Cloud-Native Event-Driven Architecture for Algorithmic Trading: A Serverless Approach

Qiang Xue
40300671

Yicheng Cai
26396283

Yikai Chen
40302669

Yifan Wu
40153584

Alexander Sutherland
40321783

ABSTRACT

We present a cloud-native, event-driven architecture for algorithmic trading built entirely on Google Cloud Platform (GCP). The system separates batch screening from real-time monitoring and uses managed, serverless services–BigQuery, Cloud Run, Pub/Sub, and Firestore–to achieve horizontal scalability, fault tolerance, and low operational overhead. Statistical screening (Hurst exponent, Variance Ratio) runs as massively parallel BigQuery queries to maintain a Firestore watchlist, while distributed Cloud Run producers and local monitors stream and evaluate price data with lightweight indicators (ADX, Bollinger Bands). The design highlights distributed systems principles: loose coupling via messaging, polyglot persistence, autoscaling stateless compute, and geo-replicated storage.

1 INTRODUCTION

Modern trading systems demand scalable data ingestion, resilient computation, and low-latency decisioning. Our goal is to design and implement a distributed pipeline that cleanly separates responsibilities: (i) batch statistical screening at cloud scale, and (ii) real-time signal monitoring with minimal operational burden. We leverage GCP's serverless ecosystem to offload infrastructure concerns while preserving distributed systems guarantees: autoscaling stateless compute (Cloud Run), asynchronous decoupling (Pub/Sub), massively parallel analytics (BigQuery), and durable, consistent operational storage (Firestore). Trading logic is intentionally kept lightweight and modular; the emphasis is on architecture, data flow, and reliability across components.

2 BACKGROUND

This project implements an algorithmic trading system entirely on Google Cloud Platform (GCP) using serverless architecture. We leverage four core GCP services: BigQuery for distributed data analytics, Cloud Run for serverless compute, Pub/Sub for message-oriented middleware, and Firestore for NoSQL document storage. The following sections detail the technical specifications and distributed systems characteristics of each service.

2.1 Serverless Computing and GCP

GCP provides infrastructure, platform, and serverless computing environments across multiple geographic regions [13]. The platform implements fundamental distributed systems principles including data replication, fault tolerance, and horizontal scalability [2].

Serverless computing represents a cloud execution model where the cloud provider dynamically manages server allocation [18]. From a distributed systems viewpoint, serverless platforms implement **auto-scaling** through dynamic resource allocation, spawning new container instances in response to incoming loads and terminating idle instances [25]. **Fault tolerance** is built-in through automatic request retry and instance replacement [26]. The event-driven nature promotes loose coupling, where services communicate asynchronously through event triggers and message queues [1].

2.2 BigQuery

BigQuery is Google's fully managed, serverless data warehouse designed for large-scale analytics [23]. It separates storage and compute resources, allowing independent scaling based on workload requirements.

The distributed architecture utilizes **Dremel**, Google's distributed query engine for interactive nested data analysis [22]. Dremel employs a tree-based serving architecture where queries decompose into smaller sub-queries distributed across thousands of nodes for parallel execution. Results aggregate back up the tree hierarchy. Storage leverages **Colossus**, Google's distributed file system providing durability through multi-datacenter replication [6]. Data is stored in columnar format optimized for analytical workloads, enabling efficient compression and minimizing I/O.

BigQuery implements **strong consistency** guarantees, ensuring reads always reflect recent writes [8]. **Query optimization** occurs through a distributed query planner analyzing patterns, estimating data sizes, and determining optimal execution strategies including join ordering and partition pruning [23]. The system dynamically allocates compute resources based on query complexity, automatically parallelizing operations across available slots.

2.3 Cloud Run

Cloud Run is a fully managed compute platform that automatically scales stateless containers [10]. It builds upon Knative, an open-source Kubernetes-based platform for serverless workloads [19].

Container orchestration is managed through Kubernetes primitives adapted for serverless execution [4]. When requests arrive, Cloud Run automatically instantiates container instances, routes traffic to healthy instances, and scales down to zero when idle. **Load balancing** distributes incoming requests across multiple container instances using Google's global load balancer [10]. The load balancer performs health checks and removes unhealthy containers, implementing fault tolerance through automatic replacement.

Request routing follows a path through multiple distributed components: global anycast IPs routing users to the nearest Google point of presence, regional load balancers distributing requests within a region, and service meshes handling instance-level routing [9]. Cloud Run provides **concurrency control**, allowing each container instance to handle multiple simultaneous requests up to a configurable limit [10].

2.4 Pub/Sub

Google Cloud Pub/Sub is a fully managed, real-time messaging service enabling asynchronous communication between independent applications [14]. It implements the publish-subscribe pattern where publishers send messages to topics without knowledge of subscribers, and vice versa—a fundamental principle in distributed systems design [7].

The architecture provides **at-least-once delivery** guarantees through message persistence and acknowledgment protocols. Subscribers must acknowledge receipt within a configurable deadline; otherwise, Pub/Sub automatically redelivers the message [14]. **Horizontal scalability** is inherent—the service automatically partitions message flow across distributed servers, handling millions of messages per second [16].

Message ordering requires special consideration. By default, messages deliver in no particular order to maximize throughput. However, Pub/Sub supports ordering keys guaranteeing messages with the same key are delivered in publication order through affinity routing [15]. **Geo-replication** capabilities allow topics to replicate messages across multiple regions, improving availability and reducing latency [14].

2.5 Firestore

Firestore is a NoSQL document database built for automatic scaling, high performance, and ease of development [11]. As a distributed database, it provides strong consistency guarantees, ACID transactions, and automatic multi-region replication.

The distributed architecture builds on Google's **Spanner** technology, which pioneered TrueTime for globally consistent transactions [6]. TrueTime leverages GPS and atomic clocks to provide globally synchronized timestamps with bounded uncertainty, enabling external consistency where transaction timestamps properly order committed transactions.

Data replication occurs across multiple zones within a region, following a consensus protocol similar to Paxos where a quorum of replicas must acknowledge writes before commitment [20]. **Query processing** is optimized for document-oriented data models with automatic indexing. **Real-time listeners** enable clients to subscribe to document snapshots and receive immediate change notifications through distributed streaming infrastructure [11]. **Transactions** support both optimistic concurrency control through versioning and pessimistic locking mechanisms [12].

3 ALGORITHM TRADING

This section summarizes the trading logic at a high level to provide context for the distributed pipeline.

3.1 Mean Reversion Strategy

Prices tend to revert toward a long-run average. Signals buy when price is below typical range and sell when above [5, 24]. We use this only to motivate data flow.

3.2 Statistical Screening

Screening selects symbols with mean-reverting characteristics:

- **Hurst Exponent (H)**: $H < 0.5$ suggests anti-persistence/mean reversion [17].
- **Variance Ratio (VR)**: $\mathrm{VR}(k) < 1$ indicates sublinear variance growth, consistent with mean reversion [21].

3.3 Technical Indicators

Real-time signaling uses simple, robust indicators:

- **ADX**: trend strength filter; low ADX favors range-bound conditions [27].
- **Bollinger Bands**: dynamic bands around moving average; lower-band touches indicate oversold conditions [3].

4 SYSTEM ARCHITECTURE

Our algorithmic trading system implements a distributed architecture on Google Cloud Platform, leveraging serverless computing for scalability, reliability, and cost efficiency. Figure 1 illustrates the system architecture showing data flow paths and component interactions.

4.1 Architecture Overview

The system operates in two primary modes: **screening mode** for identifying trading candidates, and **monitoring mode** for generating trading signals. The architecture separates batch processing from real-time stream processing, allowing independent scaling. From a distributed systems perspective, the architecture implements event-driven patterns enabling loose coupling through asynchronous message passing via Pub/Sub. Separation of concerns isolates data storage (BigQuery, Firestore), computation (Cloud Run, BigQuery UDFs), and monitoring (local client). Polyglot persistence stores different data types in specialized databases—analytical queries in BigQuery and operational data in Firestore.

4.2 Initialization and Historical Data Ingestion

The initialization component acquires two years of daily price data for all NASDAQ securities from Yahoo Finance and loads it into BigQuery using the Cloud API with batch

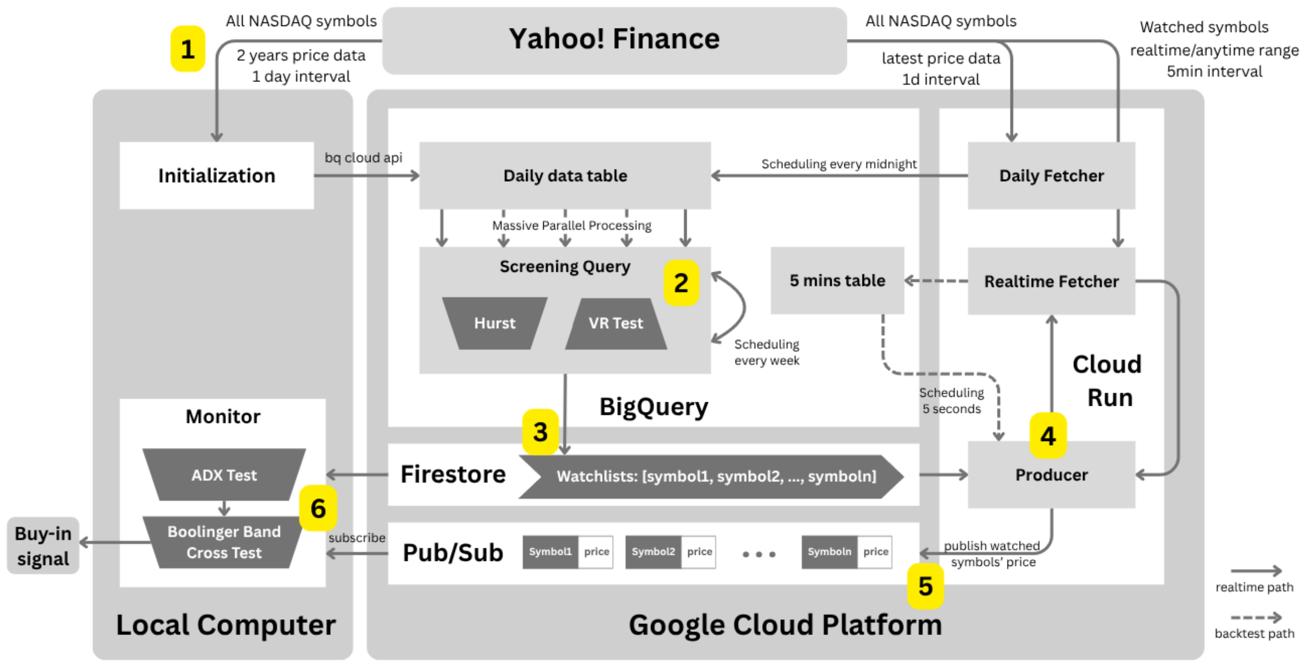

Figure 1: Distributed System Architecture on Google Cloud Platform

operations. Parallel data loading partitions symbols into batches, processing concurrently to reduce ingestion time from days to hours. Data is stored in a date-partitioned BigQuery table, allowing the system to scan only relevant partitions when filtering by date ranges. BigQuery's storage layer (Colossus) automatically replicates data across multiple nodes for durability. Separation of storage and compute allows concurrent screening jobs without resource contention.

4.3 Statistical Screening Query

The screening component identifies mean-reverting securities, executing weekly via Cloud Scheduler to maintain an updated watchlist.

User-Defined Functions (UDFs). Screening logic uses two custom SQL UDFs. The Hurst Exponent UDF computes mean reversion scores; the Variance Ratio Test UDF validates statistical characteristics. Implementing as UDFs enables massive parallelization where BigQuery distributes computation across thousands of worker nodes.

Distributed Query Execution. BigQuery's Dremel engine decomposes queries into execution tree stages. Leaf nodes scan partitions in parallel reading columnar data. Intermediate nodes aggregate price arrays using shuffle infrastructure to colocate symbol records. Root nodes apply screening UDFs and collect results. This tree-based execution achieves horizontal scalability.

4.4 Watchlist Storage in Firestore

Screening results are stored in Firestore as a collection of documents, one per selected symbol. After weekly screening, the system updates the watchlist by adding newly qualified symbols and removing those no longer meeting criteria.

4.5 Cloud Run Producer and Data Routing

The Producer orchestrates retrieval of high-frequency price data for watchlist symbols and publishes to Pub/Sub topics.

Dual-Mode Operation. The Producer operates in backtesting mode (retrieving historical 5-minute data from BigQuery) or real-time mode (fetching live data from Yahoo Finance with rate limiting).

Cloud Run Deployment. The Producer deploys as a Cloud Run service, auto-scaling instances based on request volume and scaling to zero when inactive. Each instance runs stateless, enabling Cloud Run to freely create, destroy, and migrate instances. Cloud Scheduler triggers the Producer every 5 seconds during market hours.

Publishing to Pub/Sub. The Producer publishes messages to symbol-specific Pub/Sub topics containing latest OHLCV data. This provides temporal decoupling—the Producer doesn't know about downstream consumers, and consumers process messages at their own pace. Pub/Sub buffers messages for up to 7 days.

4.6 Local Monitor and Signal Generation

The Monitor subscribes to Pub/Sub topics via pull subscriptions, receives real-time price data, computes technical indicators (ADX and Bollinger Bands), and generates buy-in signals. Running locally provides direct control and immediate notification.

Signal Generation Logic. The Monitor uses a two-filter approach: ADX below threshold (e.g., < 25) indicates range-bound markets suitable for mean reversion; price crossing below lower Bollinger Band suggests oversold conditions. When both conditions are met, a buy-in signal is generated.

4.7 Daily Fetcher Component

A scheduled Cloud Run service executes nightly to update the Daily Data Table with the latest daily prices from Yahoo Finance, ensuring current data for weekly screening.

5 CONCLUSION

This project demonstrates how managed, serverless services can be composed into a robust distributed architecture for algorithmic trading. Batch screening executes as parallel BigQuery jobs that update a Firestore watchlist, while Cloud Run producers and Pub/Sub enable elastic, event-driven real-time monitoring. The resulting system exhibits loose coupling, horizontal scalability, strong consistency for storage, and operational simplicity. Although we employ basic trading indicators, the architecture is strategy-agnostic and readily extensible: additional analytics, alternative data sources, and new consumers can be integrated without disrupting existing components. Future work includes automated deployment, end-to-end observability, and multi-region failover to further strengthen resilience.

REFERENCES

[1] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. 2017. Serverless Computing: Current Trends and Open Problems. *Research Advances in Cloud Computing* (2017), 1–20.

[2] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. 2013. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines* (2nd ed.). Morgan & Claypool Publishers.

[3] John Bollinger. 2002. *Bollinger on Bollinger Bands.* McGraw-Hill.

[4] Brendan Burns, Joe Beda, and Kelsey Hightower. 2018. *Kubernetes: Up and Running: Dive into the Future of Infrastructure* (2nd ed.). O'Reilly Media.

[5] Ernest P Chan. 2009. *Quantitative Trading: How to Build Your Own Algorithmic Trading Business.* John Wiley & Sons.

[6] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. 2013. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems (TOCS)* 31, 3 (2013), 1–22.

[7] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The Many Faces of Publish/Subscribe. *ACM Computing Surveys (CSUR)* 35, 2 (2003), 114–131.

[8] Google Cloud. 2025. BigQuery: Consistency Model. https://cloud.google.com/bigquery/docs/consistency.

[9] Google Cloud. 2025. Cloud Run: Container Runtime Contract. https://cloud.google.com/run/docs/container-contract.

[10] Google Cloud. 2025. Cloud Run Documentation. https://cloud.google.com/run/docs.

[11] Google Cloud. 2025. Firestore Documentation. https://cloud.google.com/firestore/docs.

[12] Google Cloud. 2025. Firestore: Transactions and Batched Writes. https://cloud.google.com/firestore/docs/manage-data/transactions.

[13] Google Cloud. 2025. Google Cloud Products. https://cloud.google.com/products. Accessed: 2025.

[14] Google Cloud. 2025. Pub/Sub Documentation. https://cloud.google.com/pubsub/docs.

[15] Google Cloud. 2025. Pub/Sub: Message Ordering. https://cloud.google.com/pubsub/docs/ordering.

[16] Google Cloud. 2025. Pub/Sub: Scalability and Performance. https://cloud.google.com/pubsub/docs/performance.

[17] Harold Edwin Hurst. 1951. Long-Term Storage Capacity of Reservoirs. *Transactions of the American Society of Civil Engineers* 116 (1951), 770–799.

[18] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. *arXiv preprint arXiv:1902.03383* (2019).

[19] Knative Authors. 2025. Knative Documentation. https://knative.dev/docs.

[20] Leslie Lamport. 2001. Paxos Made Simple. *ACM SIGACT News* 32, 4 (2001), 51–58.

[21] Andrew W Lo and A Craig MacKinlay. 1988. Stock Market Prices Do Not Follow Random Walks: Evidence from a Simple Specification Test. *The Review of Financial Studies* 1, 1 (1988), 41–66.

[22] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: Interactive Analysis of Web-Scale Datasets. In *Proceedings of the VLDB Endowment*, Vol. 3. 330–339.

[23] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, Hossein Ahmadi, Dan Costello, Alexander Kipnis, et al. 2020. Dremel: A Decade of Interactive SQL Analysis at Web Scale. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3461–3472.

[24] Andrew Pole. 2007. *Statistical Arbitrage: Algorithmic Trading Insights and Techniques.* John Wiley & Sons.

[25] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J Yadwadkar, Raluca Ada Popa, Joseph E Gonzalez, Ion Stoica, and David A Patterson. 2021. What Serverless Computing Is and Should Become: The Next Phase of Cloud Computing. *Commun. ACM* 64, 5 (2021), 76–84.

[26] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 133–146.

[27] J Welles Wilder. 1978. *New Concepts in Technical Trading Systems.* Trend Research.