# Technical Case Study: Data Explorer (Technical Leadership Edition)

## Executive Summary

Data Explorer was designed and delivered to address a critical gap in data integration and analysis across the CTO’s enterprise application estate. The initiative introduced a unified, secure, and scalable data integration layer, built on containerized microservices and governed data pipelines. The platform enables CTO and architecture teams to perform cross-system analysis, validate data integrity, and improve decision velocity through automation and visualization.

## Background and Problem Statement

The CTO organization manages a complex ecosystem of applications—LCT, Jira, ServiceFirst, Hypercube, and multiple internal data sources. These systems store overlapping but inconsistent information related to business outcomes, release tracking, and service metrics. The absence of a consolidated, queryable data view created challenges in data-driven governance and led to inefficiencies such as manual reconciliation, error-prone reporting, and limited visibility.

Our core challenge was to design a secure, high-performance integration platform capable of aggregating and reconciling heterogeneous data sources, while adhering to Barclays’ internal security and data protection standards.

## Objectives and Leadership Role

I served as the technical architect and initiative lead, responsible for designing the architecture, defining non-functional requirements, and overseeing implementation across distributed teams. My role involved selecting technology stacks, managing DevSecOps integration, and ensuring the solution aligned with CTO’s data governance strategy. Additionally, I engaged directly with senior leadership to align architectural goals with strategic priorities and secured cross-functional collaboration across application, data, and infrastructure domains.

## Architecture Overview

Data Explorer is built using a distributed microservices architecture with two key services: the Data API and the Data Loader. Both are implemented in Python using Flask and FastAPI frameworks. Dagster orchestrates ETL workflows, providing a declarative approach to scheduling and monitoring data jobs. Each component is containerized using Docker and deployed through GitLab CI/CD onto the OpenShift platform. Authentication is managed via BAM, and secure secrets management is integrated through CSM.

Data flows from upstream systems (LCT, Jira, ServiceFirst, Hypercube, AWS Billing, etc.) into the Data Loader pipeline. Dagster schedules extraction and transformation jobs, applies data quality validations, and stores normalized data in an on-premises S3 object store in Parquet format. The Data API exposes REST endpoints for Tableau and other consumers to query data dynamically. Tableau dashboards then visualize KPIs, enabling leadership to track metrics such as release health and service trends.

## Key Technical Decisions and Rationale

- \*\*Dagster over Airflow\*\*: Chosen for its Pythonic syntax, modularity, and compatibility with existing internal libraries. It allowed easier CI/CD integration and dynamic pipeline configuration compared to Airflow’s DAG management.  
- \*\*Object Store over Relational Database\*\*: Selected S3-compatible object storage for better scalability, cost efficiency, and flexibility in handling unstructured and semi-structured data.  
- \*\*Parquet Format\*\*: Adopted Parquet over Pickle for performance and compression benefits. Parquet’s columnar storage model reduces IO overhead and integrates efficiently with Spark and Pandas for downstream analytics.  
- \*\*FastAPI and Flask\*\*: Provided lightweight, asynchronous APIs capable of handling concurrent data queries. The REST APIs expose well-documented endpoints for integration with Tableau and other BI tools.  
- \*\*GitLab CI/CD with OpenShift\*\*: Ensured automated deployment, container scanning (via Prisma), and continuous integration, enabling rapid yet secure iteration cycles.

## Implementation and Integration

The deployment leveraged infrastructure-as-code principles. GitLab CI/CD pipelines handled container builds, static analysis via SonarQube, and security scans via Prisma. Artifacts were versioned and deployed to Nexus before being released to OpenShift clusters. Each microservice was subjected to over 90% unit test coverage, ensuring operational reliability.

Integration across systems included APIs for ServiceFirst (CMDB and Change Management), SPDW for service metrics, Hypercube APIs for business data, and AWS Billing reports. Tableau’s consumption layer accessed Data Explorer through short-lived, pre-signed S3 URLs, automatically generated for each user session to maintain data security and auditability.

## Security and Non-Functional Requirements

Security and compliance were embedded in the architecture from design stage. SSL/TLS was enforced across all endpoints, and BAM authentication restricted API access to authorized users. CSM managed key rotation, while Dagster pipelines ran with least-privilege access.

Scalability was achieved by decoupling ETL and API layers, allowing independent scaling based on load. Reliability was reinforced with retry logic at the job level and job failure notifications through monitoring alerts. Resiliency was classified under ResCat 4, ensuring compliance with CTO reliability standards.

## Results and Business Impact

The platform eliminated manual reporting dependencies, reducing data collation time by 65% and improving the accuracy of governance reports. The automated pipelines provided real-time updates across key systems, enabling CTOs to make faster, evidence-based decisions. Additionally, Data Explorer became the foundation for ongoing data automation efforts within the CTO domain, influencing future data initiatives.

## Leadership and Technical Mentorship

Throughout development, I led cross-functional design reviews, mentored engineers on API and data pipeline development, and established coding and deployment standards. I introduced design patterns for secure data access and modular pipeline design, which have since been adopted across other CTO initiatives. This role strengthened my capability to influence architecture strategy and align teams toward shared goals.

## Future Roadmap and Innovation

The next phase of Data Explorer focuses on integrating Azure AI and LLM models for automated report summarization and anomaly detection. This will extend the platform’s analytical capabilities, allowing predictive insights and further reducing manual interpretation of complex datasets.

## Conclusion

Data Explorer demonstrates how deep technical leadership, paired with architectural rigor, can drive organizational impact. It stands as a reference model for secure, scalable data integration and analytics in a complex enterprise ecosystem—delivering both technical excellence and measurable business value.