evolution over time
- Email lineage tracking (which email caused which change)

5. AI/ML Implementation

5.1 Azure OpenAI Integration

Model: GPT-4 via Azure OpenAI Service

- Deployment: East US 2 region
- Used for: Complex field extraction, entity recognition, date interpretation

Structured Output Approach:

- Define Pydantic schema for expected output
- Use LangChain's `with_structured_output()` method
- Model responds with JSON matching schema

5.2 LangChain Pipeline

Chain Components:

1. Document Loader: Load processed email content
2. Prompt Template: Instructions for extraction with few-shot examples
3. Azure OpenAI LLM: GPT-4 for extraction
4. Output Parser: Pydantic model validation

Prompt Engineering:

- System prompt defines role as "corporate actions data extractor"
- Few-shot examples for each event type
- Explicit instructions for date format handling
- Instructions to return null for missing fields (not hallucinate)

5.3 Azure Document Intelligence

Use Case: PDF attachment processing

- Model: Prebuilt document model
- Features: Table extraction, key-value pairs, layout analysis

Integration:

- Called when email has PDF attachment
 - Output merged with HTML body extraction
 - PDF content prioritized for conflicting fields
-

6. Azure Services & Architecture

6.1 Complete Azure Stack

Service	Purpose	Configuration
Azure Logic Apps	Email monitoring and ingestion	Exchange Online connector, blob output
Azure Blob Storage	File storage across pipeline stages	5 containers with lifecycle policies
Azure Functions	Serverless processing (Python v2)	2 function apps: JsonEventExtractor, EventDocumentManager
Azure Cosmos DB	Document database with versioning	NoSQL API, partition key: <code>/event_id</code> , 400 RU/s
Azure OpenAI Service	LLM for data extraction	GPT-4 deployment
Azure Document Intelligence	PDF processing	Prebuilt document model
Azure API Management	API gateway for frontend	REST endpoints with RBAC
Azure Key Vault	Secrets management	Connection strings, API keys
Application Insights	Monitoring and logging	Connected to all functions

6.2 Azure Functions Architecture

Function 1: JsonEventExtractor

- Trigger: Blob trigger on `processed-emails` container
- Input: JSON file with parsed email data
- Processing: Extract events, validate schema, normalize data
- Output: Call EventDocumentManager for each event
- Error Handling: Move failed files to `json-processing-errors` container

Function 2: EventDocumentManager

- Trigger: Direct invocation from JsonEventExtractor
- Input: Normalized event data with email_id
- Processing: Query Cosmos DB, create or update document, manage versions

- Output: Upsert to Cosmos DB

6.3 Cosmos DB Configuration

- Database: `CorpActionsDB`
 - Container: `events`
 - Partition Key: `/event_id`
 - Throughput: 400 RU/s (auto-scale enabled)
 - Indexing: Composite indexes on (event_type, security.isin), (account_number)
-

7. Operationalization & Deployment

7.1 Infrastructure as Code

Bicep Templates:

- `main.bicep`: Orchestrates all resource deployments
- `storage.bicep`: Blob storage with containers
- `functions.bicep`: Function apps with app settings
- `cosmos.bicep`: Database and container creation
- `apim.bicep`: API Management configuration

7.2 Deployment Process

Local Development:

1. Azure Functions Core Tools for local testing
2. Azurite for local blob storage emulation
3. Cosmos DB emulator for local database

Azure Deployment:

1. Run Bicep deployment: `az deployment group create`
2. Deploy function code via VS Code Azure Functions extension
3. Configure Logic Apps connection to Exchange Online
4. Set up API Management policies

7.3 Environment Configuration

Function App Settings:

- `COSMOS_CONNECTION_STRING`: From Key Vault reference
 - `STORAGE_CONNECTION_STRING`: From Key Vault reference
 - `AZURE_OPENAI_ENDPOINT`: Service endpoint
 - `AZURE_OPENAI_KEY`: From Key Vault reference
 - `DOCUMENT_INTELLIGENCE_ENDPOINT`: Service endpoint
-

8. Monitoring & Observability

8.1 Logging Strategy

Structured Logging:

- All functions use Python `logging` module
- JSON format for structured queries
- Fields: timestamp, function_name, event_id, email_id, action, duration_ms

Log Levels:

- INFO: Successful processing, version updates
- WARNING: Retries, fallback paths taken
- ERROR: Failed processing, validation errors

8.2 Application Insights Integration

Tracked Metrics:

- Function execution count and duration
- Success/failure rates
- Cosmos DB RU consumption
- OpenAI API latency

Custom Events:

- `event_created`: New corporate action detected

- `event_updated`: Existing event modified
- `version_incremented`: Version history updated
- `processing_error`: Failed to process email

8.3 Alerting

Alert	Condition	Action
High Error Rate	> 10% failures in 1 hour	Email to ops team
Processing Delay	> 30 min since last successful process	Slack notification
Cosmos DB Throttling	429 errors detected	Scale up RUs

9. User Consumption

9.1 API Endpoints

Endpoint	Method	Purpose
<code>/api/events</code>	GET	List events with filtering (status, type, date range)
<code>/api/events/{eventId}</code>	GET	Get specific event with full version history
<code>/api/accounts/{accountId}/events</code>	GET	List events for specific account
<code>/api/events/history/{eventId}</code>	GET	Get complete version history timeline
<code>/api/securities/{isin}/events</code>	GET	List events for specific security
<code>/api/emails/{eventId}/versions/{versionId}/html</code>	GET	Retrieve original email HTML

9.2 Response Format

Event List Response:

```
{
  "events": [...],
  "pagination": {
    "page": 1,
    "pageSize": 20,
    "totalCount": 150
  }
}
```

Event Detail Response:

- Event metadata (type, security, dates)
- Current state (all field values)
- Account details array
- Terms details array
- Version history with change summaries

9.3 Frontend Integration

Operations Manager Dashboard:

- Real-time event feed with status indicators
- Version timeline visualization
- Click-through to original email HTML
- Account-level filtering

Data Formatting:

- Dates displayed in user's timezone
- Amounts formatted with currency symbols
- Status badges (Pending, Released, Cancelled)

10. Issues Faced & Resolutions

10.1 Date Format Inconsistency

Problem: Emails used multiple date formats causing version history disorder:

- ISO format: `2025-03-04T08:50:50-0500`

- Human-readable: `Tue, 04 Mar 2025 08:30:11 -0500`
- Short format: `Mar 04 2025`

Symptoms: Version 11 showed timestamp after version 8, 9, 10 in UI.

Resolution:

- Created `parse_date()` function with 15+ format patterns
- All dates normalized to ISO 8601 format before storage
- Added `expected_year` parameter for dates missing year
- Timezone stripped and stored as UTC

10.2 Chronological Ordering Issues

Problem: Version history not displaying in proper chronological order.

Root Cause: `changed_date` was being copied from previous version instead of using current timestamp.

Resolution:

- Generate `current_timestamp = datetime.now(timezone.utc).isoformat()` at processing time
- Use same timestamp for both `version_history.changed_date` and `latest_update.update_date`
- Sort version history by timestamp before display

10.3 Email ID Extraction Missing

Problem: Version tracking couldn't link updates to source emails.

Root Cause: `email_id` not being extracted from JSON and passed to Cosmos DB.

Resolution:

- Updated `extract_events()` function to include `email_id` parameter
- Added `email_ref` field to version history entries
- Implemented `get_html_url()` function to construct blob URL from email_id

10.4 Blob Trigger Timing Issues

Problem: Function triggered before file fully uploaded, causing partial reads.

Resolution:

- Added 5-second delay after blob creation before processing

- Implemented file size check (retry if size changes)
- Used blob lease to ensure exclusive access

10.5 Azure Function Cold Start Delays

Problem: First request after idle period took 10-15 seconds.

Resolution:

- Enabled "Always On" for production function app (requires App Service Plan)
- Implemented warm-up endpoint called by Logic Apps health check
- Reduced package size by removing unused dependencies

10.6 Duplicate Event Processing

Problem: Same email processed multiple times due to Logic App retries.

Resolution:

- Added idempotency check using email_id hash
- Store processed email_ids in Redis cache (24-hour TTL)
- Skip processing if email_id already processed

10.7 HTML Content Retrieval Failures

Problem: Original email HTML not loading in UI.

Root Cause: Blob URL construction had `_extracted` suffix handling issue.

Resolution:

- Updated `get_html_url()` to strip `_extracted` suffix from email_id
 - Added fallback to Microsoft Graph API if blob not found
 - Implemented caching layer for frequently accessed emails
-

11. Lessons Learned & Best Practices

11.1 What Worked Well

1. **Event-driven serverless architecture:** Scaled automatically with email volume
2. **Version history pattern:** Met all audit requirements without complex schema

- 3. **Partition key choice:** `event_id` enabled efficient queries and even distribution
- 4. **Pydantic validation:** Caught data issues early in pipeline

11.2 What We'd Do Differently

- 1. **Standardize date handling from day one:** Date format inconsistency caused significant debugging
- 2. **Implement dead-letter queue:** Failed messages were harder to retry without DLQ
- 3. **Add correlation IDs earlier:** Tracing issues across functions was challenging
- 4. **Use Durable Functions:** Would simplify orchestration and add automatic retry

11.3 Recommendations for Similar Projects

- 1. **Normalize all dates to ISO 8601 UTC immediately** upon ingestion
- 2. **Include email_id/source reference** in all downstream data
- 3. **Implement idempotency** for all message-triggered functions
- 4. **Test with production email formats** early (custodian formats vary significantly)
- 5. **Build version comparison logic** that handles missing/null fields gracefully

12. Technology Stack Summary

Category	Technologies
Serverless Compute	Azure Functions (Python v2 programming model)
Integration	Azure Logic Apps, Event Grid
Database	Azure Cosmos DB (NoSQL API)
Storage	Azure Blob Storage (5 containers)
AI/ML	Azure OpenAI (GPT-4), Azure Document Intelligence
Framework	LangChain, Pydantic
API Gateway	Azure API Management
Monitoring	Application Insights, Azure Monitor
Security	Azure Key Vault, Managed Identities, Azure AD
IaC	Bicep templates

13. References & URLs

- Azure Functions Python Developer Guide: (<https://learn.microsoft.com/en-us/azure/azure-functions/functions-reference-python>)
- Azure Cosmos DB NoSQL API: (<https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/>)
- Azure Logic Apps Email Connector: (<https://learn.microsoft.com/en-us/azure/connectors/connectors-create-api-office365-outlook>)
- LangChain Structured Output: (https://python.langchain.com/docs/modules/model_io/output_parsers/)

Document Version: 1.0 Last Updated: January 2025 Author: Gagandeep Singh