September 2022

Data streaming transforms the post-trade

Building a horizontally scalable trade capture system

Clear Street®

Abstract

Over the past 20+ years, intraday changes in the financial markets have often been treated as static, end-of-day events. As observed during market volatility over the last few years, throughout the pandemic, and with meme stocks, middle- and back-office systems can be choke points in post-trade processing. These blockages materialize as sustained outages when market events in the front-office are not processed fast enough, leaving customers exposed to financial risk as market volatility unfolds throughout the day.

The solution? A more agile development of processing power and access to data from current markets in the post-trade. Clear Street, a modern fintech, is leveraging Kafka to build a clearing system that replaces legacy processes. A trade capture platform built on data streaming allows Clear Street to process multiple data feeds from different sources in real time, boosting speed and accuracy, increasing throughput, and reducing cost.

This whitepaper details how Clear Street uses data streaming to build and maintain a scalable trade capture system for our growing business.

Identifying the Challenge and the Solution

The Challenge

The inherent complexity of the post-trade process makes it both a legacy stronghold and a goldmine of potential fixes. Despite fundamental changes to the frontend of the capital markets, the middle- and back-office of the trade has remained largely unchanged over the past few decades. In fact, the industry spends an estimated $24 billion on trade processing, using mostly antiquated technology, like COBOL and mainframes.

Capital markets rely on multiple data sources, and mismanagement of this data comes at a significant cost, resulting in:

- Misaligned systems of record
- Duplicate data or incorrect schemas, causing outages in downstream applications
- Mismatched latency and data structure
- Lost information and integration issues
- Compliance risks and fines

The Solution

Clear Street is a fintech building modern infrastructure for capital markets, reimagining the legacy workflows and silos that are essential to increasing access in the financial markets, all while transparently decreasing risks and costs.

At Clear Street, processing speed is key to our success. Our platform is built on a modern foundation designed around Kafka and tightly integrated into the AWS cloud, giving us primitive horizontal and vertical scalability.

Trade capture is at the very core of Clear Street's business. It's what hydrates our ledger and manages the lifecycle of trades that go through our system. It's the obvious choke point for all data, and for this reason it needs to be scalable, resilient, and extensible.

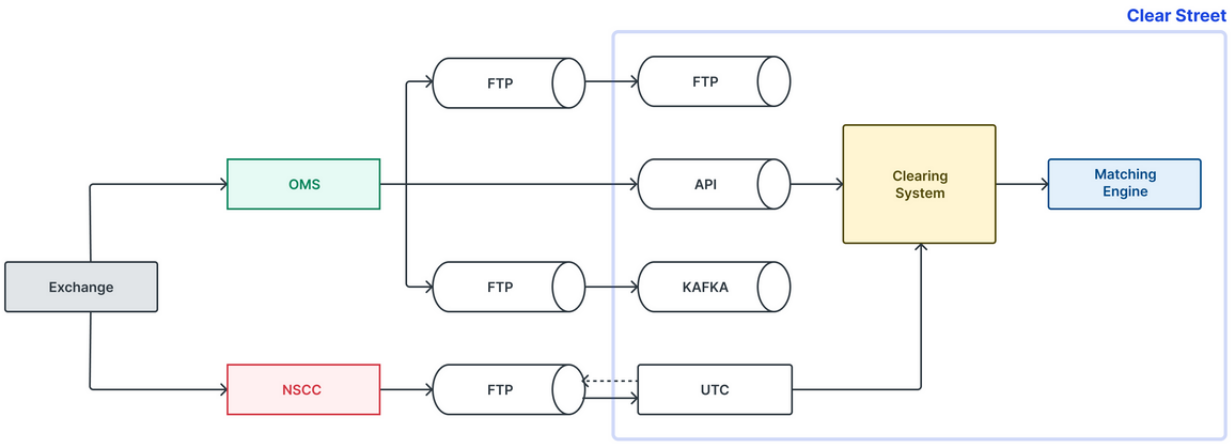
Clear Street's architecture involves a core trade capture system from which data flows downstream, like money being moved or trades coming into settlement.

At the end of the day, clearing and settlement are synchronous events with constant change. Our observable system allows us to create real-time views, and our architecture lets us easily decompose complex finance business logic into manageable chunks, or microservices, and Kafka's democratization of data access and data flows make it easy for us to create an event-sourced architecture.

Lifecycle of a Trade

An executed trade begins with an order being entered into an Order Management System (OMS) and routed to one or more venues such as an exchange (e.g. New York Stock Exchange or Nasdaq) for execution. When the order is executed, the venue sends an execution message back to the OMS. After that, the OMS routes the trade to a Clearing System (e.g., Clear Street) that is responsible for the clearing and settlement process.

Simultaneously, trades from all market centers are routed to the National Securities Clearing Corporation (NSCC), which in turn sends instructions on the trade to the Clearing System over a Financial Information eXchange (FIX) connection through Up The Coax (UTC) protocol. Then, a reconciliation process matches house trades (from OMS) and street trades (from the NSCC). After 2 days, the trade is settled (i.e., shares between the two parties involved in the trade are exchanged) through a process called continuous net settlement (CNS).

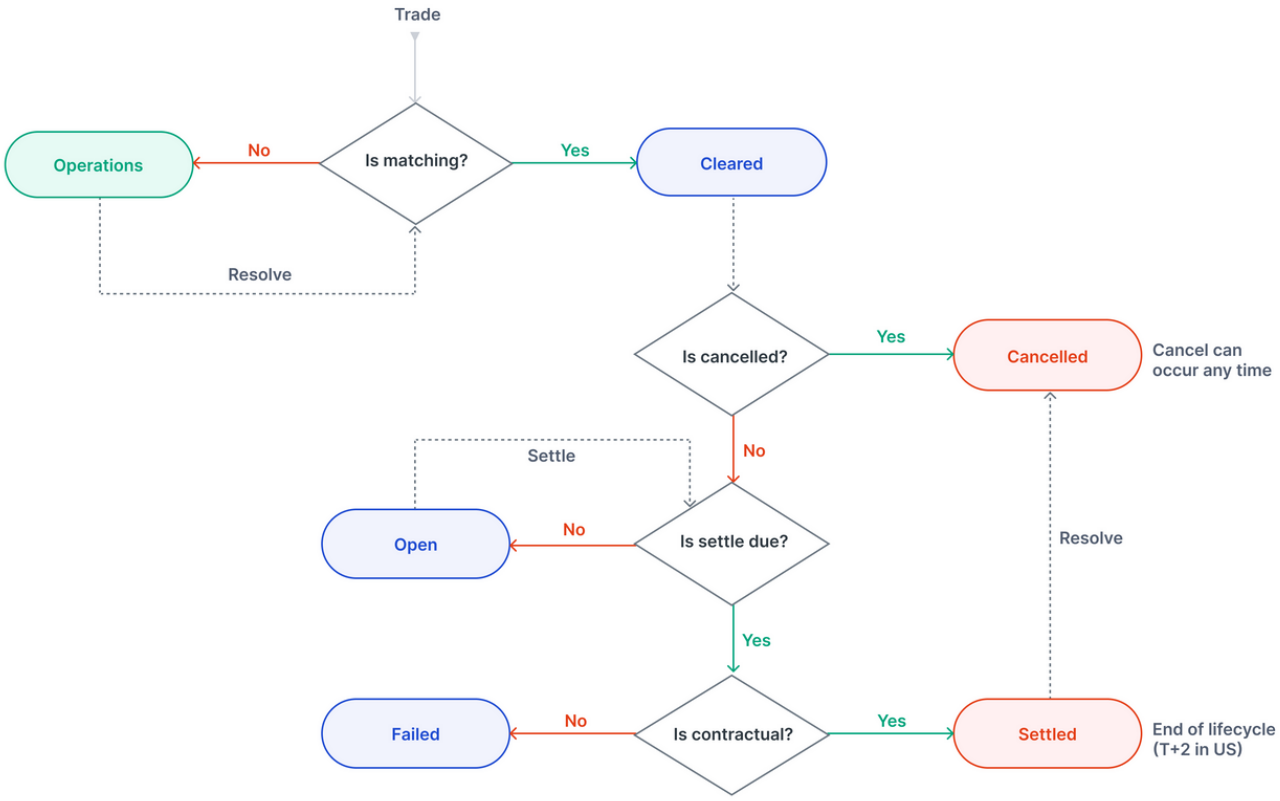


As shown above, there are 3 main ingress points for trades incoming from the Order Management System (OMS) into Clear Street's clearing system:

1. HTTP requests to our client-facing API endpoints
2. CSV files ingested via FTP, and uploaded to AWS S3
3. Trade ingestion via FIX, transformed into Kafka events

Once a trade is matched and cleared, it transitions through several stages during its lifetime as it progresses toward being settled. The primary states of interest are:

1. **Cleared:** On creation, a trade's details are verified to ensure they match with our records.
2. **Settled:** A trade enters a settlement period, which gives the buyer and seller time to do what's necessary to fulfill their part of the trade. This is typically 2 business days (i.e., T+2) after the trade date.
3. **Failed:** Trades that cannot be settled through CNS qualify for fail processing. That workflow undergoes additional stages before getting settled.
4. **Canceled:** A trade can be canceled at any point within its lifecycle, even after clearing or settlement. The canceled state is terminal.

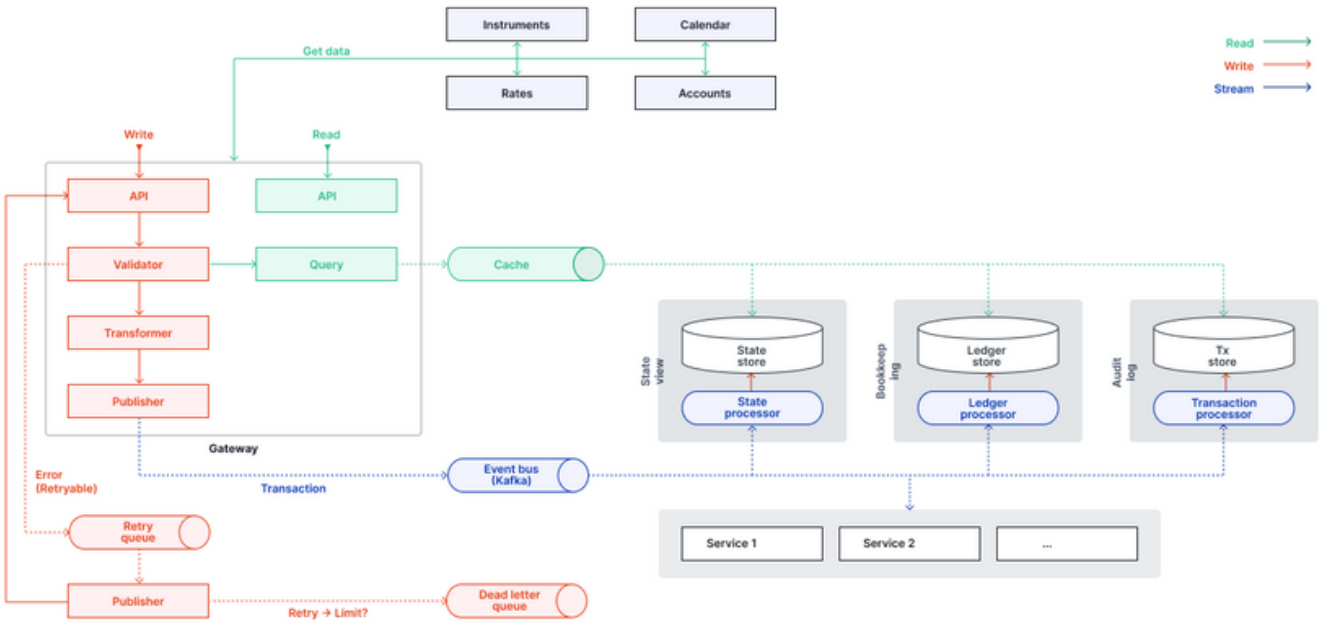


Building Clear Street's Trade Capture System

Architecture

Our system is designed to accommodate high-throughput traffic and enable horizontal scale, without compromising consistency or behavioral integrity. The system can be latency-tolerant, with sub-second processing time. It relies on event sourcing via Kafka for a single source of truth for all transactions. Availability should be fairly high to ensure no prolonged downtime for our system, however we do want to gracefully handle extended periods of failures without interrupting the incoming client trade flow.

The below figure depicts a simplified high-level design of Clear Street's system. We'll expand on the main components in what follows.



Gateway

The Gateway component serves as a common API Gateway for all trade requests (read and write paths). Unifying the API interface simplifies client integration and effectively abstracts the remainder of the system. Upon receiving a trade request, all necessary transformation and validation logic is performed upfront in the Gateway. This ensures, to the extent possible, that we maintain the integrity of data propagated downstream. To improve isolation, we have a different gateway role for each input type (e.g., file, stream, API), which can be scaled independently.

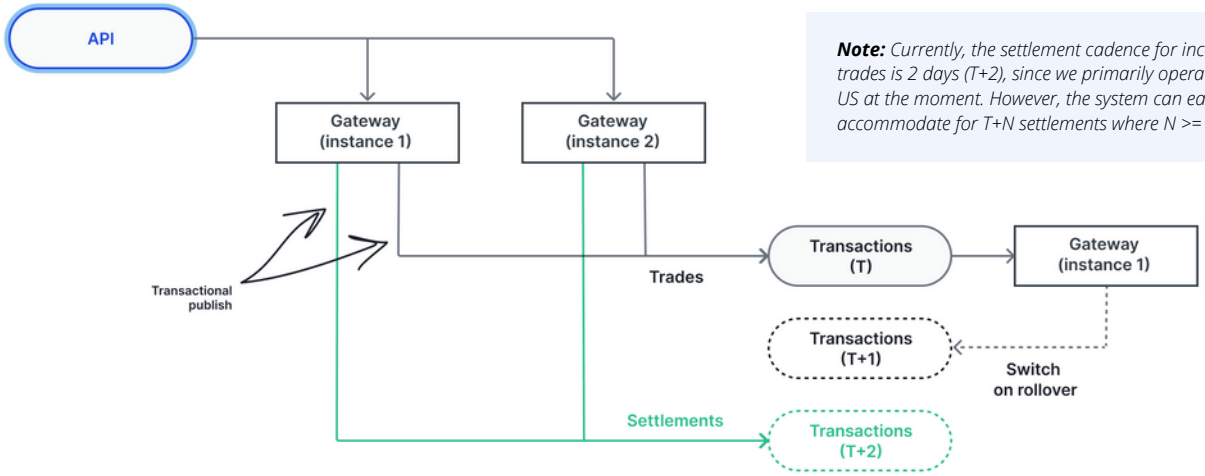
Event Source

After a trade request is processed, it is transformed to a transaction (i.e., trade action) and is published to our event bus, Kafka. The design aims to leverage event sourcing, specifically by relying on an event stream that acts as a source of truth for all transactions committed to our system. Moving directly to Kafka and bypassing the database on the critical path relieves request serving from being storage-bound, and considerably improves performance, throughput, and scalability considerations.

Dated Topics

Every transaction is associated with an effective date, which is the date it is executed. An effective date can be set for a future date. For example, a settlement transaction is created on trade creation, which is scheduled 2 days from when the trade is made. Events are received within a set time window during the day, after which there is a cutoff to the next day. The process of transitioning to the next trading date is called “rollover.”

To address the above requirements, we created a Kafka topic per date. This helps achieve a clean partition between trading activity within a day, and allows us to defer execution of transactions by inserting a future-dated topic. For requests resulting in transactions across multiple days, we leverage Kafka’s transactional producer to ensure atomicity of the operation. As a result, the downstream consumer switches to consume from the new date topics on rollover.



Note: *Currently, the settlement cadence for incoming trades is 2 days (T+2), since we primarily operate in the US at the moment. However, the system can easily accommodate for T+N settlements where N >= 0*

Processors

On the receiving end, Clear Street maintains 3 main materialized views of the transaction stream:

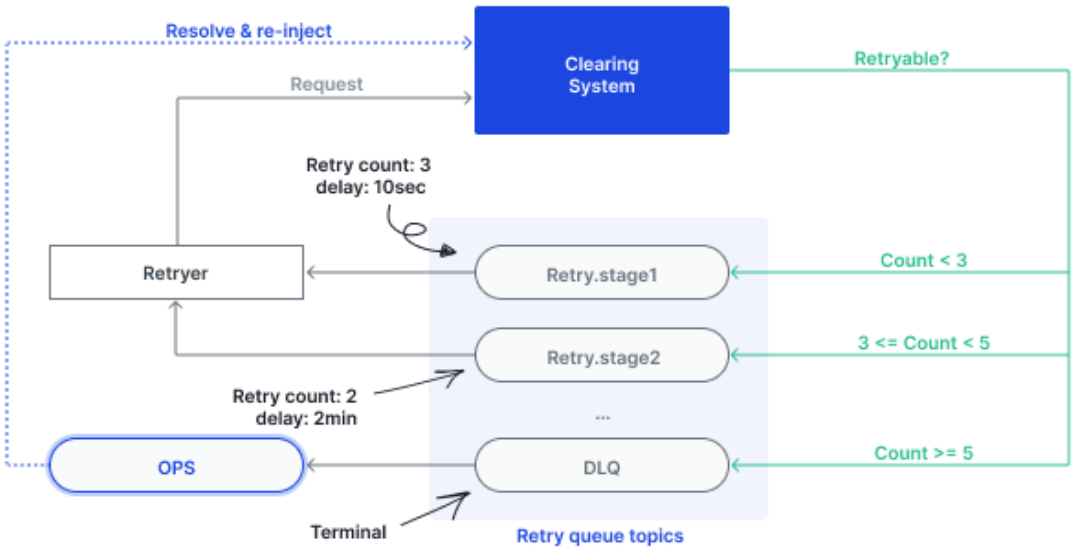
- **State View:** Tracks the state of all trades as presented in the trade lifecycle
- **Audit Log:** Stores an audit trail of transactions for historical purposes
- **Ledgers:** Maintains a bookkeeping view of client accounts, used for financial reporting

We segregate the processors for the different views to enable parallelization and promote separation of concerns (SoC). This gives us flexibility to optimize query patterns for each view, for instance by adopting different storage solutions (e.g., PostgreSQL for ledgers, DynamoDB for audit trail, MongoDB for Operations Portal). Segregation also lends itself to the Command Query Responsibility Segregation (CQRS) pattern, where read-and-write data models can naturally diverge. Finally, isolation alleviates a single point-of-failure (SPoF) in the system, and allows for independent scaling.

A tradeoff of this approach is tolerance to eventual consistency. We can incur a lag in updating views as a consequence of consuming events off of a stream. Additionally, given each workflow is processed independently, there is no guarantee at a given time that all views are in sync. In our case, a small lag is tolerable, because the the processors are mostly independent.

Retryer

On transient failures, we require a reliable method to retry operations without blocking execution or failing hard, because of factors including race conditions, database glitches, or connectivity issues that can naturally be resolved by inducing processing delay. The majority of failures in this category can be resolved by adding more resiliency to the app layer (e.g., request retry logic). However, some resolvable failures can be more prolonged (e.g., not being able to resolve a trade's stock symbol). To handle these situations gracefully, we introduced the following staged retry mechanism:

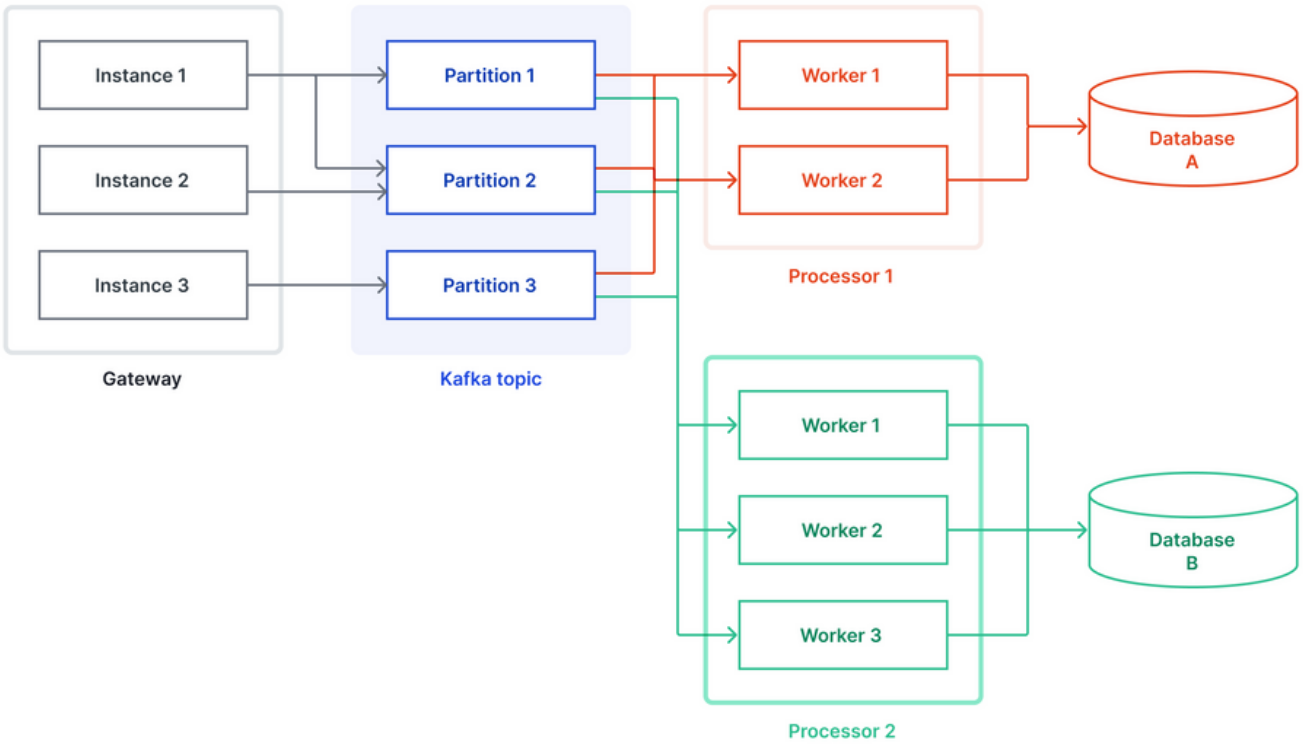


Retry Queue: Retryable failed transactions are added to a retry queue and scheduled for processing after an induced delay. On retry, the transaction is sent through to the gateway to be re-processed. Failures are repeatedly inserted into the retry queue up to a set retry count limit. We can also have several retry layers with varying settings as needed. We leverage Simple Queue Service (SQS) for retry queues, given its FIFO semantics, exactly-once guarantee, and delayed execution features. We chose SQS over Kafka here because it lets us remove individual entries once they are either processed or determined to be failed requests and routed to the Dead-Letter Queue.

Dead-Letter Queue (DLQ): Failed requests that exceed the retry limit, or require manual intervention, are added to a DLQ. Clear Street's operations team is consequently notified to try to resolve issues or notify the client of the failure. These cases are very rare, given most error categories are remediable.

Results

As we grow, Clear Street expects to support 100x our initial trade volume and beyond. This design can scale to accommodate that growth, and seamlessly adapt to traffic growth by adding more compute and/or resizing our Kafka cluster (i.e., horizontal scaling). The main unit of scale that reflects how much throughput the system can accommodate is the number of partitions per dated topics. Additionally, segregation of data persistence for each processor allows us to scale our database layer independently, or consider different storage solutions for each workflow as suitable.



The Future of Data Streaming at Clear Street

Clear Street's growth depends on our ability to build products fast, and iterate and innovate on small pieces.

This paper did not address that Clear Street's processing depends on marker events to indicate when a stream partition is done, for example to mark the End-of-Day (EOD). We have found that applying a consistent pattern to process these multi-partition event synchronizations correctly is challenging to replicate with each downstream service and corresponding development teams. We are exploring a more general solution for this pattern.

We are fundamentally oriented towards building platforms internally, and then dog-fooding those platforms for a specific use-case. We started with prime brokerage. But we believe the infrastructure we're building within our platform can be instantiated for all asset classes, investor types, and geographies.

For more information and to get in touch, visit www.clearstreet.io.