Toward a Trustworthy and Accessible Scientific Data Workflow Platform with StreamCI

Jaewoo Shin
shin152@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

Mehak Jain
jain1002@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

Shubham Jha
jha70@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

I Luk Kim
kim1634@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

Mohana Sravya Appalaneni
mappala@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

Tri Minh Hoang
hoang45@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

Jorge Iván Fuentes Rosado
jifuente@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

Elham J Barezi
ejebalba@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

Lan Zhao
lanzhao@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

Carol X. Song
cxsong@purdue.edu
Rosen Center for Advanced Computing, Purdue University
West Lafayette, Indiana
USA

Abstract

Scientific research workflows increasingly involve not only structured streaming data but also raw artifacts and derived products, requiring platforms that provide trustworthy data protection and accessible interfaces beyond simple ingestion and storage. We present extensions to StreamCI, a cloud-based streaming data management platform, that evolve it into an end-to-end scientific data workflow platform. Key advancements include expanded data lifecycle support (blob ingestion, raw data refinement, and derived data generation), a migration from RabbitMQ to Apache Kafka for scalable dataflow infrastructure, automated backup mechanisms for data reliability, and a researcher-friendly web portal paired with a Python client API library (PyStreamCI). We demonstrate these capabilities through end-to-end workflows from active research use cases and report on operational experience following deployment on Purdue University's Anvil cloud infrastructure. In production, these extensions enable researchers across domains—such as building energy sustainability, pavement condition monitoring, and precision audiology—to manage complete data workflows, from raw artifacts to analysis-ready products, without requiring backend expertise.

CCS Concepts

• **Software and its engineering**; • **Computer systems organization → Cloud computing**; • **Information systems → Computing platforms**;

Keywords

StreamCI, scientific data workflows, streaming data, cyberinfrastructure, data refinement, derived data generation

ACM Reference Format:
Jaewoo Shin, Mehak Jain, Shubham Jha, I Luk Kim, Mohana Sravya Appalaneni, Tri Minh Hoang, Jorge Iván Fuentes Rosado, Elham J Barezi, Lan Zhao, and Carol X. Song. 2026. Toward a Trustworthy and Accessible Scientific Data Workflow Platform with StreamCI. In *Practice and Experience in Advanced Research Computing (PEARC '26), July 26–30, 2026, Minneapolis, MN, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3785462.3815877

PEARC '26, Minneapolis, MN, USA

ACM ISBN 979-8-4007-2377-3/2026/07
https://doi.org/10.1145/3785462.3815877

1 Introduction

Modern scientific research is increasingly driven by continuous data streams from IoT sensors, smart facilities, and field instruments. StreamCI [5] is a scalable, cloud-based sensor data management system that enables researchers to collect, process, store, and access streaming data through APIs and a web portal. Deployed on

Kubernetes, StreamCI has supported research in plant phenotyping, building energy, ecology education [5, 6], and other domains. The AXIN data lake prototype subsequently extended StreamCI with Apache Flink-based streaming processing and ML/AI modeling workflows [6].

As adoption grew and workflows became more complex, several gaps emerged. This paper presents extensions to StreamCI along four key axes: (1) **data lifecycle expansion** through blob data ingestion, raw data refinement, and derived data generation, the last powered by a JSON-configurable service that automatically generates Apache Flink SQL jobs so researchers obtain derived data products without writing stream-processing code; (2) **scalable dataflow infrastructure** via migration from RabbitMQ to Apache Kafka for persistent, log-based messaging; (3) **data reliability** through automated incremental and full backup mechanisms; and (4) **researcher accessibility** via a redesigned user portal and PyStreamCI, a Python client API library.

These extensions address immediate workflow needs while establishing a scalable and extensible platform foundation. The remainder of this paper is organized as follows: Section 2 motivates these extensions; Section 3 describes each extension in detail; Section 4 presents end-to-end workflows; Section 5 reports on operational experience; and Section 6 concludes.

2 Motivation

As StreamCI's user base expanded across multiple research domains, three specific gaps emerged.

Support for heterogeneous data beyond structured records. Several use case teams needed to store and process raw binary artifacts alongside structured sensor streams. For example, a pavement assessment group collects hundreds of gigabytes of imagery from vehicle-mounted sensors that must be stored and linked with structured GPS metadata. Building energy researchers needed automated operations such as unit conversions, outlier filtering, and temporal aggregations on incoming sensor data before it could be used for modeling. The initial StreamCI platform only supported text-based record ingestion and lacked built-in processing pipelines, leaving these lifecycle stages outside the platform.

Protection against data loss. As the volume of research data managed by StreamCI grew rapidly, the absence of systematic backup and recovery mechanisms became a critical risk. A single hardware failure or operational error could result in the loss of months of data, undermining trust in the platform as a production repository.

Accessible interfaces for non-expert users. StreamCI's user community now includes domain researchers (audiologists, agricultural scientists, civil engineers, ecologists, and education users) alongside research software engineers. The original portal lacked capabilities for pipeline configuration, flexible schema management, or fine-grained access control, making it difficult for domain researchers to manage their workflows independently.

3 Extending StreamCI

Figure 1 shows the updated StreamCI architecture, integrating the extensions described in this section.

3.1 Blob Data Support, Refinement, and Derivation

The original StreamCI supported only text-based, record-oriented data in JSON and GeoJSON formats. To address evolving research needs, we extended the system with three capabilities that together manage the full scientific data lifecycle.

Blob data support enables the platform to treat raw binary files, such as images, video, and sound recordings from scientific instruments, as first-class data objects. Blobs are stored in object storage with metadata maintained in a database, allowing users to query and retrieve raw artifacts alongside structured data within a unified environment.

Raw data refinement applies user-configured processing operations to incoming data streams at ingestion time, producing cleaner and more standardized data without losing the original records. The refinement engine executes a metadata-driven, user-defined sequence of operations configured through the portal. Users control the exact sequence of transformation operations, enabling workflows such as converting a temperature unit and then filtering on the converted value.

Supported operations include quality checks (flagging, dropping, or quarantining), filtering, and field transformations such as enrichment and remapping. The operation sequence is validated at configuration time through schema checking and dependency analysis. At runtime, passing records are stored in the primary collection, while quarantined and dropped records are isolated or logged for auditing.

Derived data generation complements refinement by operating on stored data to produce new, higher-level data products which are persisted in separate collections. It supports both real-time processing of continuously arriving data and historical backfill over existing stored records. This capability is powered by Apache Flink [3] through a JSON-configurable SQL generation service that automates job creation and deployment. Users define data sources, sinks, and transformation logic either programmatically or through the portal interface, and the service generates and submits the corresponding Flink SQL jobs to the cluster. Supported transformations include joins across multiple data sources, window aggregations (tumbling, hopping, and session windows), column projections, and rule-based conditional logic. By generating Flink SQL automatically from declarative JSON specifications, this service lets researchers deploy production stream-processing jobs without writing or operating Flink code directly—a key accessibility abstraction of the platform.

Together, refinement and derivation represent two complementary processing paradigms: refinement cleans and standardizes individual incoming records during ingestion, while derivation performs analytical transformations on stored data to produce new datasets and insights.

3.2 Scalable and Reliable Dataflow Infrastructure

The original StreamCI architecture relied on RabbitMQ [2] as its message broker. As the system evolved to support blob ingestion, multi-stage processing pipelines, and a growing number of data

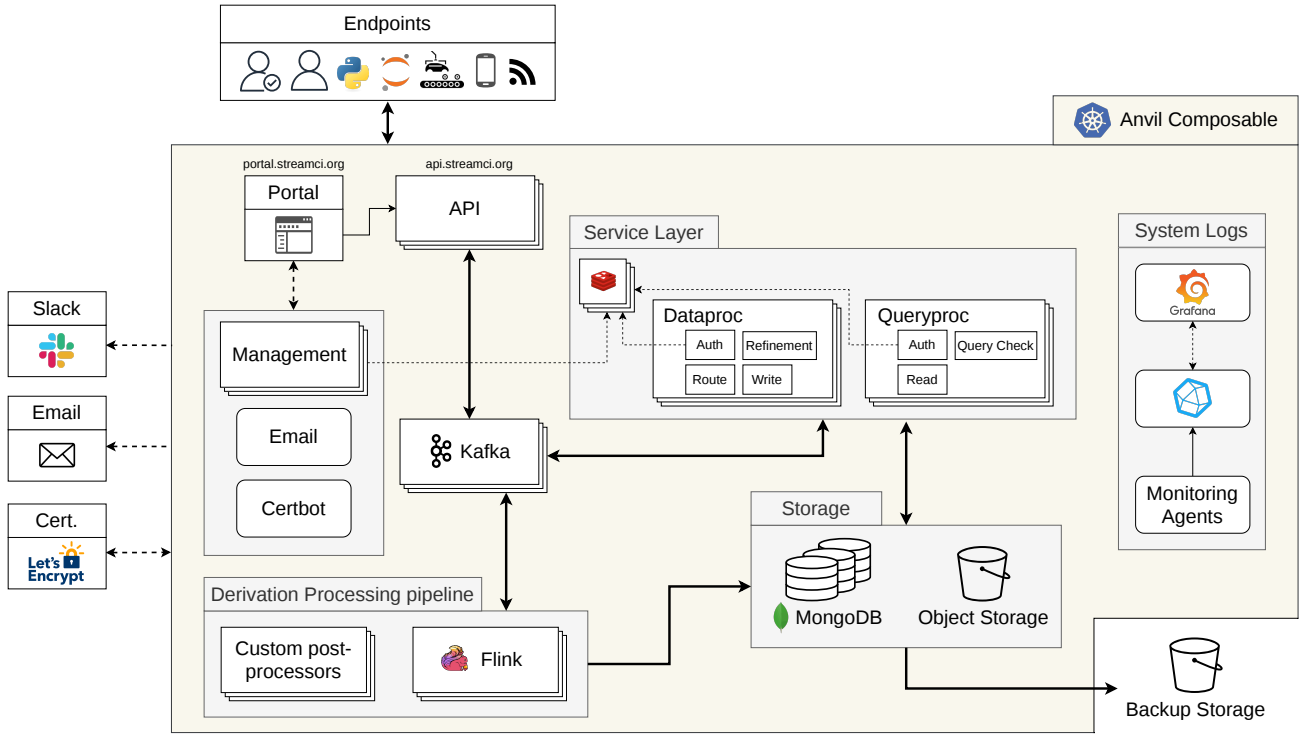

Figure 1: Updated StreamCI architecture. Data enters through the Kafka-based messaging backbone and is processed by the refinement engine in the dataproc container before being stored to MongoDB. Stored data can then be further processed by derivation processing pipelines. Automated backups protect stored data. Users interact via the portal or PyStreamCI.

sources and users, it required a more scalable and extensible messaging backbone. To address these needs, we migrated all messaging flows to Apache Kafka [1].

Kafka's log-based, persistent messaging model offers key advantages for scientific data workflows: its decoupled topic-consumer architecture allows new processing stages to be added without impacting existing pipelines; built-in message persistence enables replay and reliable reprocessing; and partitioned topics, keyed by data source identifier, provide horizontal scalability as connected data sources grow. Together, these features enable StreamCI to scale efficiently while allowing new workflow stages to be introduced with minimal disruption.

3.3 Data Reliability and Protection

We implemented automated backup with two complementary strategies. Incremental backups run every 6 hours, capturing recent database changes and limiting potential data loss to at most 6 hours. Weekly full backups create complete database and blob storage snapshots for catastrophic recovery. Both operate automatically and are monitored through the system's alerting infrastructure. This backup architecture is distinct from Kafka's dataflow reliability: Kafka ensures reliable message delivery during processing, while backups protect already-stored data.

3.4 Accessible Interfaces: Portal and Python API

User portal. The StreamCI portal[1] was redesigned as the primary interface for domain researchers who need to work with streaming data without requiring backend expertise. Through the portal, researchers can independently register data sources, define schemas, configure refinement and derived data pipelines, and browse and visualize stored data. They can also manage access permissions without requiring direct assistance from the engineering team. The portal implements role-based access control that supports both human users and machine credentials (e.g., IoT device keys), with configurable visibility scopes for data sharing. Figure 2 shows the portal's data source registration workflow and derived data generation pipeline configuration canvas.

PyStreamCI. For programmatic access, we developed PyStreamCI [4], an official Python client library that provides a connector-style API for StreamCI, similar in concept to database drivers. It enables sensor devices and automated scripts to ingest data into StreamCI and retrieve it programmatically. PyStreamCI supports authentication, batch and single document insertion (including blobs), querying with filtering and sorting, memory-efficient streaming of large result sets, and CRUD operations. It also integrates with Jupyter Notebooks for interactive exploration.

[1] https://portal.streamci.org

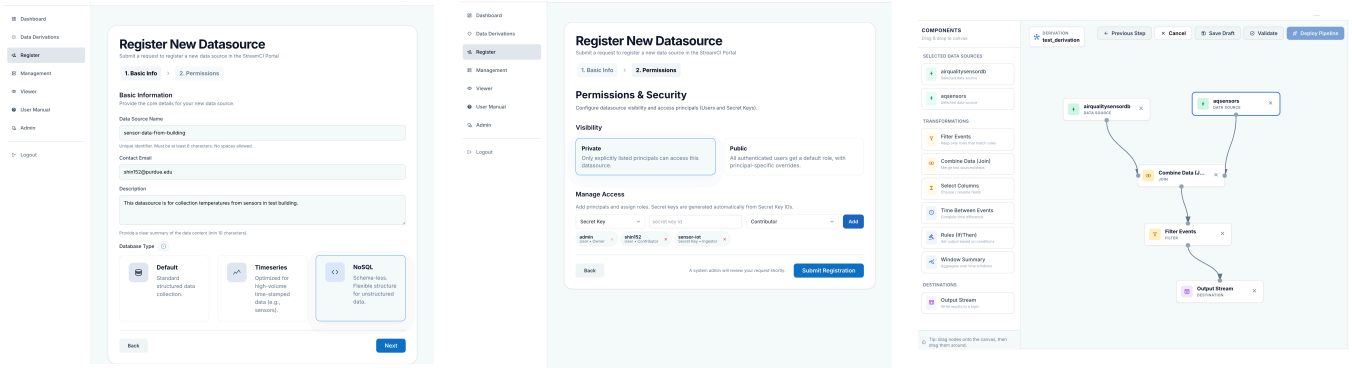

Figure 2: The StreamCI user portal. Left: Data source registration step 1 (basic info). Center: Data source registration step 2 (permissions). Right: Derived data generation pipeline configuration canvas.

4 End-to-End Workflow and Use Cases

4.1 Representative Workflow

A typical workflow proceeds as follows: (1) a researcher registers a data source through the portal, specifying the schema and access permissions; (2) raw data (structured records or blobs) is ingested via PyStreamCI, with Kafka distributing messages to processing pipelines; (3) refinement validates, cleans, and standardizes incoming records; (4) derivation pipelines produce analytical products in separate collections; (5) automated backups protect all stored data; and (6) researchers access data through the portal or PyStreamCI for integration with notebooks and downstream analysis.

4.2 Use Cases

StreamCI has been applied across a wide range of research and education use cases. We highlight two representative examples below.

Building energy sustainability. Building on the AXIN data lake work described in our prior publication [6], StreamCI continues to support the building energy research team working with IoT sensor data from the Emerging Manufacturing Collaboration Center (EMC^2) facility at Indianapolis. Over 1,000 asset-property pairs of HVAC, temperature, humidity, and energy consumption data are continuously ingested into StreamCI at approximately 2,500 records per minute. The following components are currently *operational*: data ingestion via PyStreamCI, and derived data generation jobs that perform 1-minute sampling and 5-minute rolling average computations on selected sensor fields, producing derived time series that serve as inputs for energy prediction models. It significantly reduces the manual data wrangling workload that previously required researchers to download, preprocess, and aggregate sensor data using ad hoc scripts. The next phase, currently under development, introduces a custom post-processing pipeline for automated outlier detection on the aggregated data and triggers alerts to facility operators when anomalous consumption patterns are detected.

Pavement condition monitoring. In this use case, we developed a web portal, PaveHub, to support an end-to-end workflow for processing vehicle-mounted imagery and GPS data for pavement condition assessment. Using PyStreamCI, raw imagery data is ingested as blobs alongside road condition metrics with their coordinates and geospatial metadata. The dataset currently spans two cities, Carmel and Indianapolis, with approximately 100 GB and 300 GB of imagery respectively, totaling approximately 400 GB of blob storage managed by StreamCI.

Beyond these examples, StreamCI supports additional domains including ecology education, precision audiology, biodiversity monitoring, crop health, and manufacturing digital twins in various stages of integration. The breadth of these use cases validates StreamCI's design as a general-purpose scientific data workflow platform serving heterogeneous data types and diverse processing requirements across disciplines.

5 Initial Evaluation and Operational Experience

The extensions have been deployed on Purdue's Anvil cloud infrastructure [7] and are serving active research use cases.

Reduced manual data wrangling. The building energy use case (Section 4.2) demonstrates concrete workflow improvement. The refinement and derived data generation pipelines replaced manual download-preprocess-aggregate scripting workflows, now running automatically within the platform for the EMC^2 facility's 1,000+ sensor streams.

The PaveHub use case validates blob support at scale. The stored imagery and metadata are served to a PaveHub web application that enables researchers to interactively explore pavement conditions on a map interface with filtering and query capabilities, overlaying condition metrics and viewing imagery for selected road segments.

Extensible dataflow architecture. The production Kafka deployment runs a 3-node cluster with replicated, partitioned topics and 30-day retention. The derived data generation pipeline was added as a new set of Kafka consumers without modifying the existing ingestion path, confirming that the decoupled topic model supports independently addable processing stages, a property that was less cleanly supported under the previous message queue architecture.

Data protection. The automated backup system produces 6-hourly incremental and weekly full backups of both the database and blob storage without manual intervention. A weekly full backup of approximately 70 GB (database) completed in 74 minutes and 128 GB (blob storage) in 6 minutes. Restoration tests confirmed practical recovery times: 62 minutes for the database and 3.5 minutes for blob storage.

Ease of Integration. PyStreamCI has been adopted for integrating sensor devices and automated data collection scripts with StreamCI, providing a connector-style API that allows existing data collection environments to connect by adding a single library dependency. The redesigned portal has been deployed and is being introduced to use case teams for self-service data source registration, pipeline configuration, and access management.

Availability. PyStreamCI is publicly available on PyPI [4], and the redesigned portal is publicly accessible online (Section 3.4). The platform currently runs as a managed deployment on Anvil; a lightweight version that can be self-hosted on a local Kubernetes cluster is planned, with deployment at other institutions as a future direction.

6 Conclusion and Future Work

We presented a set of extensions that transform StreamCI from a stream-centric ingestion platform into a comprehensive, end-to-end scientific data workflow platform. The addition of blob data support, raw data refinement, and derived data generation enables management of the full data lifecycle from raw artifacts to analysis-ready products. The migration to Apache Kafka provides a scalable and extensible dataflow backbone. Automated backups strengthen data protection and operational trustworthiness. The redesigned user portal and PyStreamCI library lower adoption barriers for a diverse research community. These capabilities are deployed on Purdue's Anvil cloud infrastructure and support active projects in building energy sustainability, pavement condition monitoring, precision audiology, and other domains.

Looking ahead, the extensible architecture established by these contributions provides a strong foundation for integrating ML/AI capabilities, building on the Flink-based processing and ML/AI modeling workflows prototyped in the AXIN data lake [6]. Our primary direction is the automated generation of AI-ready datasets—using the refinement and derivation pipelines to produce cleaned, structured, analysis- and model-ready data products—coupled with in-platform model training and serving. We are also pursuing an LLM-assisted portal interface that lets researchers configure data sources and processing pipelines from natural-language descriptions, and automated anomaly detection over derived data streams, building on the building energy use case. By combining broader data support, scalable infrastructure, data protection, and accessible interfaces, StreamCI moves toward a trustworthy and researcher-centered platform for end-to-end scientific data workflows.

Acknowledgments

This work was supported in part by the National Science Foundation under award OAC-2513947.

References

[1] Apache Software Foundation. 2024. Apache Kafka. https://kafka.apache.org/. Accessed: 2026-03-25.

[2] Broadcom Inc. 2024. RabbitMQ: Open Source Message Broker. https://www.rabbitmq.com/. Accessed: 2026-03-25.

[3] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Ankur Toshniwal. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015), 28–38.

[4] Jaewoo Shin, Mehak Jain, and Jorge Iván Fuentes Rosado. 2026. PyStreamCI: Python Client Library for StreamCI. https://pypi.org/project/pystreamci/. Version 0.4.1.

[5] Jaewoo Shin, Lan Zhao, Carol X. Song, Rajesh Kalyanam, Jian Jin, Jacob D. Hosen, Ananth Grama, and Dongyan Xu. 2022. Enabling Scalable and Reliable Real Time Data Services for Sensors and Devices in StreamCI. Gateways 2022, San Diego, CA. doi:10.5281/zenodo.7186972

[6] Jaewoo Shin, Lan Zhao, Carol X. Song, Dikai Xu, Ming Qu, Ananth Grama, and Dongyan Xu. 2024. Integrating ML/AI Workflows in a Streaming Data Management and Processing Platform for Building Energy Research. In *Practice and Experience in Advanced Research Computing (PEARC '24)*. ACM, Providence, RI, USA, 1–5. doi:10.1145/3626203.3670599

[7] Carol Song, Preston Smith, Rajesh Kalyanam, Xiao Zhu, Eric Adams, Kevin Colby, Patrick Finnegan, Erik Gough, Elizabett Hillery, Rick Irvine, Amiya Maji, and Jason St John. 2022. Anvil – System Architecture and Experiences from Deployment and Early User Operations. In *Practice and Experience in Advanced Research Computing (PEARC '22)*. ACM, Boston, MA, 1–9. doi:10.1145/3491418.3530766