Bocconi Students Investment Club

BSIC Quant Library: Streamlined Financial Data Retrieval

A Python Framework for Market Data

Authors

Bocconi Students Investment Club

Tancredi Liani (Project Lead)

Giacomo Cirò

Luca Aron

Martin Patrikov

Matteo Mendicini

December 2025

bsic.it — Bocconi University, Milan

Abstract

This technical report details the architecture and implementation of the BSIC Quant Library, a Python-based infrastructure designed to centralize and optimize financial data retrieval for the BSIC members. The system is designed to facilitate access to financial data from the association's members by providing a robust pipeline for data crawling, storage, and retrieval.

The library utilizes a technology stack featuring DuckDB for fast querying, AWS S3 for scalable cloud storage, and Apache Parquet for efficient columnar data formatting. Key architectural features include a secure authentication mechanism via AWS SSO, strict environment isolation (Development vs. Production), and a flexible crawling template.

By distinguishing storage (S3) from compute (DuckDB), the library enables members to query massive financial datasets efficiently without local storage overhead, while providing developers with a structured CI/CD workflow for data ingestion. This infrastructure serves as the backbone for future quantitative research in BSIC, and it will be upgraded with the addition of many new types of financial assets.

All the views expressed are opinions of Bocconi Students Investment Club members and can in no way be associated with Bocconi University. All the financial recommendations offered are for educational purposes only. Bocconi Students Investment Club declines any responsibility for eventual losses you may incur implementing all or part of the ideas contained in this website. The Bocconi Students Investment Club is not authorised to give investment advice. Information, opinions, and estimates contained in this report reflect a judgment at its original date of publication by Bocconi Students Investment Club and are subject to change without notice. The price, value of and income from any of the securities or financial instruments mentioned in this report can fall as well as rise. Bocconi Students Investment Club does not receive compensation and has no business relationship with any mentioned company.
Copyright © 2025 BSIC — Bocconi Students Investment Club

Contents

1 Project Overview

1.1 Purpose and Scope

The **BSIC Quant Library** is a Python-based quantitative finance data infrastructure designed to ease the process of data retrieval. It serves two different user groups within the association:

- **Association Members (End Users):** Provides efficient and secure access to the centralized BSIC database of financial market data, enabling rapid fetching without the need for advanced data engineering knowledge.
- **Developers:** Provides a framework for crawling external data sources, validating schema integrity, and uploading structured data to the centralized AWS S3 repository.

1.2 Key Features

The library is built following modern data engineering standards, to ensure scalability and ease of use:

- **High-Performance Query Engine:** Utilizes DuckDB to execute queries directly on Parquet files, minimizing data transfer and latency.
- **Cloud Storage:** Leverages AWS S3 with the Apache Parquet file format for efficient storage that reduces costs and improves read speeds.
- **Secure Authentication:** Implements AWS SSO profiles to manage access, eliminating the risks associated with long lived credentials.
- **Environment Isolation:** Enforces strict separation between Development and Production environments to ensure data integrity.
- **Extensible Architecture:** Features a flexible crawler design that allows for easy integration of new data providers (e.g., Yahoo Finance, FRED API, US Treasury API).

1.3 Technology Stack

The infrastructure relies on a detailed selection of open source technologies:

Table 1: BSIC Quant Library Technology Stack

Component	Technology	Version	Role
Language	Python	3.12.3	Core Logic
Query Engine	DuckDB	≥1.1.3	SQL execution
Cloud Storage	AWS S3	—	Object Storage
Data Format	Apache Parquet	≥18.0.0	Columnar Storage (via PyArrow)
AWS SDK	boto3	≥1.35.0	Cloud Interaction
Data Processing	pandas/numpy	≥2.3.3	Data Manipulation
Financial API	yfinance	≥0.2.66	Market Data Source
Package Mgr	uv	—	Dependency Management
Testing	pytest	≥8.4.2	Unit & Integration Testing

2 System Architecture

2.1 High-Level Architecture

The system follows a layered architecture that separates the data persistence layer from the ingestion and retrieval logic.

The architecture flows through three primary layers:

1. **The Storage Layer (AWS S3):** The foundation of the system is the S3 bucket structure, partitioned by environment (Prod/Dev). Data is stored as Parquet files, typically organized by ticker symbol (e.g., `s3://bsic-database-prod/data/AAPL.parquet`).
2. **The Middleware Layer (DBHandler):** Acting as the central operator of the system, the `DBHandler` links the cloud storage and the application. It configures the DuckDB engine with valid AWS credentials and facilitates data movement (uploading processed data and querying historical data).
3. **The Application Layer:**
   - **Crawlers (Ingestion):** Automated scripts that fetch raw data from external APIs, normalize it, and push it to the Middleware.
   - **BsicDB (Retrieval):** The user-facing API that associates use to request assets (e.g., `db.get('AAPL')`).

2.2 Module Organization

The codebase is structured into three main modules within the `bsic` package to clearly separate the scopes:

- **`bsic.common`**: Contains shared utilities and core infrastructure code.
  - `dbhandler.py`: The S3 + DuckDB database handler.

  - `asset.py`: Domain models for financial instruments (Stock, Bond).

- **`bsic.etl`**: Manages the Extract-Transform-Load pipeline.
  - `crawler.py`: Abstract base class defining the ingestion interface.
  - `yahoo_finance_crawler.py`: Concrete implementation for yfinance.
- **`bsic.quant`**: The user-facing interface.
  - `client.py`: Wrapper for the `BsicDB` client.

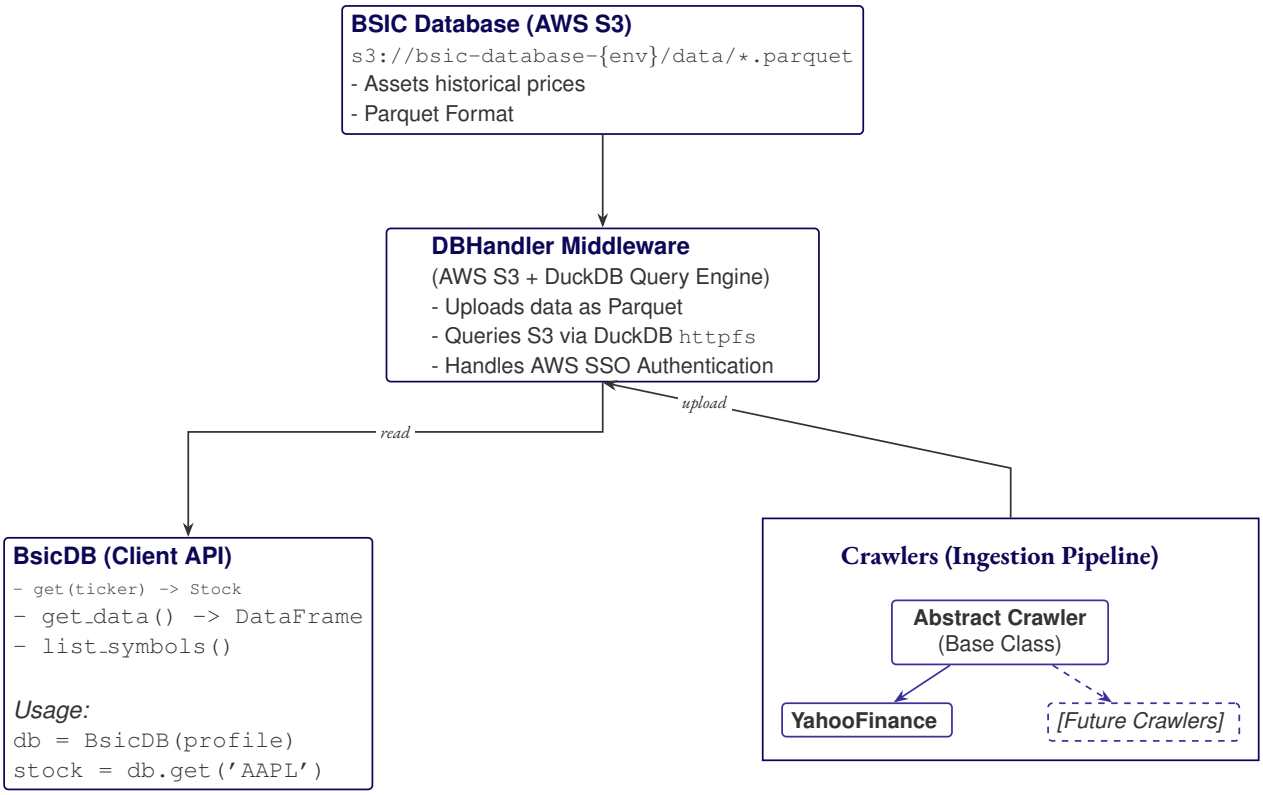

Figure 1: BSIC Quant Library System Architecture

2.3 Common Module: The Core Infrastructure

2.3.1 DBHandler

The `DBHandler` is the most critical component of the library. Upon initialization, it creates a `boto3` session using the user's AWS SSO profile. Critically, it extracts temporary "frozen" credentials (access key, secret key, and session token) and injects them directly into the DuckDB configuration.

This allows DuckDB to perform `httpfs` operations (reading remote files over HTTP) securely without requiring the user to manually manage keys.

Listing 1: DBHandler Credential Injection

```
# Configure DuckDB with frozen credentials
```

```
conn.execute(f"SET s3_access_key_id='{creds.access_key}';")
conn.execute(f"SET s3_secret_access_key='{creds.secret_key}';")
conn.execute(f"SET s3_session_token='{creds.token}';")
conn.execute(f"SET s3_region='{DEFAULT_REGION}';")
```

2.3.2 Asset Domain Model

The library defines precise domain models. The `Asset` abstract base class ensures that all financial instruments have a fundamental `asset_class` attribute. The `Stock` subclass extends this to include the ticker symbol and the associated pandas DataFrame.

2.4 ETL Module: Data Ingestion

2.4.1 The Crawler Pattern

To support future expansion (e.g., adding Bonds data), the library uses an abstract `Crawler` base class. Concrete implementations must override the `crawl()` method.

The **Yahoo Finance Crawler** implements this pattern with a configuration based approach. It reads a YAML file defining the range of tickers and the desired time span.

Listing 2: Crawler Usage Example

```
# yahoo_finance_crawler_config.yaml
start: null         # Fetch max history
period: 1y          # Lookback period
tickers:
  - AAPL
  - NVDA
```

The crawler performs a strict validation pipeline before upload:

1. **Schema Check:** Ensures columns [”Open”, ”High”, ”Low”, ”Close”, ”Volume”] exist.
2. **Type Check:** Verifies all price columns are numeric.
3. **Sanity Check:** Rejects data with negative prices or negative volume.
4. **Format:** Sets a standard `DatetimeIndex` and removes duplicates.

2.5 Quant Module: Data Retrieval

The `BsicDB` client provides an API for end-users. It abstracts the complexity of SQL generation and S3 connectivity.

Listing 3: Client API Usage

```
from bsic.quant.client import BsicDB
```

```
# Initialize with SSO profile
db = BsicDB(profile='bsic-dev-profile')

# Simple object retrieval
stock = db.get('AAPL')

# Advanced querying with predicate pushdown
df = db.get_data(
    'AAPL',
    columns=['date', 'Close'],
    filter_condition="Close > 150 AND date >= '2020-01-01'"
)
```

3 Data Flow & Optimization

3.1 Ingestion Pipeline (ETL)

The flow of data into the system is linear:

$$\text{External Source} \rightarrow \text{Crawler} \rightarrow \text{Validation} \rightarrow \text{Parquet Conversion} \rightarrow \text{S3 Upload}$$

3.2 Retrieval Pipeline and Query Optimization

The retrieval process leverages DuckDB's advanced optimization capabilities:

1. **Projection Pushdown:** If a user requests only the "Close" price, DuckDB reads only that specific column from the Parquet file on S3, ignoring the other columns. This significantly reduces network I/O.
2. **Predicate Pushdown:** If a user filters by `date > '2024-01-01'`, DuckDB utilizes Parquet metadata (row groups) to skip reading parts of the file that do not contain relevant data.

4 Development Security

4.1 Development Workflow

The library utilizes modern Python tooling to maintain code quality:

- **Package Management:** Uses `uv` for extremely fast dependency resolution and package accessibility.
- **Linting:** `Ruff` is used for both linting and formatting, enforcing strict formatting rules and import sorting.

- **CI/CD:** Pipeline that ensures that linting, tests and deployments work as intended on every merged pull request and commit.

4.2 Testing Strategy

Testing is organized into three tiers:

1. **Unit Tests:** Offline tests that verify logic (e.g., data validation rules) using mocked services when needed.
2. **Integration Tests:** Verify the interaction between the Crawler and the DBHandler.
3. **E2E Tests:** Marked with `@pytest.mark.e2e`, these perform actual network calls to valid S3 buckets and external APIs (skipped by default in CI).

4.3 Security Considerations

- **No Hardcoded Credentials:** The system relies entirely on temporary AWS SSO tokens.
- **Bucket Isolation:** Bucket names in the `DBHandler` prevent accidental writes to Production from a Development environment.
- **Read-Only Defaults:** Members are granted read-only permissions via AWS IAM policies, with write access reserved for developers.

5 Conclusion

The BSIC Quant Library was built to lay the foundations for future data related innovation that could facilitate the association's processes in any way. It is supposed to be the starting point for more complex and specific data retrieval pipelines that might be implemented in the future. The architecture's modularity allows for upcoming expansion into different asset classes to ensure that all members' needs are satisfied.