DOI: https://doi.org/10.5281/zenodo.12224846

OVERVIEW OF FILE SHARES AND THEIR ROLE IN MODERN COMPUTING

Istamov Mirjahan Muminjan
Tohirov Quvonchbek Musurmon o‘g‘li
Sultonov Hayotjon Baxodir o‘g ‘li
Students at the Tashkent University of Information Technologies named after Mukhammad al-Kharezmy

Abstract: *The analysis included an exploration of what SAS token is, what it contains, its impacts on protecting the files on blob storage, and the techniques for event-driven environment. The effectiveness and limitations of methods were evaluated, and directions for further research were identified.*

Key words: *Disk File System, Flash File Systems, Database File Systems and Network File Systems.*

Event-driven architectures are made up of components that have specific roles in transmitting and processing data. In general, there are the components that make data available, components that publish data onto streams or queues, components that implement a message broker or “event bus,” and one or more “components” that listen for and consume the data.

This data may be consumed to perform analysis and generate real-time visualizations. The data may be stored in a real-time database and used to generate responses to API endpoint requests. Often the data is ingested into a real-time data platform used to serve all these needs. Real-time data platforms are built with real-time databases, provide ways to filter, sort, and aggregate data, and offer ways to publish and share the data.

Let’s explore these components in more detail.

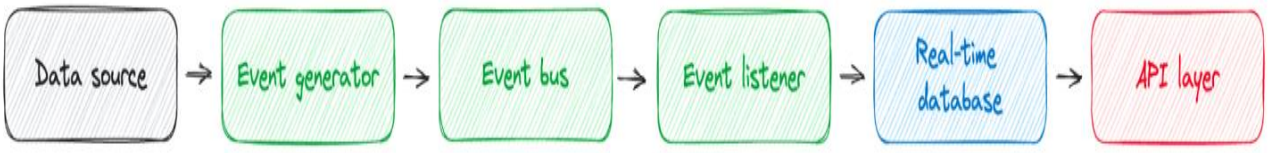

Fig 2.1

Data sources

Data sources encapsulate the reasons you want to build an event-driven system. They contain the information and data you want and need to share with other stakeholders, such as other internal systems and customers. These sources may take the form of webstore customer actions, IoT networks providing weather and logistics information, financial data, logging systems or any other data that is being generated in real time.

Often this data lives in legacy storage systems that were not built to support the low latency and high concurrency that event-driven systems demand. Commonly, this data is stored in traditional databases such as MySQL, or even as files behind a network server. The good news starts with the fact that many tools and techniques exist to integrate these sources into real-time systems. If you have legacy databases and file systems with data you'd like to share with other systems in real time, you can use change data capture (CDC). Also, see this blog post for an overview of architectural best practices for integrating databases and files.

Event generators

These components read your source data and publish it on a stream or queue. Generators write data to a message broker or event bus, making the data available to event listeners.

This component can range from a CDC component that writes database data, to custom code that writes data objects, to the event bus. For example, Debezium and its cloud-hosted variants are popular tools for implementing CDC-based sources. Also, here is an example Python script that writes data directly to an event bus.

Event buses

An event bus is where events are loaded and where events are offloaded. This component is responsible for real-time data ingestion from the generators and for making that data available to components listening for the data.

There are many forms of event buses, often referred to as streams or queues. Options for streaming components include Apache Kafka, Amazon Kinesis, Google Pub/Sub, Confluent Cloud and Redpanda. Likewise, there are many excellent choices for implementing queues, such as Amazon SQS, RabbitMQ and Redis.

These components are based on classic publish/subscribe concepts and offer a range of features and capabilities to meet the diverse needs of organizations.

Event buses can be configured to retain a custom window of data, enabling data consumers to get data on their own schedule. The advent of event bus architectures has been a boon for anyone who requires full-fidelity and reliable systems. If a consumer fails, it can reconnect and start where it left off.

Event Listeners/Subscribers

These components consume the data, and there is commonly more than one listener. While most consumers are designed to read data only, it is possible to have consumers that remove data from the event bus or queue.

Listeners are designed to access data as soon as it is available by continuously polling the event bus for topics and messages of interest. In some cases, it may be possible to implement triggered polling, where some other signal is given to start the continuous polling process.

Listeners also have the responsibility to do something with the incoming data — to write the data somewhere. This could be to databases, data lakes or even additional downstream streams and queues.

Real-Time Databases

One common destination that listeners write to is a database. To handle the data volumes and velocities typically associated with EDAs, it's important to use a real-time database. There are several options here, including the open source Apache Druid, Apache Pinot and ClickHouse. These open source database packages are all also available via a cloud-hosted service.

These databases can be primary or secondary sources of data. Primary sources are the original and authoritative source of data, while secondary sources are copies of data from multiple sources. This compilation of data from multiple sources is a common motivation for building EDAs.

Like traditional databases, these databases support SQL for filtering, sorting, aggregating and joining data. So it's good news that these cutting-edge data storage tools also support a querying language widely used across a large range of technical roles. Chances are that you and your colleagues are well-equipped to start analyzing and integrating these new data streams.

In addition, these databases may support real-time, incremental materialized views, which auto-populate query results into new table views as event-driven data is ingested in real time.

Publication Layer

In most cases, event-driven systems are built to make real-time data available to a variety of consumers and stakeholders. These stakeholders may include data scientists and analysts performing ad hoc analysis, dashboards and report generators, web and mobile application features driven by real-time data or automated control systems that take actions without human intervention.

While this data availability may be implemented with a wide range of methods, ranging from webhook events to generating flat files, the most common method is building API endpoints for consumers to request data from. These API endpoints have

the advantage of being extremely flexible, since they are able to serve customized data content to their consumers.

Real-Time Data Platforms

Real-time data platforms combine many of the components that EDAs are built with. These platforms include native data connectors for both streaming and batch data sources. In the case of streaming sources, these platforms provide ways to seamlessly consume from a variety of event buses such as Apache Kafka and others implementing the publisher/subscriber model. In addition, the platforms typically provide an endpoint for streaming data into it.

The platforms also manage data storage of the incoming data by integrating real-time databases. The systems are typically built on top of open source real-time databases, which enable them to manage and process high volumes and velocities of data.

Along with these integrated databases comes the ability to perform data analysis with SQL. The platforms commonly provide user interfaces for writing and designing queries for filtering and aggregating data and joining from multiple data sources.

Finally, the platforms integrate methods for publishing and sharing data. In some cases, they are used to publish data to streams or export data in a batch process. Most advanced platforms make it possible to serve data via low-latency and high-concurrency data APIs.

The advantage of building a real-time data platform into your event-driven architecture is that by combining fundamental EDA components, they remove many of the complexities of using separate components and ‘gluing’ them together. In particular, self-hosting real-time databases and building APIs from scratch demand experience and expertise that is abstracted away by real-time data platforms.

Event-Driven System Design Patterns

To demonstrate how these components fit together, here are two reference architectures (For additional architecture examples, see this blog post).

First, we have a fundamental pattern that focuses on data storage. Here incoming events are immediately stored in three different types of storage:

Transactional database — may power functionality for user-facing applications and typically persist fundamental state information keys.

Data warehouse — built for large datasets, long-term storage and building historical archives.

Real-time database — may power real-time analytics and a publication layer. Built to support low-latency data retrieval and high concurrency and is well-suited to serve data consumers at scale.

Here we have three listeners and destinations for new event data. With this design, the data warehouse makes requests for new data directly from the real-time database.

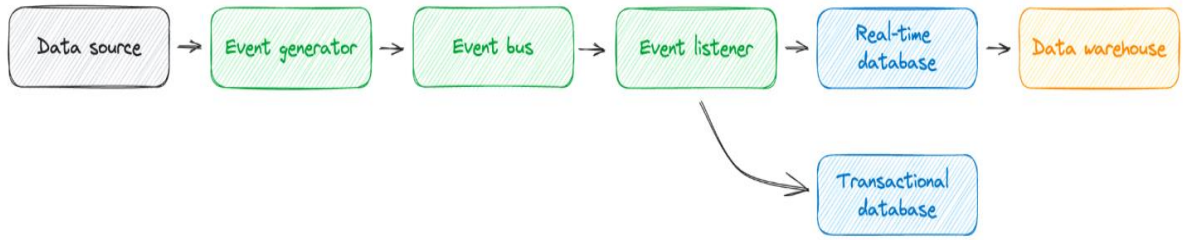

Fig 2.2

We can extend that design by adding a real-time data analytics platform, along with a publication layer. With this type of design, the data analytics platform encapsulates several EDA components.

For example, here the real-time data platform includes a "connector" that listens for events and consumes them, provides a real-time database and analytical tools, and is able to host APIs for sharing data and analysis.

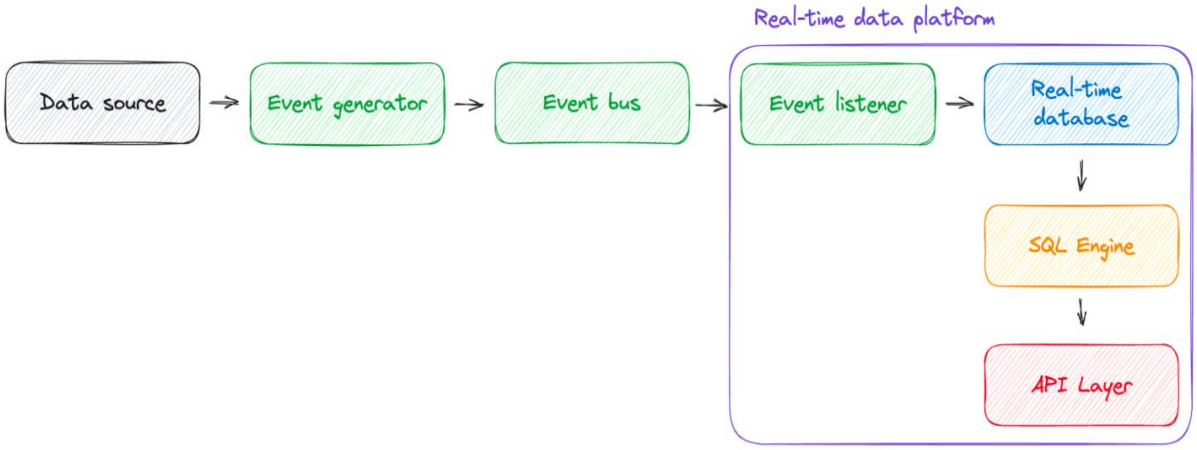

Fig 2.3

Whether you are designing for something new or revisiting designs implemented a long time ago, it's worth exploring how event-driven architecture patterns can help. You can probably identify many cases where introducing real-time data would improve your product, system or user experience.

If you are in the business of building customer-facing apps, you probably already have a list of data-driven features that would delight your customers. At a minimum, there are probably a few existing pain points that are due for a tune up, along with lots of opportunities for small, quick performance improvements.

As you get started, there are three distinct areas to consider. First, identify where and how the data you want to build with is generated, stored and made available. Perhaps you have a data source that is already written to a stream or queue and all you have to do is add a new listener. Perhaps you have a backend database that you can integrate using change data capture techniques.

Second, decide what type of "event bus" to implement and start building the bridge from where events are generated to where you can listen for them. As mentioned above, there are many open source solutions available that can be self-hosted or cloud-hosted.

Third, with your data sources and event stream sorted out, it's time to build data consumers.

Real-time data platforms are a common type of event data consumer. These platforms integrate many system components into a single package. For example, Tinybird is a real-time data platform that manages real-time event ingestion and storage, provides real-time data-processing and analysis tools, and hosts scalable, secure API endpoints.

CONCLUSION

This approach addresses key challenges in secure file access and efficient event handling, offering a robust solution for modern cloud-based applications.

This work includes:

- Exploring event-driven architecture
- SAS token in azure cloud services
- Integration of SAS tokens with 2 microservices in event-driven environment

The primary objectives of this work were to ensure secure access to shared files, provide scalability through event-driven architecture, and maintain efficient communication between various components of the system.

REFERENCES

1. https://www.researchgate.net/publication/373046292_Uncovering_the_Hidden_Potential_of_Event-Driven_Architecture_A_Research_Agenda
2. ASP.NET Core in Action - https://www.manning.com/books/asp-net-core-in-action-third-edition
3. Enterprise Service Bus - https://www.oreilly.com/library/view/enterprise-service-bus/0596006756/
4. Microsoft Azure Storage - https://www.amazon.com/Microsoft-Azure-Storage-Definitive-Practices/dp/013759318X
5. Designing Data-Intensive Applications - https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/
6. https://github.com/asindarov/kabutar
7. "Protecting Your API with OAuth 2.0" by Brian Pontarelli
8. "Building Event-Driven Applications with Azure Functions" by Jeff Hollan
9. OAuth 2.0 Authorization Framework (RFC 6749)