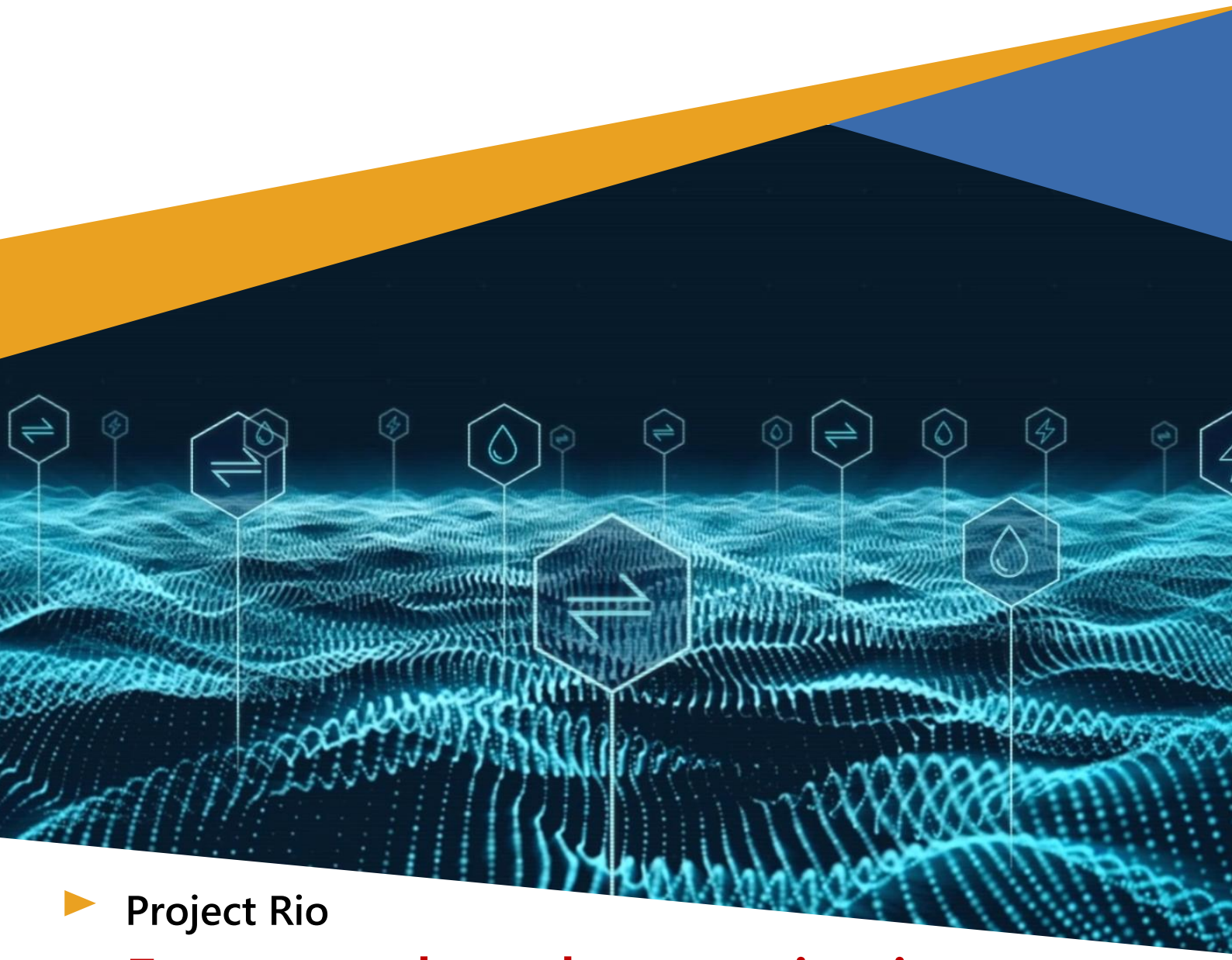


Project Rio

Fast-paced market monitoring

Technical white paper

November 2025

BIS Innovation Hub

SCHWEIZERISCHE NATIONALBANK
BANQUE NATIONALE SUISSE
BANCA NAZIONALE SVIZZERA
BANCA NAZIUNALA SVIZRA
SWISS NATIONAL BANK

Publication date: November 2025

ISBN 978-92-9259-893-8 (online)

Contents

Executive summary

Project Rio is a real-time market monitoring platform developed by the BIS Innovation Hub, in collaboration with central bank partners, to keep pace with fast-moving electronic markets, especially foreign exchange (FX), where conditions can shift in fractions of a second. Traditional batch systems, which process information in hours or days, need to be complemented for implementing monetary policy or managing reserves; Rio illustrates how streaming analytics can provide immediate insights with *latency* often below two seconds, transforming high-frequency venue data into actionable views for human decision-makers.

Built on modern, open source technology, Rio streams live *order book* and trade data from the main FX venues where price discovery happens, harmonised through a market data aggregator to simplify integration and ensure consistent timestamps. Data flow through a real-time backbone into specialised processing engines that clean and reconstruct limit order books (LOBs) and calculate liquidity, activity and risk measures in near real time. An interactive dashboard displays these insights through a unified interface, combining livestreams with historical context so analysts can see what is happening as it happens and whether it is typical for that time of day.

The architecture separates ingestion, processing, storage and visualisation to achieve speed, resilience and flexibility. A high-performance store provides the dashboard with the latest snapshots, while a time series database retains a complete history for analysis. A dedicated layer handles more complex, multi-user queries without creating bottlenecks. This modular design allows teams to update individual components independently, add new instruments or venues – demonstrated by a successful extension to Brazilian real futures – and develop analytics without large-scale system overhauls.

Rio's development followed an agile, user-centred process. A small, multidisciplinary team created prototypes with seven central banks, refining measures and visual design based on structured feedback. During a multi-month pilot involving nine central banks, users reported that Rio integrated seamlessly into workflows and received high satisfaction ratings, confirming both its value and approach. nevertheless, its components, methods and lessons have been reused across Innovation Hub initiatives and shared within the central bank community.

Rio has show how central banks can monitor algorithmic, fragmented markets in real time using widely accessible, modern technology. By combining streaming analytics, scalable storage and intuitive visualisation within a robust, adaptable architecture, Rio offers a practical blueprint for monitoring of fast-paced markets, bridging the gap between machine-speed markets and human-speed decision-making.

1 Introduction

Technology is transforming finance, enabling markets to trade at ever greater speed. Increasingly, trading is machine-to-machine. Market monitoring is a cornerstone of effective monetary policy implementation and foreign exchange (FX) reserve management. By applying technology themselves, central banks have come a long way from the days when trading and market monitoring relied on phone calls to market players.[1] Still rapid technological change drives central banks to continuously look for new tools and approaches to monitor markets.[2]

Today, market monitoring systems typically process data in batches over hours or days. While sufficient for some slower markets or longer-term analysis, this does not keep pace with fast markets, such as FX, where developments can occur in fractions of a second. Speed is not the only challenge for traditional systems. The new market structures generate vast amounts of data due to continuous trading and smaller *tick sizes*. Additionally, markets involve more participants and are increasingly fragmented, which presents challenges regarding data ingestion, storage and processing.

The BIS Innovation Hub developed Rio to help. Rio uses *stream processing*, enabling continuous data ingestion, processing and analysis as events occur. By processing data in real-time streams, central banks can significantly reduce *latency* and gain immediate insights into markets. Rio calculates a range of measures, both point-in-time and aggregated, to provide a flexible and intuitive view of the market through its dashboard. Open source components combine to build a scalable, real-time FX monitoring platform.

This paper tells the story of how Rio was built. It begins with an overview, followed by a step-through of the design and development challenges and how they were overcome, and concludes with some practical reflections. It was written for three audiences. First, for FX analysts and traders with limited prior exposure to real-time analytics, the paper explains the platform's concepts, architecture and benefits in a clear and accessible manner. Second, for engineers and software architects interested in building real-time FX monitoring platforms, the paper delves deeper into platform architecture, technology and design decisions. Finally, for innovation or technology practitioners working in central banks, the paper serves as a case study in how central bank idiosyncratic requirements can be met with flexible, modern technology.

Terminology. This paper references different software, programming languages and other technical details. For any word in *italics*, a description is included in the glossary.

[1] See M Bech, A Illes, U Lewrick and A Schrimpf, "Hanging up the phone – electronic trading in fixed income markets and its implications", *BIS Quarterly Review*, March 2016.

[2] See Markets Committee, "Monitoring of fast-paced electronic markets", *Markets Committee Papers,* no 10, September 2018.

2 Overview

- Rio monitors fast markets like FX, where developments can occur in fractions of a second. The FX market is vast and generates enormous amounts of data from thousands of market participants, trading venues and data sources.
- *Stream processing* can reliably process vast amounts of data and accommodate the variety of analytical tools necessary to monitor FX markets.
- Rio, a fast analytics tool developed by the BIS and central banks, can handle thousands of updates every second and analyse markets in less than two seconds, before presenting the results in an interactive dashboard.

2.1 The challenge of fast-paced analysis

Financial markets have undergone profound structural changes over the last two decades. Technological advances have increased the share of electronic and automated trading. Trading occurs at ever higher frequencies across multiple venues and involves new types of financial institution. As these trends make market monitoring more challenging for central banks, there is a demand for tools capable of handling live high-frequency data from multiple venues. But since industry solutions tend to target algorithmic trading and market-making use cases, they are less suited for the market monitoring needs of central banks.

When central banks monitor fast-paced markets, the most important objectives are to better understand current market liquidity and functioning conditions and to identify key market drivers (Markets Committee (2018)). This requires computing metrics such as trading activity, market liquidity and market risk. These measures are well established, but there is a technical challenge in computing them with large quantities of data and a user experience challenge in making the metrics meaningful for real-time analysis by humans.

2.2 Data streaming

Stream processing is the technology for real-time analytics. Developed since the early 1990s, it experienced significant advancements in the early 2000s with the introduction of systems like *Apache S4* and *Storm*. These systems were supplemented by open source frameworks, such as *Kafka Streams* and *Apache Flink*, which saw increasing adoption. By the early 2020s, these systems and frameworks had matured and become more widely available thanks to cloud platforms.[3]

Stream processing handles continuous data streams, processing high volumes of events instantaneously as they arrive and with the capacity to scale. *Stream processing* is flexible and allows for various real-time data operations (like filtering, aggregation and pattern detection) with reliability and resilience.

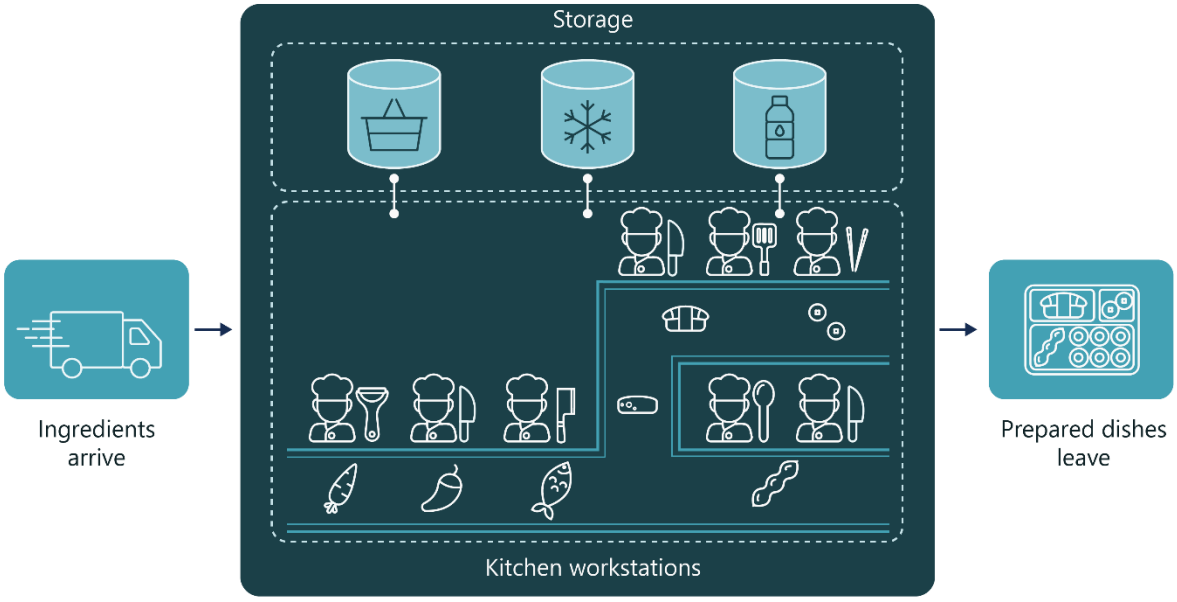
[3] See P Carbone, M Fragkoulis, V Kalavri and A Katsifodimos, *Beyond analytics: the evolution of stream processing systems*, May 2020, dcatkth.github.io/papers/SIGMOD-streams.pdf.

Unlike traditional *batch processing*, which works with static data, *stream processing* works with a continuous flow of incoming data. Data points receive immediate attention by being split into parallel tracks, allowing different processes to work side by side and further speeding up processing. Sometimes data need to be processed sequentially, and sometimes they do not.[4] *Stream processing* can handle either.

Data *stream processing* systems work like a digital assembly line, combining a "conveyor belt" event-streaming platform that moves data with multiple processing engines that clean, organise and analyse data. Processing engines operate independently, like chefs in a kitchen preparing ingredients one after another. Multiple preparation stations can operate simultaneously, and the semi-finished products of one chef can be stored and used by others to create more complex combinations. In Rio, the engines quickly arrange data for use in real-time dashboards or alerts, often completing the process in under a second (Graph 1).

This setup offers practical benefits. Since each engine is separate from the others, one part can be upgraded without shutting down the whole system. If a problem occurs, the conveyor belt keeps a data backup, making it easier to restart where things left off. Data streaming systems can also handle structured or semi-structured data (eg spreadsheets) and non-structured "messy" data (eg social media posts), allowing teams to adapt quickly to new types of information without rebuilding everything from scratch.

Data streaming as a kitchen **Graph 1**

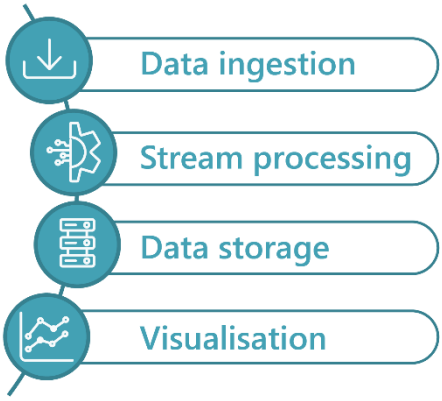


[4] Parallel processing suits **stateless**, order-insensitive stages, eg mappings like "convert units from °F to °C", in which events can be split with no coordination. Sequential processing is needed for **stateful** or **order-dependent** stages, eg for computing a running account balance, where each result relies on the preceding event, so items must flow one after another.

To turn the endless conveyor of fresh ingredients into bite-sized, analysable portions, the chefs periodically scoop everything that has rolled past in, say, the last 30 seconds or five minutes into a mixing bowl – this is windowing. Each bowl captures a short slice of time, so recipes such as running averages or top-N rankings can be completed before the next bowl arrives. Because several bowls can be on the counter at once, different chefs can season, sauté or plate them in parallel, keeping the line moving even as the volume swells. In other words, windowing carves the flow into timed mini-batches. Parallel processing allows multiple stations to work on those mini-batches simultaneously. A step-by-step illustration of these techniques appears in the Appendix.

A *stream processing* system typically consists of four components: data ingestion, *stream processing*, data storage and visualisation (Graph 2).

Stylised stream processing system components **Graph 2**

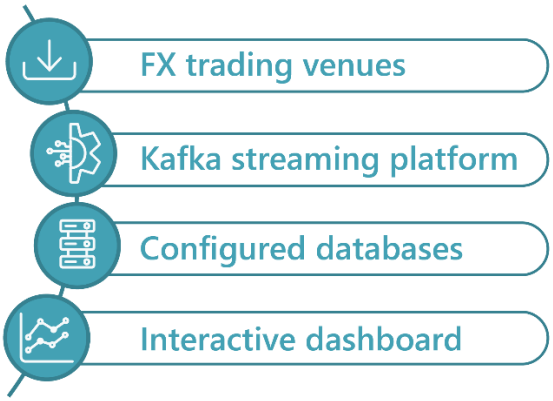


1. **Data ingestion:** Data are ingested through message queues or distributed streaming platforms. These platforms ensure reliable, fault-tolerant and scalable data ingestion from various sources. Open source examples include *Apache Kafka* and *RabbitMQ*.
2. ***Stream processing*:** *Stream processing* engines handle and transform data. These engines offer interfaces and frameworks for defining and executing real-time data processing. They can also perform essential operations for working with incomplete data, such as filtering, aggregation and windowing. Open source examples include *Kafka Streams*, *Apache Flink* or *Apache Spark*.
3. **Data storage:** Processed data are frequently stored for further analysis or reference, which is especially important for long-term analysis. Open source examples include *TimescaleDB*, a time series database, and *Delta Lake*, a data storage layer for *data lakes*.
4. **Visualisation:** Data and analysis must be visualised to enable users to gain insights from live data streams, identify anomalies and make informed decisions based on current information. Open source options include *Grafana*, which can be used to create interactive dashboards and alerts.

2.3 Architecture

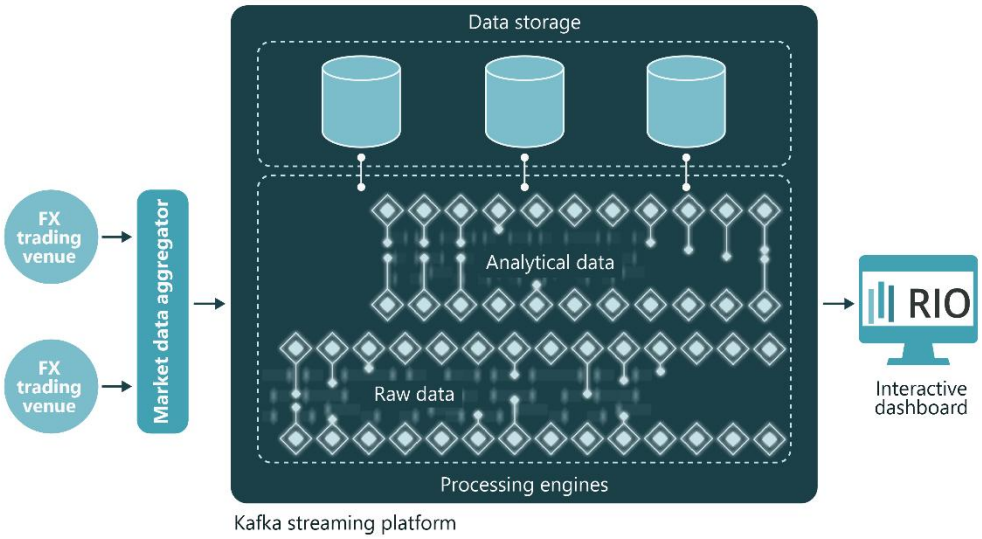
Rio is a data streaming platform (Graph 3). It ingests and stores FX trade data from various FX venues. The collected data are then transformed into a format suitable for analysis through multiple *stream processing* engines. Other engines analyse the data before a dashboard visualises them.

Rio components **Graph 3**



Rio's architecture combines the components of a *stream processing* system (Graph 4). FX data are ingested into the Kafka streaming platform through a market data aggregator. Once inside the platform, data are processed by engines that prepare and analyse them. Multiple databases are configured to ensure that data are available quickly and reliably. Finally, data are visualised in interactive dashboards.

Rio architecture **Graph 4**



Rio was deployed on *Microsoft Azure*, with separate resource groups for development and "production" environments to ensure operational clarity. An Azure-native cost management dashboard provided a comprehensive view of both real-time and historical expenditure, offering extensive insight into the platform's resource utilisation. A core design principle was *infrastructure as code* (IaC), adopted to guarantee that the platform's complex, cloud-based infrastructure could be defined, replicated and maintained consistently. Using IaC improved reliability, reduced configuration drift and enabled quick, version-controlled deployments. All components were described using *Terraform* templates stored in an *Azure DevOps* Git repository and deployed automatically through continuous integration and delivery pipelines.

The architecture focused on *software-as-a-service* (SaaS) components to lower operational overhead and enhance scalability. Azure-native services like *PostgreSQL* and *Redis* were leveraged alongside specialised solutions, including *Databricks* (for *PySpark* processing and exploratory notebooks) and *Confluent* (for *Apache Kafka* streaming). Custom business logic for data ingestion, stream processing and historical calculations was encapsulated in *Azure Container Instances* (ACI) to ensure isolation, portability and efficient resource management. Secure operation relied on *Azure Key Vault*, which managed all credential storage and rotation, while *Azure Blob Storage* acted as the primary data repository.

Apache Airflow served as the main orchestration layer, scheduling daily historical computations and dynamically adjusting the platform's power state to match active market hours, thereby optimising both performance and cost. During the pilot phase, each participating institution was allocated a dedicated *Grafana* instance for visualisation. Access to these dashboards was securely routed through an *Azure Application Gateway*, with traffic load balanced via *Nginx* to ensure resilience and a consistent user experience.

3. Data

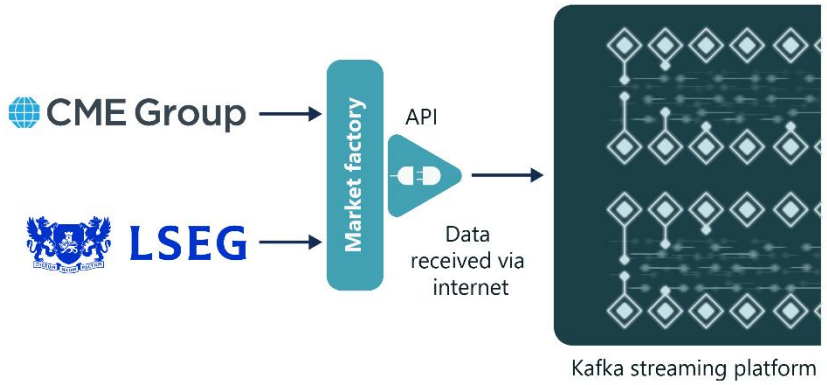
- Rio analyses markets by connecting to the two large primary venues where price discovery occurs.
- Hedge funds and proprietary trading firms use the same high-frequency data feeds.
- The use of a unified market data API simplified the integration of Brazilian real futures data (Project Samba) with metrics and visualisations created for spot markets.

3.1 Market data

FX trading activity is increasingly fragmented.[5] Additionally, an increase in "internalisation" (matching client trades) by larger banks has resulted in a decline in visible FX trading activity. Nonetheless, primary venues for FX spot remain essential for price discovery and market liquidity, especially in times of high volatility, when internal matching of client trades by dealers is less possible. As liquidity on FX venues exhibits network effects, inter-dealer trading in a currency pair tends to cluster around one primary venue.

Rio takes live high-frequency data from two primary market venues for FX spot: LSEG FX Matching (formerly Reuters Matching) and Electronic Broking Services (EBS) Market (owned by CME Group) (Graph 5). EBS and Refinitiv match clients' FX trading interests via an anonymous electronic limit order book (LOB) that operates almost 24/6 from early Monday morning, Sydney time, to Friday afternoon closing, New York time. A LOB collects and matches orders. It consists of a bid side representing the buying interest and an offer side representing the selling interest in a currency pair (Box 1).

Rio data venues and ingestion **Graph 5**



[5] See A Schrimpf and V Sushko, "FX trade execution: complex and highly fragmented", *BIS Quarterly Review*, December 2019.

Using primary venues gives central banks a "foundational" view of market liquidity. Despite the fragmentation of FX markets, these primary venues continue to be important for liquidity and price discovery. When volatility spikes or market liquidity deteriorates elsewhere, trading volume increases at *Refinitiv* and *EBS*.[6]

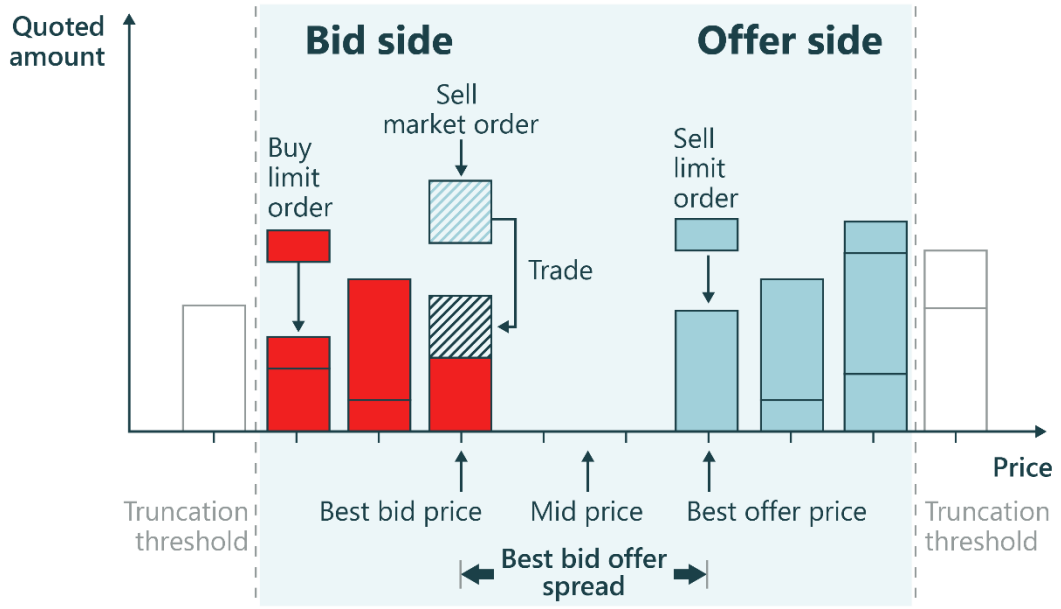
Box 1

Central limit order books

A limit order book (LOB), illustrated in Graph 1A, is a record of buy and sell orders for a market. For Rio and FX markets, LOBs are electronic systems that aggregate and match buy and sell orders for currency pairs. Every LOB has a bid (buy) side and an offer (sell) side, reflecting the buying and selling interests of its participants. Traders use two order types. First, **limit orders** specify a trade direction (buy/sell), quantity and price, and the remain active until they are cancelled, modified or matched. Second, **market orders** execute immediately at the best available price for a specified amount. Order matching follows a strict **price-time priority** protocol: superior-priced orders execute first, while orders at identical prices are prioritised by submission time. When compatible orders meet, a trade occurs. Modern trading venues stream high-frequency incremental LOB updates to maintain efficiency, providing cumulative changes to the *order book*'s state since the last update. These real-time feeds enable precise reconstruction of the LOB's status, often including concurrent trade data for comprehensive market visibility. To manage the high data volume, **a truncation threshold** is applied, retaining only the most relevant price levels or changes while discarding minor updates. This ensures efficient data processing without compromising analytical accuracy.

Illustration of the LOB

Graph 1A



[6] See A Chaboud, D Rime and V Sushko, "The foreign exchange market", in R Gürkaynak and J Wright (eds), *Research handbook on financial markets*, Edward Elgar, May 2023, pp 253–75.

3.2 Data aggregation and ingestion

Rio connects to *EBS* and *Refinitiv* using a market data aggregator. The aggregator acts as a middleman between the market venues and the data streaming platform. Hence, Rio only needs to interface with the aggregator rather than with each venue. This allows for additional market venues to be added easily (eg B3 FX futures; see Box 2).

Rio relies on the MarketFactory aggregation and connectivity service, owned by ION Group, to get*order book* updates. In addition to the single interface, MarketFactory also harmonises data fields and provides consistent timestamping. The benefits of a market data aggregator were also evident during the project's development, as both *EBS* and *Refinitiv* upgraded their platforms and APIs. This created some overhead, as the type of data Rio was ingesting was no longer available. Using a data aggregator meant that no additional development was required

Box 2

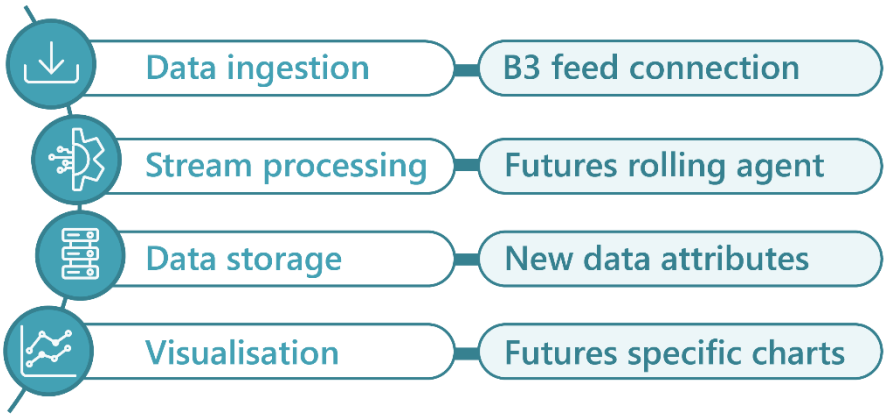
Brazilian real futures: Project Samba

Samba was a project to extend Rio (Graph 2A), to include Brazilian FX futures in collaboration with the Central Bank of Brazil. Rio connected to the B3 market data feed using MarketFactory and added features to integrate Brazilian FX futures with Rio's existing instruments.[1]

FX futures trading differs from FX spot trading in LOBs. To monitor them effectively, three significant adaptations were required. First, futures contract "rollovers" needed to be managed. Second, futures' fixed daily trading hours needed to be accommodated. Third, trading volumes needed to be converted to account for different traded contract sizes (full-size (USD 50,000) and mini (USD 10,000) USDBRL futures contracts).

The successful implementation demonstrated Rio's versatility across all components of the monitoring platform. Data about different instruments from new markets in other formats were ingested and aggregated with additional logic. Existing measures were amended to create new and meaningful monitoring measures for FX futures. Finally, charts and visualisations were incorporated into the customisable dashboard.

Samba extension of Rio components — Graph 2A



[1] B3 is a financial market infrastructure in Brazil that provides exchange and OTC trading services.

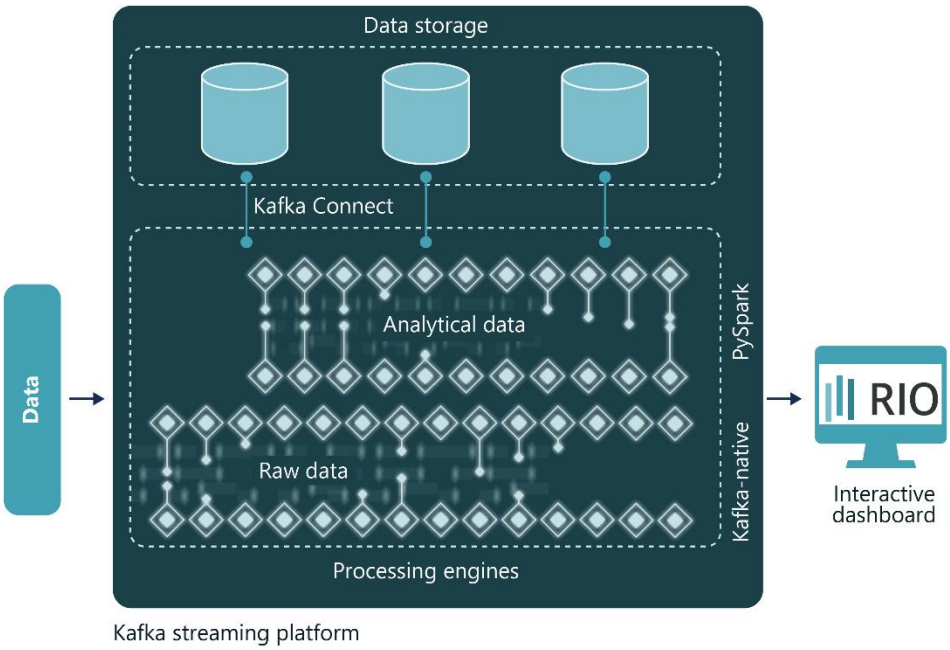
4. Stream processing

- Rio processes data fast.
- It uses Apache Kafka as a digital conveyor belt to push data through multiple processes that clean, organise and analyse them.
- Rio lets central bank analysts trial and adjust different measures and metrics (see Appendix B).

4.1 Data processing

Rio constantly receives market data, which must be processed and analysed. This is not carried out by a single programme or component but rather is distributed across multiple engines that consume the data streaming through the Kafka platform (Graph 6). Each engine is distinct but can rely on others. For example, engines clean and filter data before reconstructing the limit *order book*. Other engines then use this reconstructed limit *order book* to compute liquidity, trading activity and volatility metrics.

Rio stream processing engines **Graph 6**



The *latency* requirements for processing and analytical engines are different. Data processing and reconstruction of the limit *order book* must be fast, efficient and reliable, as engines consume them on the platform. On the other hand, analysis consumed by humans can afford to take a little longer, but it also needs to be more flexible and accessible. Therefore, different technologies were used to construct the engines.

4.2 Processing engines

Rio uses three "Kafka-native" technologies to process data to construct engines. First, *Kafka Streams* is used to develop engines that reconstruct the limit *order book*. *Kafka Streams* uses *Scala* with highly customisable logic and a wide programming library. Second, *KSQL* is used to quickly carry out basic data processes (eg filter, join, aggregate). *KSQL* uses *SQL*, which does not have the flexibility of *Kafka Streams* but can carry out simple tasks quickly. Finally, *Kafka Connect* populates the various data stores with the outputs of the different engines (outlined in the next section).

Rio uses *PySpark* to construct the engines that carry out different analyses. The "Py" in *PySpark* stands for the programming language *Python*. Rio uses *PySpark* for engines that calculate complex market measures and analyse historical data that update every minute rather than every second. Changes and additions are easy thanks to central bank analysts' existing familiarity with *Python*.

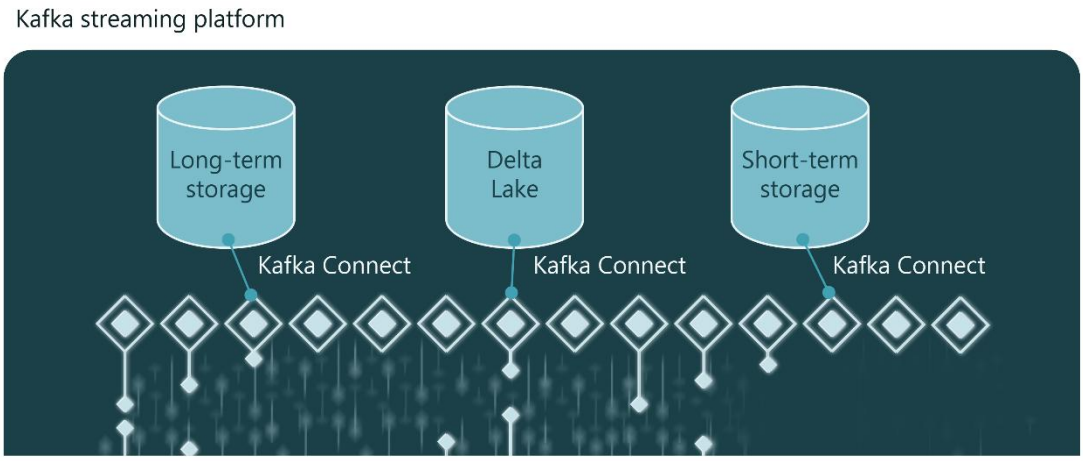
5. Data storage

- Rio's data storage is managed by *Kafka Connect* and distributed across different components to make the platform fast and scalable.
- An *in-memory data store* (*Redis*) stores the latest information for the dashboard to maximise speed. In contrast, a longer-term database (*TimescaleDB*) holds information in the cloud.
- A "storage distribution framework" (*Delta Lake*) avoids creating a bottleneck when multiple queries run simultaneously.

5.1 Data storage overview

Rio generates a large amount of data that must be stored. In 24 months, four terabytes were produced. The Rio platform serves different functions and needs three data storage components to run efficiently (Graph 7). *Kafka Connect* populates all of the components: (i) a core database (*TimescaleDB*); (ii) a faster *in-memory data store* (*Redis*); and (iii) a framework that structures and distributes data (*Delta Lake*) to manage simultaneous queries.

Rio data storage **Graph 7**



5.2 Short-term data storage

Rio's dashboard must be fast and responsive, using an *in-memory data store* (*Redis*). Within *Redis,* a specialised *RedisTimeSeries* module caches the most frequently accessed data (like the top levels of bids and offers in the *order book*) so that dashboard updates are always real-time. Additionally, *Redis* has a channel data structure designed for real-time data streams, allowing the dashboard to subscribe to specific channels to receive real-time messages from *Kafka Connect* continuously and as they arrive. In Rio's *Redis*, calculations that would otherwise need to be carried out by the

dashboard (eg calculating moving averages or summing values over a sliding window) are also performed. This makes the Rio dashboard fast and responsive.

5.3 Long-term data storage

Rio stores all ingested data and computed metrics to allow for long-term analysis. It uses *TimescaleDB*, as it is optimised for quickly ingesting time series data and allows complex queries. Additionally, as *TimescaleDB* uses *PostgreSQL*, it integrates with a wide range of *SQL*-based analytics tools, both visual (eg *Grafana*) and computational (eg *Python*, *R*, *BI platforms*).

5.4 Data distribution

Rio allows multiple users to run complex queries over years of granular data, without causing bottlenecks in the underlying databases. To achieve this, Rio includes *Delta Lake*. *Delta Lake* supplements the two databases by providing an alternative route for longer, more complex historical queries and data downloads. Each query is processed efficiently, allowing Rio to be used for both real-time analytics and more in-depth enquiries.

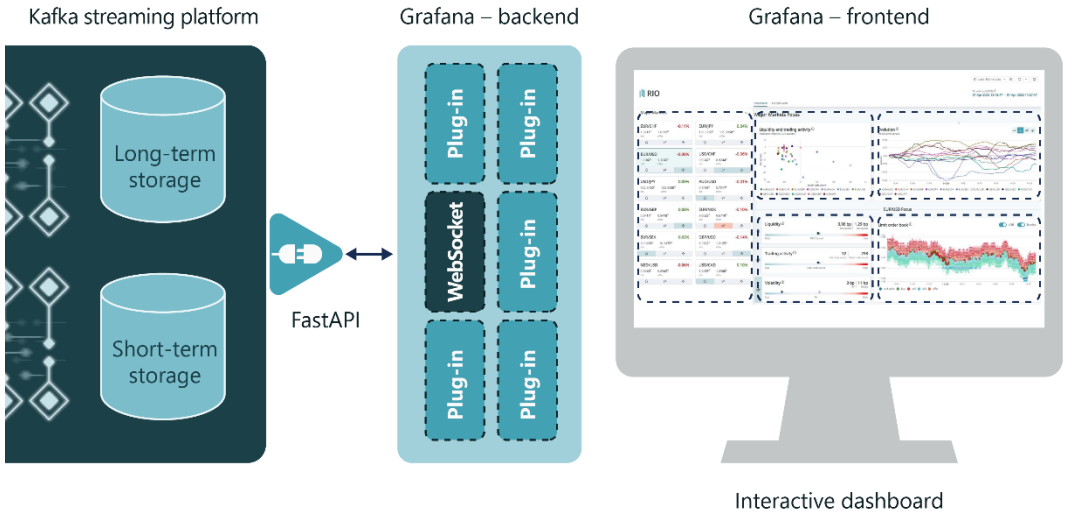
6. Visualisation

- Rio provides an interactive dashboard for visualising data, which allows central banks to understand immediately what is happening in markets.
- Creating a dashboard that could effectively display data in real time was a technical and design challenge.
- Not only does the streaming platform have to connect reliably to the visualisation software, the software must also be flexible enough to provide users with the necessary information.

6.1 Interactive dashboard

Rio helps central banks analyse complex data. Meaningful analysis must be accessible and useful to humans. Rio's dashboard helps central bankers put real-time data and metrics into an understandable context and see what happens in FX markets. An API connects the streaming platform to the dashboard (Graph 8).

Rio visualisation **Graph 8**



The Rio dashboard was built using *Grafana*.[7] *Grafana* is an open source software with two significant advantages for visualising data:

- It has a two-tier architecture, with a front-end for querying and visualising data in dashboards (based on *TypeScript*) and a back-end that sources the data and prepares them for visualisations (based on *Go*).
- Both the front- and back-end benefit from a large open source community that has developed libraries of "plug-ins" – visualisations and applications that are easy to add and change.

Although the development team initially used *Grafana* as a stopgap measure until a more sophisticated dashboard could be developed, the software proved flexible enough to meet the project's needs. This flexibility enabled fast prototyping, allowing the team to try out different ideas with central banks and quickly respond to their feedback.

Rio used many of the standard *Grafana* plug-ins and carried out some additional development. The front-end had a plug-in for each chart, gauge and layout. *Apache ECharts*, a visualisation library, was also integrated to provide even more dynamic and interactive graph options. On the back-end, Rio enhanced a community-developed *WebSocket* plug-in. The Rio *WebSocket* is a continuous, two-way connection between the dashboard and the Rio API. This ensures that the dashboard display updates as soon as new data are received (compared with a "normal request", in which a request for information is sent periodically or in response to a prompt).

6.2 Serving multiple users

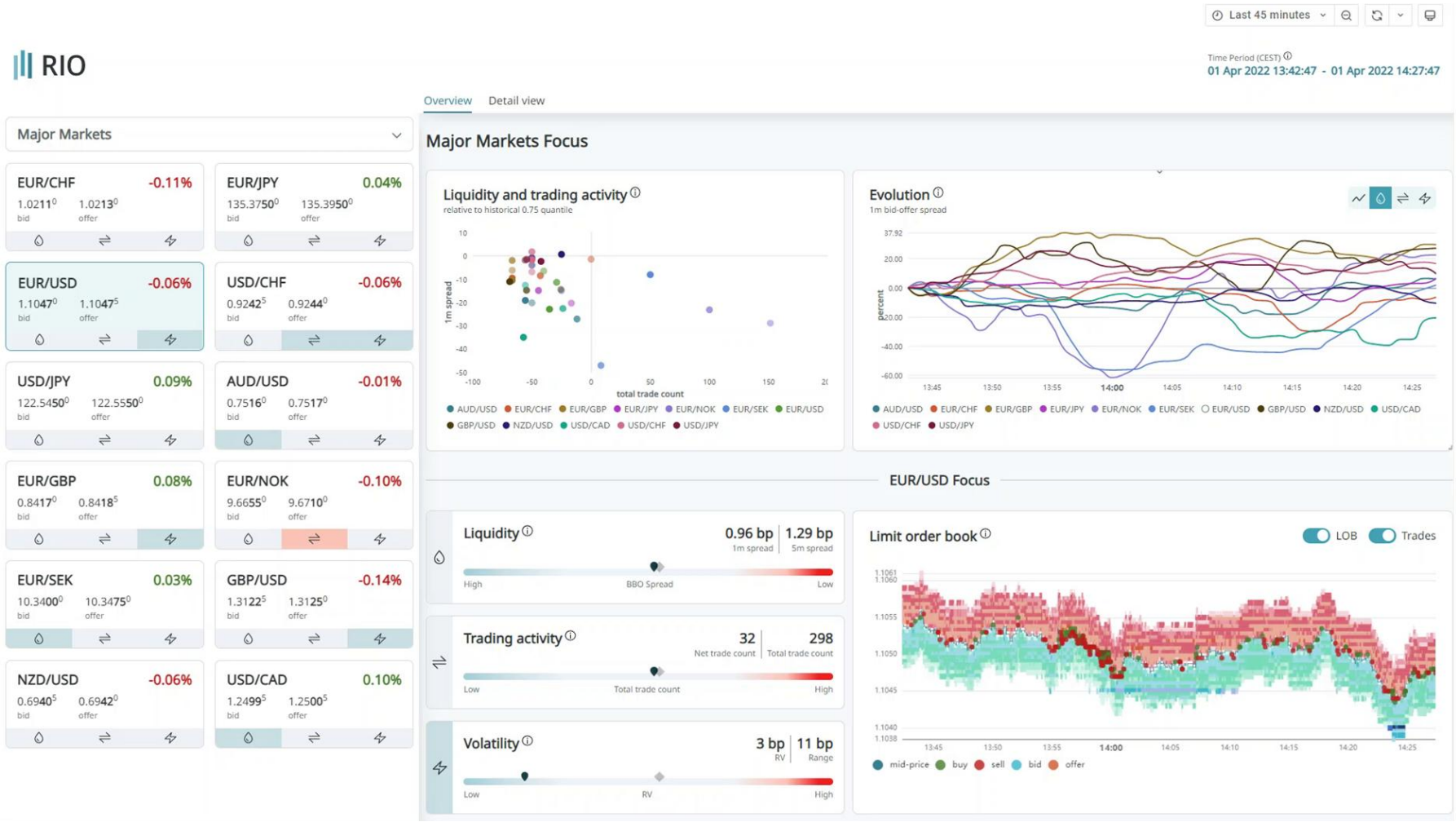
Rio's dashboard is interactive and therefore needs to be able to access the underlying data easily and quickly. During Rio's development, *Grafana* directly queried *Redis* and *TimescaleDB*. This had to be changed for three reasons. First, it created a bottleneck that slowed the dashboard as *Grafana* searched the two databases for information. Second, it made maintenance and additions to the streaming platform more onerous, as corresponding changes were also required for *Grafana*. Finally, it did not allow easy access to Rio's processed data from other potential applications.

Rio uses *FastAPI* to solve these challenges. Through this setup, a *unified API* provides the dashboard with access to data stored in *Redis* and *TimescaleDB*. The API streamlines access to the stored time series and real-time data, integrating historical and real-time data so the dashboard can provide analysis with context (Graph 9).

[7] Initial Rio prototypes trialled both *Tableau* and *PowerBI*, but neither worked as well as *Grafana* for real-time data, and licensing requirements also introduced overhead that could be avoided by opting for open source software.

The Rio dashboard

Graph 9



7. Development

Rio was built in two and a half years by a small team of developers. Initially, the project began as a quick prototype, streaming FX data and calculating simple metrics. However, as the prototype was shared with central bank experts, demand for additional fidelity and functionality grew. The early prototype grew into a minimum viable product in successive phases, successfully trialled in a pilot with central banks worldwide.

Rio's development addressed three interconnected and overlapping challenges: first, overcoming the technical difficulties and uncertainties associated with building a novel system; second, ensuring that Rio was designed to meet and serve central banks' needs; and finally, ensuring that Rio was viable and attractive as a coherent product for central banks.

7.1 People

The Rio team used the *Scrum* agile methodology to organise the project. An agile approach was essential because the project was complex and innovative, with multiple challenges. *Scrum* also helped establish a clear structure for a team that began during the Covid-19 lockdowns and did not meet in person for several months during the development phase.

The Rio team evolved throughout the project as development requirements changed. Initially, leadership was shared between an economist and a software architect seconded from the Swiss National Bank (SNB). They were supported by two engineers hired from an external firm and another economist. The development of the visualisation dashboard necessitated new skills. At this stage, the economists returned to the SNB, and the software architect led a team of external engineers, consultants, and visualisation developers from a digital agency.

7.2 Design

Rio's dashboard is the product of collaboration between seven central banks and the BIS. In a dashboard working group, central bank experts with market monitoring responsibilities met to describe their requirements. Then, prototypes were demonstrated through *design thinking* exercises and interviews with the project team, and feedback was collected in iterative cycles.

This user-centric development approach benefited Rio. The team received clear feedback on what they should prioritise, and the central bankers saw their feedback quickly incorporated into a tool built for their needs. Given the novelty of this type of development, a digital agency was hired to assist the team in following a structured process, conducting interviews with central banks and undertaking some of the front-end development work.

7.3 Pilot

Following Rio's development, a pilot exercise was conducted for central banks to better understand its value. The pilot had two parts. First, it evaluated user feedback on the product (ie central bank

experts used Rio daily). Second, it gathered information on how and where central banks encountered challenges in market monitoring that Rio could address (ie senior management was interviewed to understand broader monitoring needs).

The user pilot ran over several months with nine central banks. Feedback and ratings were collected on access to data, historical analysis, real-time analytics, visualisation and customisation. The results were positive, with an average satisfaction rating of 82%. The feedback was consistent with the design and development – central banks said that using Rio was intuitive, valuable and fitted "seamlessly" into their existing workflows. In short, the pilot established Rio as a useful and complementary tool for central banks' daily operations.

The senior management interviews added valuable business context. They highlighted the high costs of market data, long potential delivery times for technology upgrades and the attractiveness of a subscription service available through a browser.

8. Conclusion

Rio was one of the first projects at the BIS Innovation Hub. Breaking new ground, it helped many subsequent projects understand the latest technologies available to central banks. Rio was the first project at the Innovation Hub to be developed using an agile methodology and employing *design thinking*. It was also the first project to widely utilise open source technologies and involve multiple central banks at a working level. This facilitated valuable technology and know-how transfer to central banks. It was the first Innovation Hub project showcased to central bank governors. Finally, it was the first project to run a successful pilot.

Central bank market monitoring remains a challenge. Electronic markets are not getting slower and continue to grow.[8] Monitoring algorithms that can trade millions in the blink of an eye requires novel technological approaches. Rio succeeded by taking an experimental and incremental approach. By combining multiple ubiquitous software components outside central banks, the development team demonstrated that novel challenges can be addressed with innovative technology. Its components have been reused and shared with central banks, and the central bankers involved in the project discovered new ways of working collaboratively. Specifically, at the SNB, Rio catalysed stream-processing projects that underpin the most critical operations of the central bank. Rio showed that central banks can keep pace with the right approach and focus.

[8] See Bank for International Settlements, *BIS Triennial Central Bank Survey*, September 2025.

Glossary

Apache ECharts: An open source JavaScript visualisation library that makes it possible to create interactive and customisable charts and graphs for web applications. It supports various chart types, from basic line, bar and pie charts to more complex visualisations like heat maps and Sankey diagrams.

Apache Flink: An open source, unified stream processing and batch processing framework that handles bounded and unbounded data sets.

Apache Kafka: A distributed event streaming platform for building real-time data pipelines and streaming applications. It is capable of handling trillions of events a day.

Apache S4: An early Yahoo!-built platform for continuous, real-time data processing. It was officially retired in 2017, but it helped shape later streaming frameworks.

Apache Spark: An open source unified analytics engine for large-scale data processing, with built-in modules for streaming, SQL, machine learning and graph processing.

Apache Storm: Originally created by Twitter, Apache Storm is a distributed, fault-tolerant system for real-time processing of unbounded data streams.

Azure Application Gateway: A web traffic load balancer and security service from Microsoft Azure that manages and optimises the delivery of web applications, including support for SSL termination and routing rules.

Azure Blob Storage: A Microsoft Azure service for storing large amounts of unstructured data, such as text or binary files, with scalable access and redundancy options.

Azure Container Instances (ACI): A Microsoft Azure service that allows users to run Docker containers directly in the cloud without managing virtual machines or container orchestration systems.

Azure DevOps: A suite of development tools and services from Microsoft that supports source control, continuous integration and delivery (CI/CD), and project management for collaborative software development.

Azure Key Vault: A cloud service that securely stores and manages cryptographic keys, passwords, and other secrets used by applications and services.

Batch processing: A method of processing data in which transactions are collected over a period and processed together in a single batch.

Business Intelligence (BI) platforms: Software systems or technology-driven solutions that organisations use to collect, integrate, analyse, visualise and present business data.

Confluent: A data streaming platform based on Apache Kafka that provides enterprise-grade tools for building, managing, and scaling real-time data pipelines.

Databricks: A cloud-based analytics and machine learning platform built on Apache Spark that enables collaborative data engineering, data science, and real-time analytics.

Data lake: A centralised repository that stores all structured and unstructured data at any scale.

Delta Lake: An open source storage layer that brings transactions (processed **a**tomically, **c**onsistently, in **i**solation and with **d**urability ("ACID")) to Apache Spark and big data workloads.

Design thinking: A human-centred problem-solving approach that cycles through empathise, define, ideate, prototype and test stages to create solutions that balance user desirability, technical feasibility and business viability.

FastAPI: A modern, fast (high-performance) web framework for building APIs with Python based on standard Python type hints.

Grafana: An open source platform for monitoring and observability, particularly useful for visualising time series data.

Go (also known as Golang): An open source, general-purpose programming language whose features include memory safety and support for concurrent programming, making it well suited for cloud and network services, microservices and other scalable applications.

Hopping window: A windowing technique in stream processing in which windows overlap by a specified amount.

Infrastructure as code (IaC): A method of managing and provisioning infrastructure through machine-readable definition files rather than manual configuration, ensuring consistency, repeatability, and version control.

In-memory data store: A database management system that primarily relies on main memory for computer data storage in order to provide faster query responses than disk-based storage.

Kafka Connect: A tool for scalably and reliably streaming data between Apache Kafka and other systems.

Kafka Streams: A client library for building applications and microservices in which the input and output data are stored in Kafka clusters.

KSQL: An open source, Apache Kafka-native SQL engine for stream processing on Kafka topics using SQL-like queries.

Latency: The time delay between the cause and effect of some physical change in the system being observed.

Microsoft Azure: A cloud computing platform and service created by Microsoft, offering a wide range of infrastructure, platform, and software services through a global network of data centres.

Nginx: An open-source web server and reverse proxy used for load balancing, caching, and managing network traffic efficiently across distributed systems.

Order book: An electronic list of buy and sell orders for a specific security or financial instrument, organised by price level.

PostgreSQL: An advanced, open-source database management system, often shortened to "Postgres." It is renowned for its high level of SQL standards compliance, extensibility, reliability, and robust feature set that rivals many proprietary database systems.

PowerBI: Microsoft's business-intelligence suite for modeling, analyzing, and sharing interactive reports. It is tightly integrated with Excel, Azure, and Microsoft 365.

PySpark: The Python API for Apache Spark, allowing Python developers to write Spark applications using Python APIs.

Python: A high-level, interpreted, object-oriented programming language known for its clear syntax and readability. It is used for web development, data analysis, artificial intelligence, scientific computing and more.

R: An open source programming language and software environment primarily designed for statistical computing, data analysis and graphical representation.

RabbitMQ: An open source message broker that facilitates communication between applications by sending, receiving and storing messages in a distributed system. It supports multiple messaging protocols, such as AMQP (Advanced Message Queuing Protocol), and is widely used for decoupling services, load balancing and enabling asynchronous processing in microservice architectures.

Redis: An open source, in-memory data structure store used as a database, cache and message broker.

RedisTimeSeries: A Redis module that adds a dedicated time-series data structure to Redis, optimised for high-volume ingestion and real-time querying of time-stamped data. It enables efficient storage and analysis of metrics that change over time.

Scala (Scalable Language): A high-level, general-purpose programming language for web development, data processing and distributed computing. It runs using the Java Virtual Machine (JVM) runtime and allows access to Java libraries.

Scrum: An agile framework that organises work into short sprints (one to four weeks) with defined roles (product owner, scrum master, developers) and regular events (planning, daily stand-up, review, retrospective) to deliver incremental value and continuously improve.

Software-as-a-service (SaaS): A software delivery model in which applications are hosted and maintained by a provider and accessed over the internet, reducing the need for local installation or infrastructure management.

Stream processing: The continuous processing of real-time data streams to derive insights or perform actions instantaneously as data arrive.

SQL (Structured Query Language): A standardised language used to manage data in relational database management systems (RDBMS). It performs various operations on the data, including querying, manipulating, defining and controlling them.

Tableau: A data-visualization and analytics platform (by Salesforce) for exploring data and building interactive dashboards. Known for intuitive drag-and-drop visuals, broad data-source connectivity, and strong support for exploratory analysis and storytelling

Terraform: An open source infrastructure as code (IaC) tool that enables users to define and deploy cloud infrastructure using declarative configuration files across multiple service providers.

Tick size: In the foreign exchange market, it is the smallest possible price movement between two currency pairs, representing the minimum increment by which an exchange rate can change.

TimescaleDB: An open source relational database optimised for fast ingest and complex queries on time series data.

TypeScript: A statically typed superset of JavaScript (JS), the ubiquitous and dynamic language of the web that runs in browsers and on servers via Node.js. TypeScript adds optional types, interfaces, and generics with rich tooling, compiling to plain JS for broad compatibility and more reliable large-scale development.

Tumbling window: A windowing technique in stream processing in which each window is fixed in size and does not overlap with other windows.

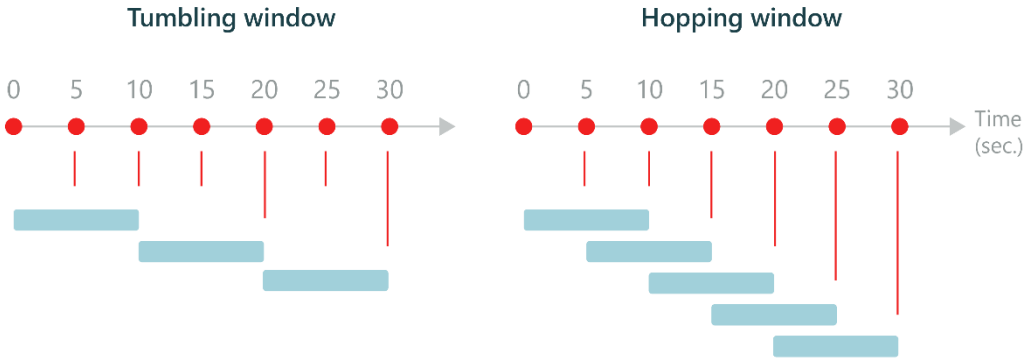
Unified API: An application programming interface that provides a single access point to multiple underlying services or systems, simplifying integration and interaction.

WebSocket: A computer communications protocol, providing full-duplex (two-way) communication channels.

Appendix A: Windowing techniques in stream processing

Following the ingestion and transformation of real-time data, a crucial step involves applying windowing techniques to process the data. Windowing is a key method in *stream processing* that involves dividing a continuous data stream into discrete segments, or "windows", for processing. This approach is vital for managing infinite data streams and delivering timely analytics. Rio specifically utilises *tumbling* and *hopping windows* (Graph A1).

Comparison of the tumbling and the hopping window — Graph A1



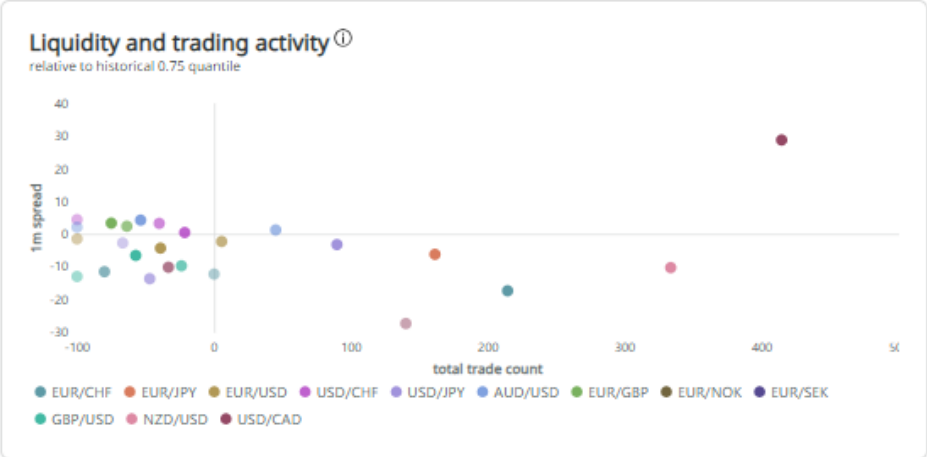
- ***Tumbling windows*:** These are fixed-size, non-overlapping intervals used to segment the data stream. Each window is processed independently, making *tumbling windows* suitable for generating consistent, regular summaries of data such as hourly trading volume or daily price volatility.

- ***Hopping windows*:** These windows overlap and are defined by two parameters: the window size and the hop size. *Hopping windows* enable more frequent updates than do *tumbling windows*, offering a more detailed view of the data. They are especially useful for applications in which it is important to smooth out the effects of anomalies in individual windows, such as in the calculation of moving averages or the detection of short-term trends.

Appendix B: Overview of Rio measures and metrics

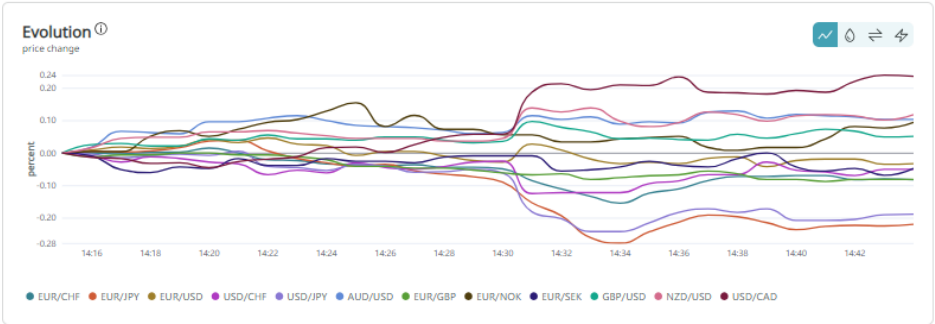
Graph B1 illustrates how currency pairs relate to liquidity and trading activity. The chosen time window is divided into three sub-intervals. For each currency pair, the three dots show the evolution of liquidity (1m bid-offer spread) and trading activity (total trade count). The darkest coloured dot (lightest) indicates the most (least) recent sub-interval. The position of the dots demonstrates the percentage deviation of total trade count and spreads from the four-week historical 0.75 quantile, with time-of-day effects considered.

Liquidity and trading activity — Graph B1



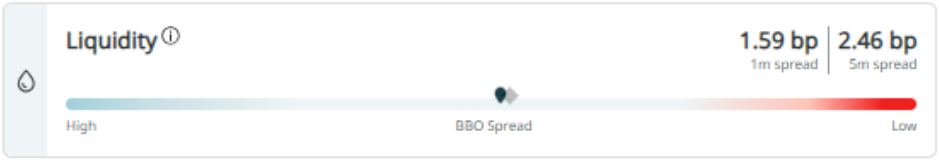
Graph B2 shows the Rio monitoring component that alternatively makes it possible to track the evolution of price (returns), 1m bid-offer spread (liquidity), realised volatility and cumulated net trade counts (trading activity as cumulated number of buy minus sell trades) for selected time window and currency pairs.

Evolution — Graph B2



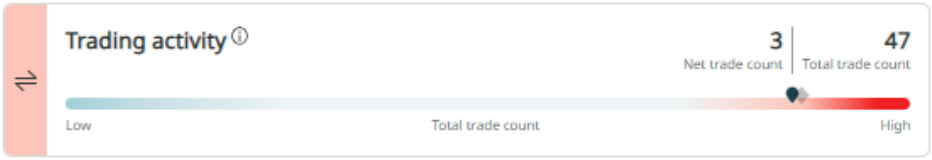
For a selected time window, Graph B3 displays the gauge showing how liquidity relates to the four-week historical distribution (with time-of-day effects considered). The figures represent the best bid-offer spread for 1m and volume-weighted bid-offer spreads for 5m (in base currency), expressed in basis points.

Liquidity — Graph B3



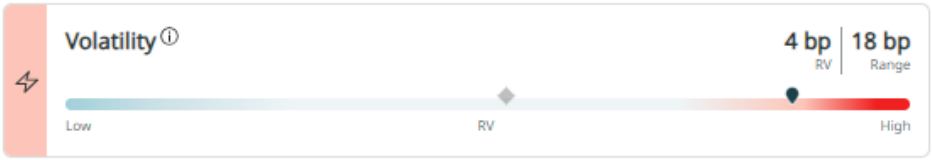
For the selected time window, Graph B4 shows the gauge indicating how trading activity compares with the four-week historical distribution (accounting for time-of-day effects). The figures are net trade count (ie number of buy minus sell trades) and total trade count (ie sum of buy and sell trades) for the selected time window.

Trading activity — Graph B4



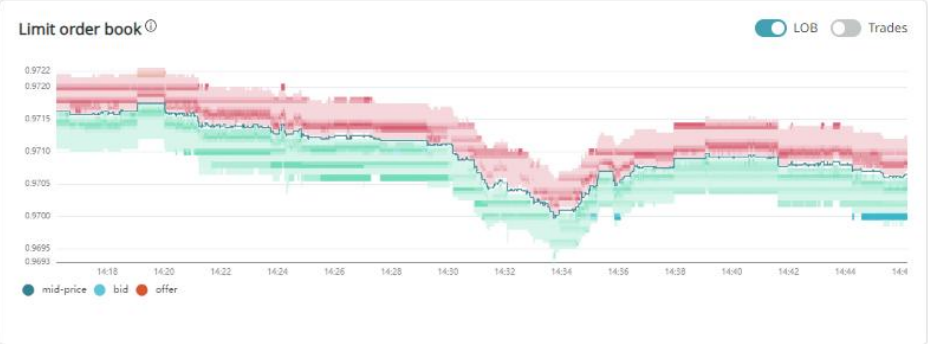
For the selected time window, Graph B5 shows the gauge indicating how volatility relates to the four-week historical distribution (taking into account time-of-day effects). The numbers display realised volatility (RV) and traded price range in basis points for the chosen time interval.

Volatility — Graph B5



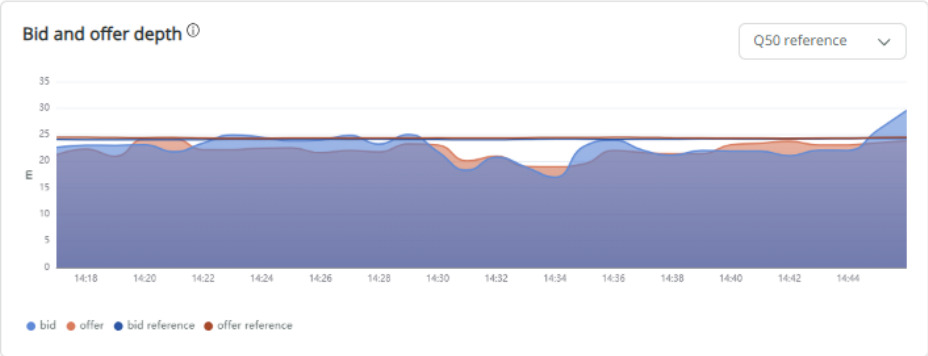
Graph B6 shows the mid-price inclusive trades and/or limit order book (LOB). For the LOB chart, darker (lighter) colour shading indicates larger (smaller) quoted volume at a price level.

Limit order book — Graph B6



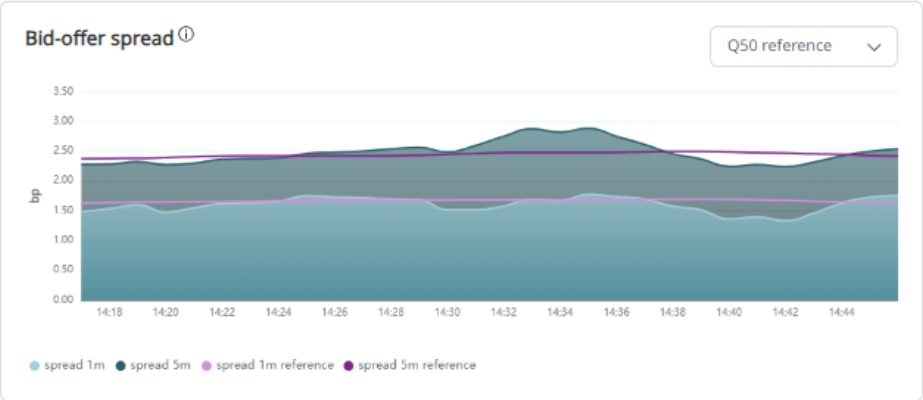
Graph B7 displays the LOB bid and offer side depth (ie the cumulative quoted volumes in millions of the base currency). Reference lines indicate quantiles derived from the four-week historical distribution, with time-of-day effects considered.

Bid and offer depth — Graph B7



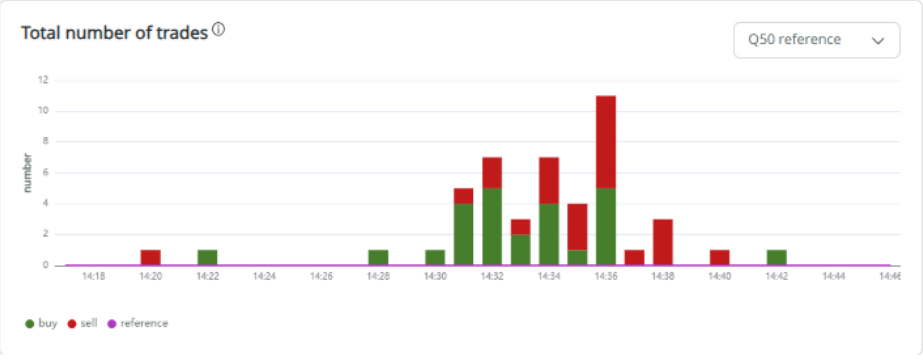
Graph B8 shows the volume weighted bid-offer spreads for 1m (ie best bid-offer spread) and 5m (in base currency) in basis points. Reference lines show quantiles from the four-week historical distribution (time-of-day effects taken into account).

Bid-offer spread — Graph B8



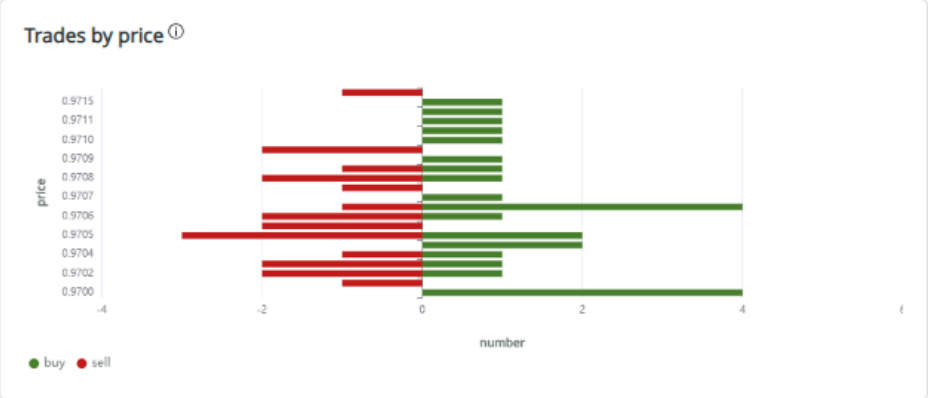
Graph B9 displays the total trade count as the sum of buy and sell trades. The reference line indicates quantiles derived from the four-week historical distribution, accounting for time-of-day effects.

Total number of trades — Graph B9



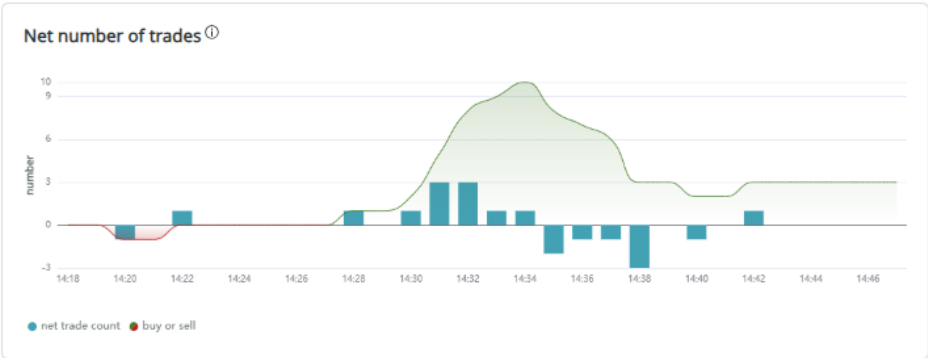
Graph B10 shows the number of buy and sell trades on the respective trade price levels aggregated for the selected time window.

Trades by price — Graph B10



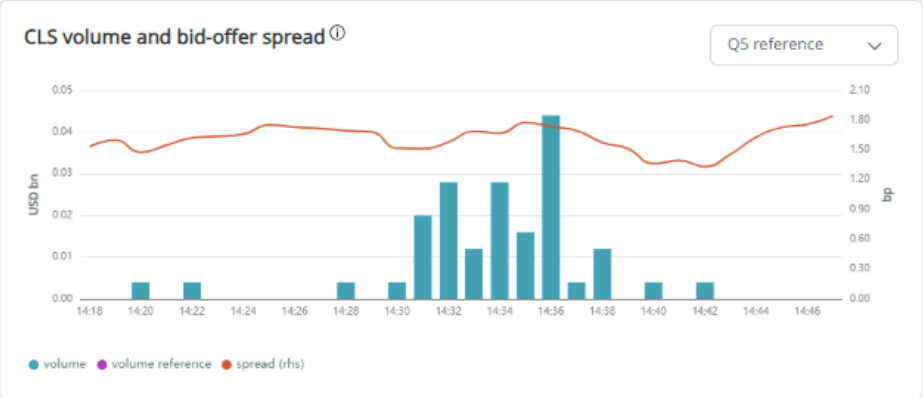
Graph B11 shows the bars for net trade counts as the number of buy minus sell trades. Shaded areas indicate the cumulated net trade counts since the start of the selected time window.

Net number of trades — Graph B11



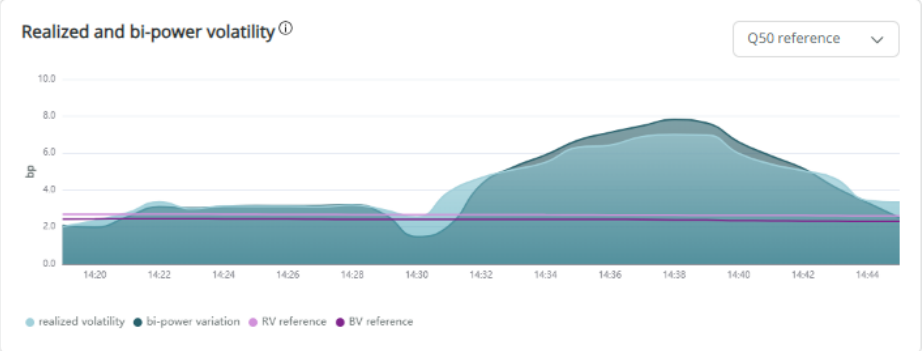
Graph B12 shows the CLS FX spot volumes in USD equivalent with 1m bid-offer spreads in basis points (right-hand scale, calculated based on FX trading venue data).

CLS volume and bid-offer spread — Graph B12



Graph B13 shows the realised volatility and bi-power variation in basis points. Reference lines show quantiles from the four-week historical distribution (time-of-day effects taken into account).

Realised and bi-power volatility — Graph B13



Graph B14 shows the candlestick chart shows price evolution over the selected time interval. A green (red) candlestick indicates a price increase (decrease).

Candlestick price — Graph B14