

O'REILLY®



Compliments of

imply

Building Real-Time Analytics Applications

Operational Workflows
with Apache Druid

Darin Briskman

REPORT



The analytics database when real-time matters.

Trusted by 1000s of companies



CONFLUENT



reddit

salesforce

NETFLIX

Walmart*

Building Real-Time Analytics Applications

*Operational Workflows
with Apache Druid*

Darin Briskman

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Building Real-Time Analytics Applications

by Darin Briskman

Copyright © 2023 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Acquisitions Editor: Aaron Black
Development Editor: Angela Rufino
Production Editor: Beth Kelly
Copyeditor: nSight, Inc.

Proofreader: O'Reilly Media, Inc.
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Kate Dullea

February 2023: First Edition

Revision History for the First Edition

2023-02-03: First Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Building Real-Time Analytics Applications*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and Imply. See our [statement of editorial independence](#).

978-1-098-14656-6

[LSI]

Table of Contents

Building Real-Time Analytics Applications.....	1
What Is a Real-Time Analytics Application?	1
Examples of Real-Time Analytics Applications	4
Key Components for Real-Time Analytics Applications	9
Druid: A Database for Real-Time Analytics Applications	14
Creating Real-Time Analytics Applications	22
Conclusion	28
Further Resources	28

Building Real-Time Analytics Applications

This report will introduce you to the real-time analytics applications that organizations are building to power new operational workflows. You'll learn about the purposes of these applications, the value they provide, and the technologies needed to create them.

Once you've completed this report, you'll know how real-time analytics applications can help your organization, and you'll know what you'll need to create a solution that works for you.

What Is a Real-Time Analytics Application?

Every organization needs insight to succeed and excel. While insights can come from many wellsprings, the foundation for insights is *data*: both internal data from operational systems and external data from partners, vendors, and public sources.

For decades, the traditional approach to analytics has focused on data warehousing and business intelligence, where experts query historical data “once in a while” for executive dashboards and reports.

Meanwhile, transactional applications have been the lifeblood of business operations, storing data that enables sales, marketing, manufacturing, shipping, payroll, supply chain, customer service, financial reporting, and many other functions to operate.

Leading organizations like Netflix, Target, Salesforce, and Wikimedia Foundation¹ have recognized the opportunity and value of bringing together analytics and applications in a new way. Instead of infrequent analytics queries on historical data, their developers are building a new type of application that queries everything from terabytes to petabytes of real-time and historical data at subsecond response under load.

This new approach is the real-time analytics application, and it's formed at the intersection of the analytics and application paradigms, with technical requirements that bring the scale of data warehouses to the speed of transactional databases (Figure 1).

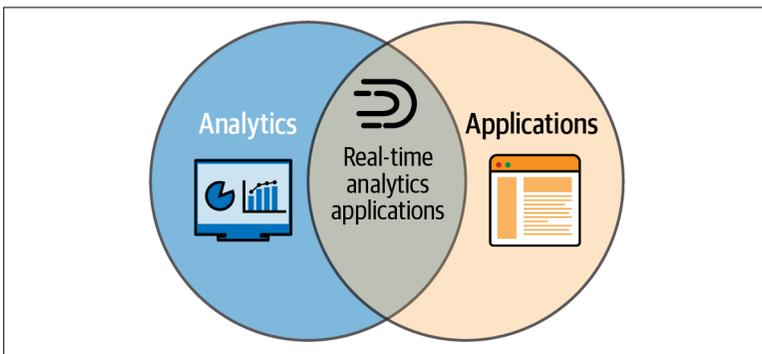


Figure 1. Real-time analytics applications combine characteristics from both traditional reporting analytics and transactional applications

Real-time analytics applications generally share three common technical characteristics:

Subsecond performance at scale

This allows humans and, sometimes, machines to quickly and easily see and comprehend complex information and to hold interactive conversations with data, drilling down to deep detail and panning outward to global views. Many real-time analytics solutions support interactive conversations with large data sets, maintaining subsecond performance even with dozens of petabytes of data.

¹ Druid, “Powered by Apache Druid”, accessed January 11, 2023.

High concurrency

This enables large numbers of users to generate multiple queries as they interact with the data. Architectures that support a few dozen concurrent queries aren't sufficient when thousands of concurrent queries must be executed simultaneously. Of course, this must be done affordably, without requiring large installations of expensive infrastructure.

Real-time and historical data

Real-time data is usually delivered in streams, using tools like Apache Kafka, Confluent Cloud, or Amazon Kinesis. Data from past streams and from other sources, such as transactional systems, is delivered as a batch, through extract, load, and transform (ELT) processes. The combination of data types allows both real-time understanding and meaningful comparisons to the past.

A core element of real-time analytics is use of *data streams*. Streaming data are events that are continuously generated and can come from a wide range of sources, including web users' clickstreams, sensors in vehicles, actions in video games, and changes in databases. The events are collected and transported by stream processors, such as Apache Kafka or Amazon Kinesis.

Traditional *batch* analytics collects incoming events into groups, eventually making them available for analysis. While this is good enough for daily or monthly reports, it doesn't provide the current (or "real-time") data needed for many modern requirements. Data streams enable immediate delivery of events as they happen, but more is required to create real-time analytics applications (Figure 2).

An additional requirement is common for real-time analytics: *continuous availability*. Unlike reporting analytics, where outages have limited impact, real-time analytics have no tolerance for downtime or data loss.

These capabilities enable a wide range of projects, from operational visibility at scale to customer-facing analytics to unrestricted data exploration to real-time decisioning.

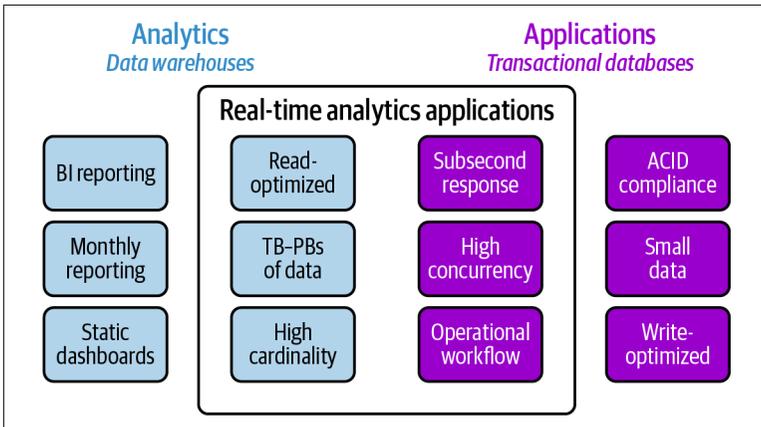


Figure 2. Real-time analytics applications combine characteristics from both traditional reporting analytics and transactional applications

Examples of Real-Time Analytics Applications

Let's look at a few of the places where organizations are using real-time analytics.

Netflix

How can we be confident that updates are not harming our users? And that we're actually making measurable improvements when we intend to?

—Ben Sykes, software engineer, Netflix²

To ensure a consistently great experience to more than 100 million members in more than 190 countries enjoying 125 million hours of TV shows and movies each day, Netflix built a real-time analytics application for user experience monitoring. By turning log streams into real-time metrics, Netflix can see how over 300 million devices are performing at all times in the field. Ingesting over 2 million events per second and querying over 1.5 trillion rows, Netflix engineers can pinpoint anomalies within their infrastructure, endpoint activity, and content flow.

² Ben Sykes, "How Netflix Uses Druid for Real-Time Insights to Ensure a High-Quality Experience", *Netflix Technology Blog*, March 3, 2020.

An ongoing challenge for Netflix is consistently delivering a great streaming entertainment experience while continuously pushing innovative technology updates. As Netflix’s adoption has skyrocketed, this challenge has grown more complex. With over 300 million devices spanning four major UIs including iOS, Android, smart TVs, and their own website, Netflix has a constant need to identify and isolate issues that may affect only a certain group, such as a version of the app, certain types of devices, or particular countries. Ben Sykes, software engineer at Netflix, says that “with this data arriving at over 2 million events per second, getting it into a database that can be queried quickly is formidable. We need sufficient dimensionality for the data to be useful in isolating issues and as such we generate over 115 billion rows per day.”³

To quantify how seamlessly users’ devices are handling browsing and playback, Netflix derives measurements using real-time logs from playback devices as a source of events (Figure 3).

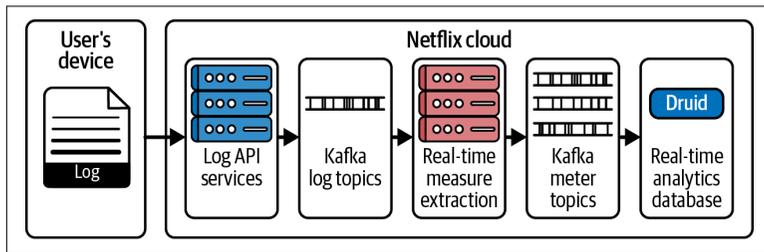


Figure 3. The log-to-metric data pipeline at Netflix⁴

Netflix collects these measures and feeds them into the real-time analytics database. Every measure is tagged with anonymized details about the kind of device being used—for example, whether the device is a smart TV, an iPad, or an Android phone. This enables Netflix to classify devices and view the data according to various aspects. This aggregated data is available immediately for querying, either via dashboards or ad hoc queries. “We’re currently ingesting at over 2 million events per second,” Sykes says, “and querying over 1.5 trillion rows to get detailed insights into how our users are

³ Sykes, “How Netflix Uses Druid for Real-Time Insights.”

⁴ Sykes, “How Netflix Uses Druid for Real-Time Insights.”

experiencing the service. All this helps us maintain a high-quality Netflix experience, while enabling constant innovation.”⁵

The ultimate benefit is speed, which is essential for a service that needs to react to a massive number of users in near real time.

Walmart

Druid has been designed to...enable exploration of real-time data and historical data while providing low latencies and high availability.

—Kartik Khare, software engineer, Walmart Labs⁶

To compete with Amazon and other retailers, Walmart needs to track the pricing of their competitors in real time and allow analysts to explore the gathered data interactively.

Many data sets at Walmart are generated by the digital business, modeled as streams of events ranging from server logs to application metrics to product purchases. The goal of the team at Walmart Labs is to make it easy for the right people across Walmart to access this data, analyze it, and make decisions in the least amount of time possible.

Walmart’s first attempt to provide low-latency analytics was to leverage the Hadoop ecosystem. It tried using Apache Hive first and then Presto. According to Amaresh Nayak, distinguished SW engineer at Walmart Labs, “The problem we faced with both of these SQL-on-Hadoop solutions was that queries would sometimes take hours to complete, which significantly impacted our ability to make rapid decisions. Although our data was arriving in real time, our queries quickly became a bottleneck in our decision-making cycle as our data volumes grew. We quickly realized that the workflow we were aiming to optimize was one where we could look at our event streams (both real-time and historical events) and slice and dice the data to look at specific subsections, determine trends, find root causes, and take actions accordingly.”⁷

5 Sykes, “How Netflix Uses Druid for Real-Time Insights.”

6 Kartik Khare, “What Makes Apache Druid Great for Realtime Analytics?”, Medium, March 30, 2019.

7 Amaresh Nayak, “Event Stream Analytics at Walmart with Druid”, Medium, November 24, 2017.

The team at Walmart knew it needed to make a change. The types of queries the team runs require column aggregation, which involves scanning a lot of rows across multiple shards. Walmart found that relational databases were poorly suited for this because they cannot efficiently enable data exploration in real time. The team also ruled out a NoSQL key-value database because of the need to query multiple partitions across a number of nodes. Using this database would have caused inefficient aggregation calculations and exponentially increased storage requirements by storing aggregates for all possible column combinations.

Using a real-time analytics application, Walmart can now pre-aggregate records as they are being ingested. Instead of getting a single price for an item for a specific moment in time, Walmart can now understand the changing price of an item over any span of time. This combined level of depth and speed to insights is essential to enabling Walmart to make critical pricing decisions in the least amount of time possible. “After we switched [to real-time analytics],” Nayak says, “our query latencies also dropped to near sub-second and in general, the project fulfilled most of our requirements. Today, our cluster ingests nearly 1B+ events per day (2TB of raw data).”⁸

Confluent

We don't shy away from high-cardinality data, which means we can find the needle in the haystack. As a result, our teams can detect problems before they emerge and quickly troubleshoot issues to improve the overall customer experience. The flexibility we have with Druid also means we can expose the same data we use internally also to our customers, giving them detailed insights into how their own applications are behaving.

—Xavier Léauté and Zohreh Karimi, lead engineers, Confluent⁹

Founded by the creators of Apache Kafka, Confluent provides a range of services for streaming, including Confluent Cloud, a fully managed cloud native data-streaming service. To enable effective customer support, Confluent built an internal-facing real-time analytics observability application. Ingesting over 3.5 million events per

⁸ Nayak, “Event Stream Analytics at Walmart with Druid.”

⁹ Xavier Léauté and Zohreh Karimi, “[Confluent Cloud: Operational Insights at Scale](#)”, Imply, Druid Summit (virtual conference), December 10, 2021.

second and handling hundreds of queries per second on the data set, Confluent provides real-time insights into the operations of thousands of clusters within Confluent Cloud.

Confluent also leveraged real-time analytics to build an external-facing application, Confluent Health+, which extends performance and health insights to their customers.

Confluent's first attempt at building an observability pipeline was with a NoSQL database, which was used to store and query telemetry data. As the volume of data grew, Confluent's legacy pipeline struggled to keep up with its data ingestion and query loads. Next, Confluent looked into common observability solutions but quickly determined that these, too, could not handle its requirements. As noted by Xavier Léauté and Zohreh Karimi, lead engineers at Confluent, "Operating multi-tenant services requires fine-grained visibility down to the individual user, tenant, or application behavior, where most traditional monitoring stacks fail to scale or become cost-prohibitive."¹⁰

To keep up with their data growth, Confluent determined their next-generation observability pipeline needed to support the following:

- Substantially increased data and query load (100x)
- Subsecond query latencies on high-cardinality metrics
- Inherent time-series data support

After evaluation and testing, Confluent found that real-time analytics application could meet all requirements (Figure 4). All Confluent Cloud clusters, as well as customer-managed, Health+-enabled clusters, publish metrics data to Confluent's telemetry pipeline. The real-time analytics application is a key differentiator for Confluent, enabling Confluent's customers to get technical support and on-demand monitoring that isn't available from using open source Kafka alone.

¹⁰ "Kafka to Druid Stack", Imply, accessed January 9, 2023.

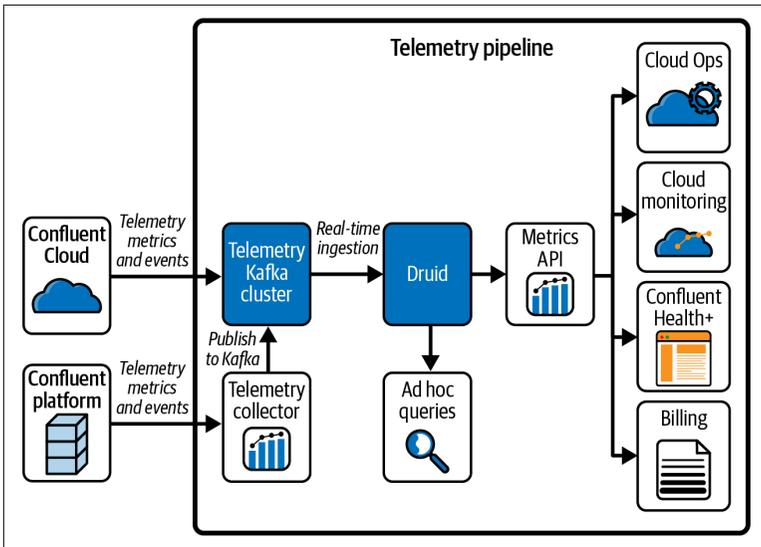


Figure 4. The telemetry pipeline at Confluent¹¹

Key Components for Real-Time Analytics Applications

Any real-time application requires several key components.

Data Sources

Data needs to come from somewhere. At least some of the data will be continuously generated, but some data may come from static sources.

For example, a real-time analytics application that places internet advertising may combine real-time sources (real-time bidding auctions, user clickstreams) with historical data (effectiveness of past advertising) and external data that is periodically updated (census data, demographic data).

¹¹ Zohreh Karimi and Harini Rajendran, “Scaling Apache Druid for Real-Time Cloud Analytics at Confluent”, Confluent, November 8, 2021.

Data Transportation

Data must be transported from the data sources to the database.

To keep the data as close to real time as possible, events should be transported using a stream platform. The most common of these is Kafka, available as open source from the Apache Software Foundation or as a managed service from many providers, including Confluent, Amazon Web Services (AWS), and Aiven. Most other stream platforms use a Kafka-compatible interface, such as Red Panda and Azure Event Hub. Another option, only available on Amazon Web Services, is Kinesis Data Streams. Any of these stream platforms can deliver data quickly, reliably, and consistently (Figure 5).

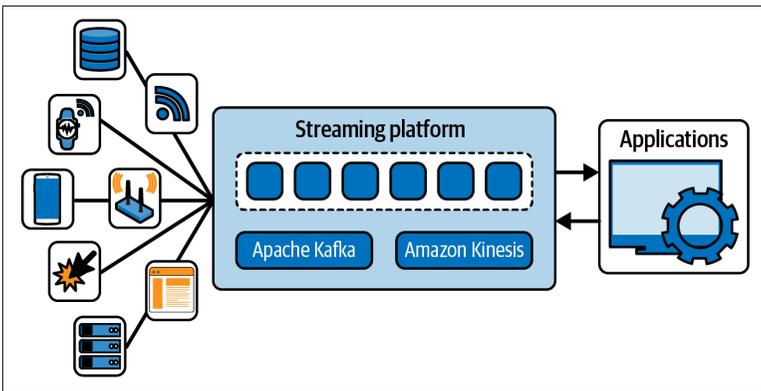


Figure 5. Events from many sources are delivered to applications by a stream platform, such as Apache Kafka or Amazon Kinesis

Sometimes, historical, external, or other slower-changing data must be extracted from sources (such as transactional databases), loaded to the real-time analytics database, and transformed to the needed data schema for the application. In many cases, this can be done by connecting the transactional database to a stream platform, with data changes appearing as events. For example, Debezium is an open source package that monitors MySQL, PostgreSQL, MongoDB, Cassandra, and other databases and copies database changes into Kafka. Another option is to load data into the database from a file, which can be created from any database using that database's data dump feature or a data query. For continuous monitoring and file creation, there are many ELT tools available.

Database

Like nearly any application, real-time analytics applications require a persistent service where data can be reliably stored and retrieved. There are hundreds of databases available, with both open source and commercial options, and nearly any database can, in theory, be used to support a real-time analytics application. However, an application that can provide the needed performance, scale, and reliability for real-time analytics will require a database with some specific capabilities.

Speed is critical for real-time analytics applications: delivering data in milliseconds isn't useful unless data queries also execute in milliseconds. A real-time analytics database must be able to both ingest incoming events and process queries with subsecond performance, even for large data sets of hundreds of terabytes or petabytes.

How quickly can data be added to the database? All databases support moving sets of records from files into the database, usually known as *batch ingestion*. For many analytics databases, such as Snowflake and Amazon Redshift, this is the only method of data ingestion. Only a few databases designed for real-time data analytics, such as Apache Druid, can perform both batch ingestion and *stream ingestion*, with each event becoming immediately available for queries as soon as it arrives.

Many real-time analytics applications also require high concurrency. A customer-facing analytics application, enabling customers of an SaaS or other provider to monitor and manage their own activities, might have anywhere from a few to many customers seeking to use the analytics at the same time. Similarly, interactive data conversations drive dozens or hundreds of queries as analysts seek to gain insights from large data sets. To support such applications, a real-time analytics database must be able to execute hundreds of queries per second.

Ironically, a real-time analytics application needs to analyze more than real-time data: it needs to enable a comparison of real-time and historical data to provide needed context (Figure 6). For example, an application that monitors financial events needs to be able to compare current events to past events. Is this a lot of activity? A little? How does the current pattern compare with historical patterns? A real-time analytics database must be able to query both incoming

real-time events and historical data and to ingest both fast-moving data streams and static batch data.

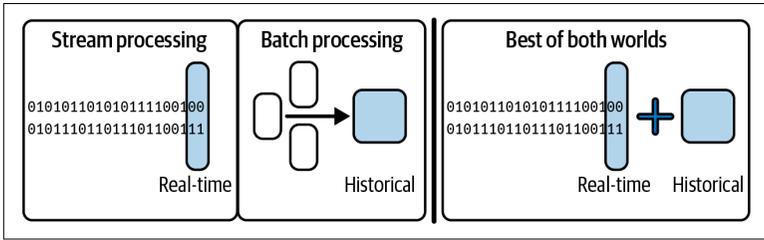


Figure 6. Real-time analytics applications need both stream and batch processing

The final key requirement for a real-time analytics database is resiliency (Figure 7). Reporting analytics are tolerant of downtime (when you're producing a monthly report, a few hours offline isn't a problem), but real-time analytics are usually always on. This requires a database that can scale up (and scale down) without outages and support upgrades without a need for planned downtime. Data must also be durable, with no data loss when hardware fails. Resiliency is the combination of high reliability and high durability.

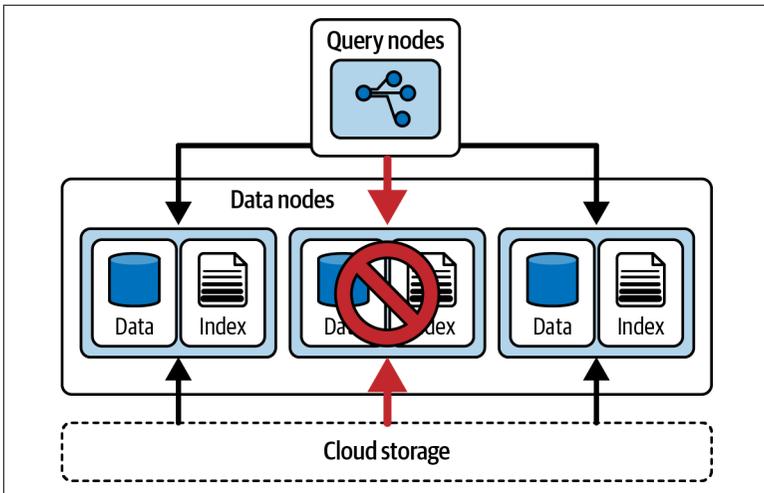


Figure 7. When hardware fails, resilient systems continue to work

For real-time analytics applications that are small (less than 1 terabyte of total data) and don't require high concurrency and availability, nearly any high-speed database is sufficient. Common choices include PostgreSQL, MongoDB, and ClickHouse, each available as open source software or as managed services from many providers. There are many commercial databases that are also suitable.

If the real-time analytics application requires some combination of scale, fast query performance, fast ingestion performance, high concurrency, real-time plus historical data in context, and high resiliency, there are only a few options available. Some databases provide many of these characteristics. One that provides all of them is Apache Druid, which is described in more detail later in this report.

Visualization

If the real-time analytics application is to be used by humans, it will probably need some sort of visualization engine to render data into human-consumable graphics.

While there are many commercial and open source visualization tools (Grafana is probably the most widely used), most of them are designed for the speed of reporting analytics and cannot render visualizations fast enough to keep up with the needs of real-time analytics applications.

There are a few exceptions. Apache Superset is a large-scale data exploration and visualization toolset created to work with the Apache Druid database. D3 (Data-Driven Documents) is an open source JavaScript library for visualizing and analyzing data, commonly used to create high-performance visualization tools for real-time analytics applications (Figure 8).

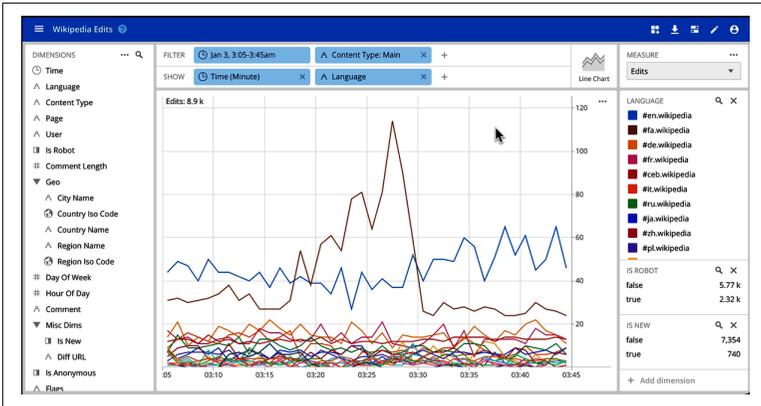


Figure 8. An example of a high-performance visualization tool created using D3

Druid: A Database for Real-Time Analytics Applications

As mentioned, one database that meets all the criteria for real-time analytics application is Apache Druid. It enables subsecond performance at scale, provides high concurrency at the best value, and easily ingests and combines real-time streaming data and historical batch data. It is a high-performance, real-time analytics database that is flexible, efficient, and resilient.

Origins of Druid

In 2011, the data team at a technology company had a problem. It needed to quickly aggregate and query real-time data coming from website users across the internet to analyze digital advertising auctions. This created large data sets, with millions or billions of rows.

The data team first implemented its product using relational databases. It worked but needed many machines to scale, and that was too expensive.

The team then tried the NoSQL database HBase, which was populated from Hadoop MapReduce jobs. These jobs took hours to build the aggregations necessary for the product. At one point, adding only three dimensions on a data set that numbered in the low millions of rows took the processing time from 9 hours to 24 hours.

So, in the words of Eric Tschetter, one of Druid's creators, "we did something crazy: we rolled our own database."¹² And it worked! The first incarnation of Druid scanned, filtered, and aggregated one billion rows in 950 milliseconds.

Druid became open source a few years later and became a top-level project of the Apache Software Foundation in 2016.

As of 2023, over 1,400 organizations use Druid to generate insights that make data useful, in a wide variety of industries and a wide range of uses. There are over 10,000 developers active in the Druid global community.

Scalable and Flexible

Druid has an elastic and distributed architecture to build any application at any scale, enabled by a unique storage-compute design with independent services (Figure 9).

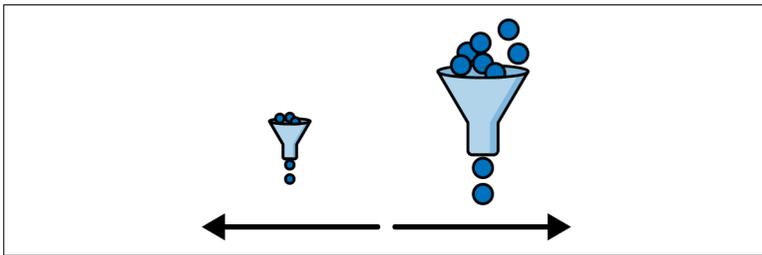


Figure 9. Druid scales up and down to meet changing requirements

During ingestion, data is split into segments, fully indexed, and optionally pre-aggregated. This enables unique value over other analytics databases, which force a choice between the performance of tightly coupled compute and storage or the scalability of loosely coupled compute and storage. Druid gets both performance and cost advantages by storing the segments on cloud storage and also prefetching them so they are ready when requested by the query engine.

Each service in Druid can scale independently of other services. Data nodes, which contain prefetched, indexed, segmented data, can

¹² Eric Tschetter, "Introducing Druid: Real-Time Analytics at a Billion Rows Per Second", Druid, April 30, 2011.

be dynamically added or removed as data quantities change. Meanwhile, query nodes, which manage queries against streams and historical data, can be dynamically added or removed as the number and shape of queries change.

A small Druid cluster can run on a single computer, while large clusters span thousands of servers and are able to ingest multiple millions of stream events per second while querying billions of rows, usually in under one second.

Efficient and Integrated

Performance is the key to interactivity. In Druid, the key to performance is “if it’s not needed, don’t do it.” This means minimizing the work the cluster has to do.

Druid doesn’t load data from disk to memory, or from memory to CPU, when it isn’t needed for a query. It doesn’t decode data when it can operate directly on encoded data. It doesn’t read the full data set when it can read a smaller index. It doesn’t start up new processes for each query when it can use a long-running process. It doesn’t send data unnecessarily across process boundaries or from server to server.

Druid achieves this level of efficiency through its tightly integrated query engine and storage format, designed in tandem to minimize the amount of work done by each data server. Druid also has a distributed design that partitions tables into segments, balances those segments automatically between servers, quickly identifies which segments (or replicas of segments) are relevant to a query, and then pushes as much computation as possible down to individual data servers.

The result of this unique relationship between compute and storage is very high performance at any scale, even for data sets of multiple petabytes.

Resilient and Durable

Druid is self healing, self balancing, and fault tolerant. It is designed to run continuously without planned downtime for any reason, even for configuration changes and software updates. It is also durable and will not lose data, even in the event of major systems failures.

Whenever needed, you can add servers to scale out or remove servers to scale down. The Druid cluster rebalances itself automatically in the background without any downtime. When a Druid server fails, the system automatically understands the failure and continues to operate.

As part of ingestion, Druid safely stores a copy of the data segment in deep storage, creating an automated, continuous additional copy of the data in cloud storage or HDFS. It both makes the segment immediately available for queries and creates a replica of each data segment. You can always recover data from deep storage, even in the unlikely case that all Druid servers fail. For a limited failure that affects only a few Druid servers, automatic rebalancing ensures that queries are still possible and data is still available during system recoveries. When using cloud storage, such as Amazon S3, durability is 99.999999999% or greater per year (or a loss of no more than 1 record per 100 billion).

High Performance

The features of Druid combine to enable high performance at high concurrency by avoiding unneeded work. Pre-aggregated, sorted data avoids moving data across process boundaries or across servers and avoids processing data that isn't needed for a query. Long-running processes avoid the need to start new processes for each query. Using indexes avoids costly reading of the full data set for each query. Acting directly on encoded, compressed data avoids the need to uncompress and decode. Using only the minimum data needed to answer each query avoids moving data from disk to memory and from memory to CPU.

In a paper published at the 22nd International Conference on Business Information Systems (May 2019), José Correia, Carlos Costa, and Maribel Yasmina Santos benchmarked performance of Hive, Presto, and Druid using a TPC-H-derived test of 13 queries run against a denormalized star schema on a cluster of 5 servers, each with an Intel i5 quad-core processor and 32 GB memory.¹³ Druid performance was measured as greater than 98% faster than Hive and greater than 90% faster than Presto in each of six test runs, using

¹³ José Correia, Carlos Costa, and Maribel Yasmina Santos, “Challenging SQL-on-Hadoop Performance with Apache Druid”, RepositoriUM, June 2019.

different configurations and data sizes. For Scale Factor 100 (a 100 GB database), for example, Druid required 3.72s, compared with 90s for Presto and 424s for Hive (Figure 10).

		Time (s)	Difference (%)	
SF 30		Presto	33	---
		Druid (SQuarter_PHashed3)	2.09	-93.7%
		Druid (SQuarter_QMonth_PHashed3)	1.35	-95.9%
SF 100		Presto	90	---
		Druid (SQuarter_PHashed6)	6.12	-93.2%
		Druid (SMonth_QWeek_PHashed2)	3.72	-95.9%
SF 300		Presto	452	---
		Druid (SQuarter_QMonth_PHashed10)	7.6	-98.3%

		Hive			Presto		
Scale factor (SF)		30	100	300	30	100	300
	Star schema	349s	865s	982s	77s	256s	452s
	Denormalized	256s	424s	---	33s	90s	---

Figure 10. Hive, Presto, and Druid results from Correia, Costa, and Santos (2019)¹⁴

In November 2021, Imply published the results of a benchmark using the same star schema benchmark,¹⁵ with Druid on a single AWS c5.9xlarge instance (36 CPU and 72 GB memory) at Scale Factor 100 (a 100 GB database). The 13 queries executed in a total of 0.747s.

¹⁴ Correia, Costa, and Santos, “Challenging SQL-on-Hadoop Performance.”

¹⁵ Eric Tschetter, “Druid Nails Cost Efficiency Challenge Against ClickHouse and Rockset”, Imply, November 22, 2021.

High Concurrency

High concurrency was one of the original design goals for Druid, and many Druid clusters are supporting hundreds of thousands of queries per second.

The key to Druid concurrency is the unique relationship between storage and compute resources. Data is stored in segments, which are scanned in parallel by scatter/gather queries. Usually, scanning each segment requires about 250ms and rarely more than 500ms.

In Druid, there is no need to lock segments, so when multiple queries are trying to scan the same segment, the resources are released to a new query immediately upon scan completion. This keeps the computation time on each segment very small and enables a very high number of concurrent queries.

In addition, Druid automatically caches query results per segment from historical data while not caching, by default, data from fast-changing stream data. This further reduces query times and computation loads.

One of many examples of Druid concurrency was published by Nielsen Marketing,¹⁶ which compared response time as a function of concurrency in Druid with their previous Elasticsearch architecture (Figure 11).

Note that with 10 concurrent queries, average response time was 280ms. Increasing this twelvefold to 120 concurrent queries in Druid only increased the average response time by 18%. The contrast with Elasticsearch is clear.

¹⁶ Danny Ruchman and Itai Yaffe, “Our Journey with Druid: From Initial Research to Full Production Scale”, SlideShare, February 25, 2018.

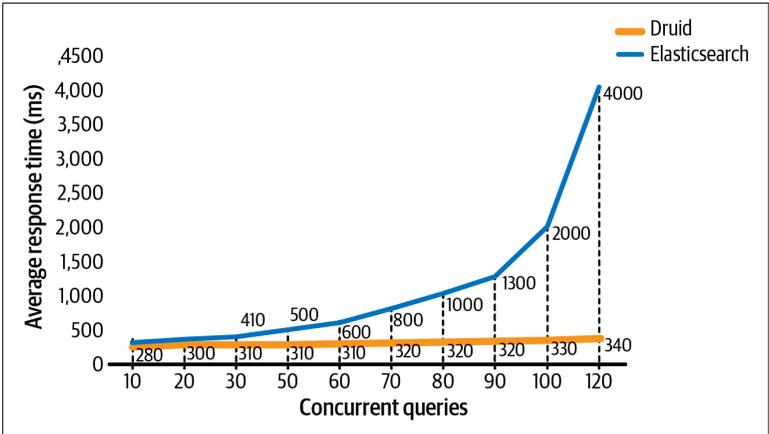


Figure 11. Comparing Druid and Elasticsearch concurrency at Nielsen Marketing

High-Speed Data Ingestion

In Druid, *ingestion*, sometimes called *indexing*, is loading data into the database. Druid reads data from source systems, whether files or streams, and stores the data in segments (Figure 12).

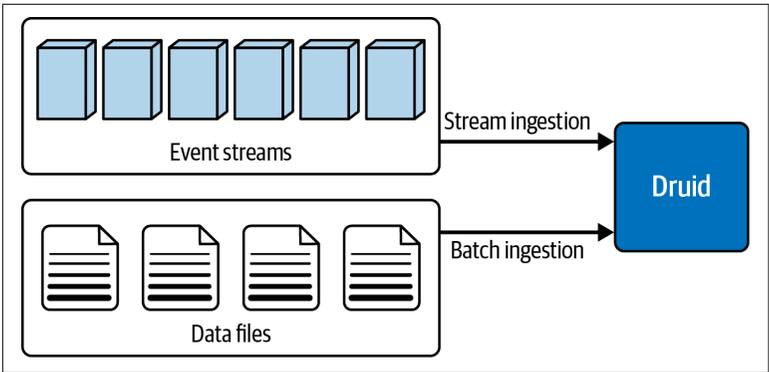


Figure 12. Druid works with both stream ingestion and batch ingestion

The ingestion process both creates tables and loads data into them. All tables are always fully indexed, so there is no need to explicitly create or manage indexes.

When data is ingested into Druid, it is automatically indexed, partitioned, and, optionally, partially pre-aggregated. Compressed

bitmap indexes enable fast filtering and searching across multiple columns. Data is partitioned by time and other fields.

Stream Ingestion

Druid was specifically created to enable real-time analytics of stream data, which begins with stream ingestion. Druid includes built-in indexing services for both Apache Kafka and Amazon Kinesis, so additional stream connectors are not needed.

Stream data is immediately queryable by Druid as each event arrives.

Supervisor processes run on Druid management nodes to manage the ingestion processes, coordinating indexing processes on data nodes that read stream events and guarantee *exactly-once ingestion*. The indexing processes use the partition and offset mechanisms found in Kafka and the shard and sequence mechanisms found in Kinesis.

If there is a failure during stream ingestion, for example, a server or network outage on a data or management node, Druid automatically recovers and continues to ingest every event exactly once, even events that arrive during the outage. No data is lost.

Batch Ingestion

It's often useful to combine real-time stream data with historical data. Batch ingestion is used for loading the latter type of data. This approach to loading data is known as batch ingestion. Some Druid implementations use entirely historical files. Data from any source can take advantage of the interactive data conversations, easy scaling, high performance, high concurrency, and high reliability of Druid.

Druid usually ingests data from object stores—which include HDFS, Amazon S3, Azure Blob, and Google Cloud Storage—or from local storage. The datafiles can be in a number of common formats, including JSON, CSV, TSV, Parquet, ORC, Avro, or Protobuf. Druid can ingest directly from both standard and compressed files, using formats including *.gz*, *.bz2*, *.xz*, *.zip*, *.sz*, and *.zst*.

The easiest way to ingest batch data is with a SQL statement, using `INSERT INTO`. Since this uses SQL, the ingestion can also include

whatever SQL statements are desired to filter, combine, or aggregate the data (WHERE, JOIN, GROUP BY, and others) as part of ingestion.

Creating Real-Time Analytics Applications

Now that you've considered the components needed, it's time to consider other requirements.

Data Design

The less data you store, the faster your queries will run. How can you minimize the database size while maintaining the precision you need?

In Druid, data is ingested from files, tables, and streams and stored in tables. If the original source data is stored, the column is a *dimension*. Many data sets will have high dimensionality, with hundreds or thousands of dimensions.

A useful option to reduce data storage size and improve performance is to use *data rollup*, a pre-aggregation to combine rows that have identical dimensions within an interval that you choose, such as “minute” or “day,” and adding metrics, such as “count” and “total.” This aggregated dimension is a *metric*.

For example, a data stream of website activity that gets an average of 400 hits per second could choose to store every event in its own table row, or it could use a rollup to aggregate the sum of the total number of page hits each minute, reducing the needed data storage by 24,000x. If desired, the disaggregated data can be kept in inexpensive deep storage for infrequent queries where individual events need to be queried.

It's possible to create multiple metrics from the same data set, with aggregations at different levels of granularity to support multiple query requirements.

Metrics can also be created that use stochastic approximation techniques, usually called *sketches* (Figure 13). These enable very fast results, provably within 2% of the precise value, even across very large data sets. Accuracy actually improves as the data set grows! This is often useful for providing subsecond information on the total number of a fast-changing dimension (How many total players

in my game? How many total clicks per second across all of my websites?), where absolute precision isn't needed.

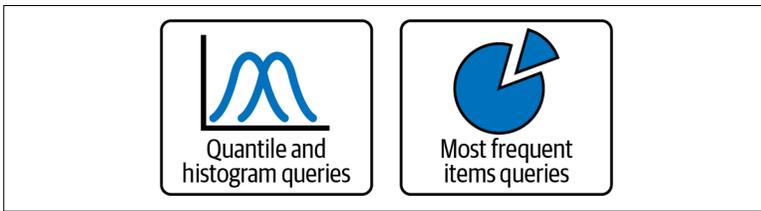


Figure 13. Some of the most common uses of sketches

Unlike most databases, real-time analytics databases don't require the often significant work of modeling the data and creating normalizations, star- and snowflake-schemas, and other designs for organizing data storage. This isn't needed in Druid. Each table is wide and denormalized. Each field value is determined as a dimension or a metric at ingestion.

Interfaces

Most real-time analytics databases use SQL as the primary interface. Either humans or machines can submit queries through APIs to retrieve the data needed.

Druid supports a wide range of programming libraries for development, including Python, R, JavaScript, Clojure, Elixir, Ruby, PHP, Scala, Java, .NET, and Rust. In any of these languages, queries can be executed using either SQL commands (returning text results) or JSON-over-HTTP (returning JSON results).

Since Druid is both high performance and high concurrency, it's a common pattern to use microservices architecture, with many services and many instances of each service able to send queries and receive results without worries about causing bottlenecks for other services.

Humans, though, also like pictures. It's common to use visualization tools with real-time analytics: open source tools like Grafana or Apache Superset, or commercial tools like Tableau, Looker, Power BI, or Domo (Figure 14). Sometimes, though, these tools can have lengthy render times, so queries that return from the database in under a second can still take 30 seconds or more to appear to the user.

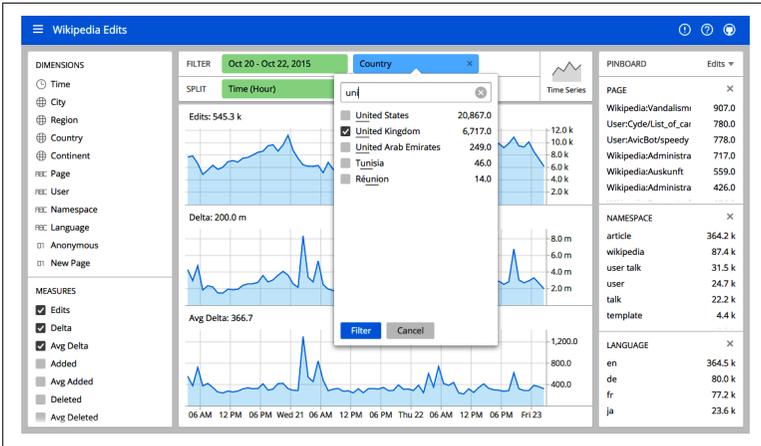


Figure 14. Druid visualization as used by the Wikimedia Foundation

Some visualization tools designed for real-time analytics are becoming available. One example is Pivot, a commercial tool from Imply designed to work with Druid and render visualizations nearly as fast as the database queries can be returned .

Security and Resilience

Any application needs to be secure: data must be protected so only authorized users can view or modify it. Similarly, a real-time analytics application needs to be resilient, without downtime and without data loss.

Security

Druid includes a number of features to ensure that data is always kept secure. Druid administrators can define users with names and passwords managed by Druid, or they can use external authorization systems through LDAP or Active Directory. Fine-grained role-based access control allows each user to query only the data sources (such as tables) to which they have been granted access, whether using APIs or SQL queries (Figure 15).

Users belong to roles that are only able to use resources where the role has been granted permission (Figure 16).

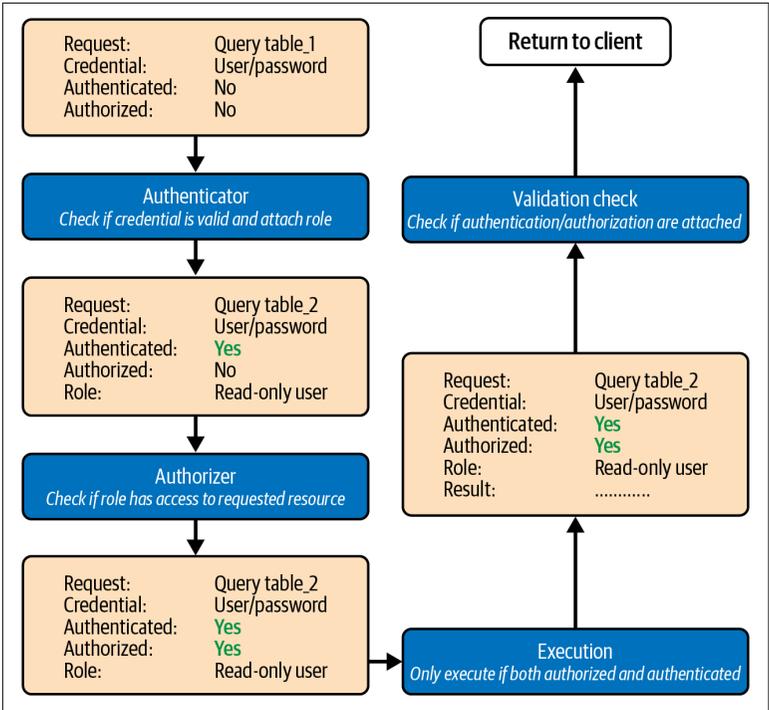


Figure 15. Druid workflow to authenticate and authorize process execution

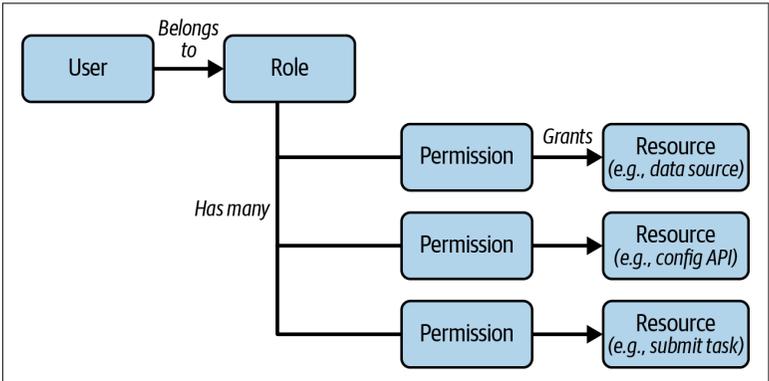


Figure 16. Druid role-based access control hierarchy

All communications between Druid processes are encrypted using HTTPS and TLS. Best practices for production deployment of Druid require clients to use HTTPS to communicate with Druid servers, including the requirement for valid SSL certificates.

Resilience

Druid provides for both very high uptime (reliability) and zero data loss (durability). It's designed for continuous operations, with no need for planned downtime.

A Druid cluster is a collection of processes. The smallest cluster can run all processes on a single server (even a laptop). Larger clusters group processes by function, with each server (“node”) focused on running management, query, or data processes.

When a node becomes unavailable—from a server failure, a network outage, a human error, or any other cause—the workload continues to run on other nodes. During a data node outage, any segments that are uniquely stored on that node are automatically loaded onto other nodes from deep storage. The cluster continues to operate as long as any nodes are available.

Replicas

Whenever data is ingested into Druid, a segment is created and a replica of that segment is created on a different data node. These replicas are used to both improve query performance and provide an extra copy of data for recovery. If desired, for a higher level of performance and resiliency, additional replicas of each segment can be created.

Continuous backup

As each data segment is committed, a copy of the data is written to deep storage, a durable object store. Common options for deep storage are cloud object storage or HDFS. This prevents data loss even if all replicas of a data segment are lost, such as the loss of an entire data center.

It is not necessary to perform traditional backups of a Druid cluster. Deep storage provides an automatic continuous backup of all committed data.

Automated recovery

If all nodes become unavailable (perhaps from a fire, flood, or seismic event damaging a data center or, once again, human error), all data continues to be preserved in both data replicas and durable deep storage. When using cloud object storage such as Amazon S3, Azure Blob, or Google Cloud Storage, multiple copies of data are stored in multiple physically separated data centers in a region, so even destruction of a data center will not cause data loss. Once a Druid cluster is restored, operations can resume with no lost data (Figure 17).

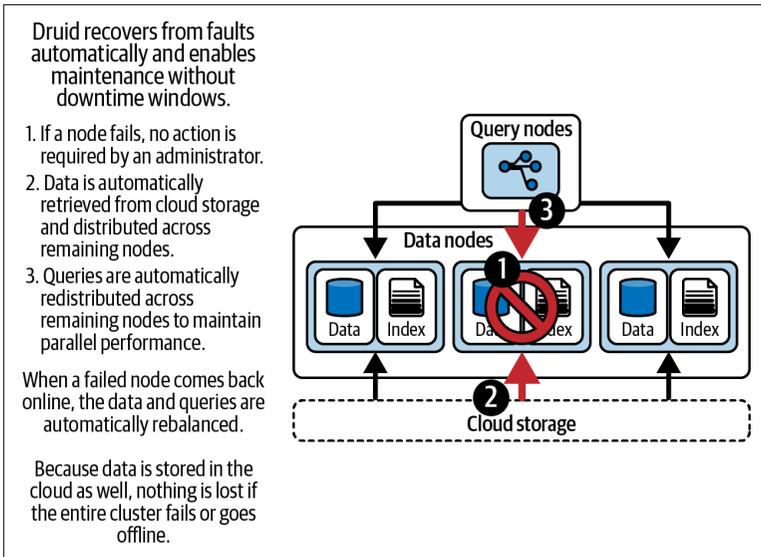


Figure 17. Druid automated recovery provides high availability

Rolling upgrades

Upgrading Druid does not require downtime. Moving to a new version uses a “rolling” upgrade—one node at a time is taken offline, upgraded, and returned to the cluster. Druid can function normally during the rolling upgrade, even though some nodes will be using a different Druid version from other nodes during the upgrade.

Using managed services

A common solution to ensuring the best security and resilience is to hire someone else to do it for you—using a managed service. Many real-time analytics databases are available as managed

services, hosted by cloud service providers. *Imply Polaris*, for example, is a fully managed Druid-as-a-service offering, with all infrastructure managed for security and resilience. Like most managed services, *Polaris* uses consumption-based pricing, so you only pay for the resources you actually use and can very rapidly scale a cluster up or down to meet changing usage requirements.

Managed services are also a great way to begin building and testing new real-time analytics applications, as they require no setup time and usually offer a free trial period to allow you to validate whether the service will work for your project.

Conclusion

If you've read this far, you've seen how real-time analytics applications are driving both insights and action from fast-moving data and how Apache Druid is designed to support real-time analytics applications. Perhaps you've had some ideas about how you can use it yourself.

How will you build a real-time analytics application to drive success for your team? Whether you need interactive conversations with data, subsecond response for queries from large data sets, high-concurrency solutions for difficult-to-predict user communities, a combination of real-time streaming data and historical context data, or high-performance machine learning, the tools you need to make it happen are now available and already being deployed worldwide.

The only limit is your imagination!

Further Resources

As a global open source project, Druid has a strong **community**, with mailing lists, forums, discussion channels, and other resources. You can also find information on Druid at imply.io, with a number of **blog articles** and other useful **resources**.

If you want to try building your own real-time analytics application, a good place to start is a **free trial of *Imply Polaris***, which provides Apache Druid as a service, plus quickstarts for push ingestion, visualization, and more.

Here are some additional resources:

- Fangjin Yang, Eric Tschetter, Xavier Léauté, Nelson Ray, Gian Merlino, and Deep Ganguli, “[Druid: A Real-Time Analytical Data Store](#)”, in *SIGMOD '14: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (ACM, June 2014), 157–168.
- Eric Tschetter, “[Introducing Druid: Real-Time Analytics at a Billion Rows Per Second](#)”, Druid, April 30, 2011.
- José Correia, Carlos Costa, and Maribel Yasmina Santos, “[Challenging SQL-on-Hadoop Performance with Apache Druid](#)”, RepositoriUM, June 2019.
- Eric Tschetter, “[Druid Nails Cost Efficiency Challenge Against ClickHouse and Rockset](#)”, Imply, November 22, 2021.
- [The Powered by Apache Druid web page](#).
- [Apache Druid home page](#).
- [Apache Druid community](#).

About the Author

Darin Briskman is director of technology at Imply, where he helps developers create real-time analytics applications. He began his career at NASA in the 1980s, and has been working with large and interesting data sets ever since. Most recently, he's had various technical and leadership roles at Couchbase, Amazon Web Services, and Snowflake. When he's not writing code, Darin likes to juggle, blow glass, and help children on the autism spectrum learn to use their special abilities to work better with the neuronormative.