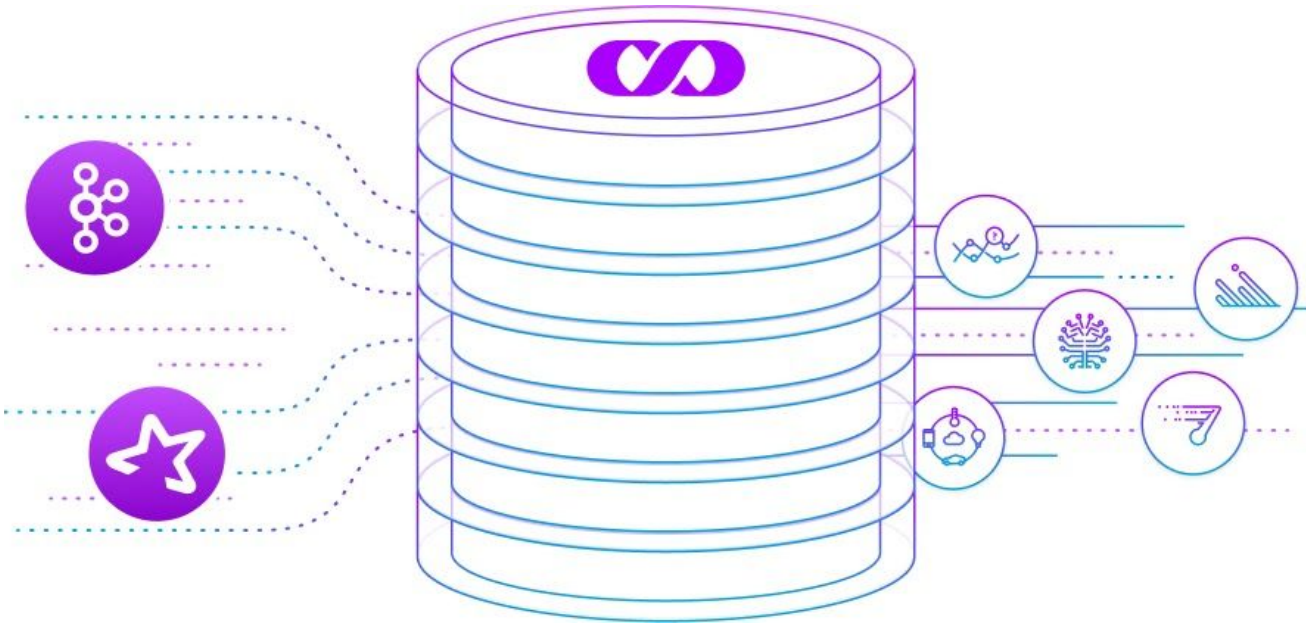
TECHNICAL WHITEPAPER

Real-Time Streaming in Big Data: Kafka and Spark with MemSQL

Nithin Krishna Reghunathan, Technical Evangelist



memSQL

2020

Table of Contents

1. Introduction to Big Data Streaming

Data streaming can be defined as a continuous process of transferring data at a very high rate of speed. The evolution of modern platforms such as the Internet of Things (IoT), cloud computing, the Internet, and social media generate humongous amounts - on the order of petabytes - of real-time data on a daily basis. This data has to be utilized in the most effective way by leveraging the right streaming techniques so that businesses can make data-driven decisions in real time.

The traditional approach of handling data is batch processing, wherein large volumes of data are loaded in batches on a nightly, weekly, or other regular basis. Moving data through in batches causes a significant lag in the availability of recently generated data for data processing. Although batch processing is a common response to limitations in hardware, software, and processes such as the time and effort required to check incoming data for correctness, it doesn't really work well for use cases that demand real-time decision making capabilities. With data streaming, you can meet the demanding requirements of such use cases.

In general, data streaming is ideal for data sources that stream data in small sizes (on the order of kilobytes per second) in a continuous flow as the data is generated. This includes a wide variety of data sources such as the Internet of Things(IoT), connected devices, geospatial applications, web applications, real-time transactions in financial trading/banks, e-commerce platforms, web-based applications, server log data, telemetry or biomedical devices, social media such as Twitter and facebook etc. However, there are more and more applications where users want to stream larger data flows, and these applications challenge the capabilities of even advanced platforms such as Kafka, Spark, and MemSQL.

This streamed data is often used for filtering, sampling, or real-time aggregation and correlation. The derived information from streamed data gives greater visibility for companies to make business decisions that enable them to respond promptly to emerging situations and changes in market conditions.

In this paper, we will discuss in detail two of the leading streaming platforms, Apache Kafka and Apache Spark, and how well they complement the MemSQL database for building real-time data pipelines.

1.1 Challenges in Deploying Real-Time Streaming

Data engineers and architects have to go through numerous challenges prior to successfully deploying a real-time streaming platform. Some of the key challenges include choosing the right development frameworks, non-deterministic delivery of data from connected devices, scaling and performance, algorithm testing, data validation, and life-cycle management.

A streaming data platform requires two layers: a processing layer and a storage layer. The storage layer is primarily responsible for maintaining strong consistency to enable high-speed, replayable reads and writes of large streams of data. The processing layer consumes the data from the storage layer, runs computations on data, and notifies the storage layer to delete the data that is no longer needed. In order to overcome these challenges and ensure successful adoption of a real-time streaming application, the data platform has to adopt a powerful tool that can deliver superior scalability, data persistence, and fault tolerance in both the storage and processing layers.

2. Real-Time Data Pipelines for Streaming Applications

The following layers play a major role in building a real-time data pipeline for streaming applications: (i) A high-throughput messaging layer, (ii) A data transformation layer, and (iii) a persistent data store/database.

Fig 1. Real-time data pipelines with MemSQL

2.1 Kafka: A Distributed Streaming Platform for High-Throughput Messaging

With the evolution of real-time streaming, high-throughput messaging systems have become an integral part of enterprise IT infrastructure. As enterprise companies start putting more effort into building real-time data pipelines, high-performance messaging systems (fast, scalable solutions) have gained momentum, especially for use in real-time analytics.

Modern, real-time data pipelines rely on high-throughput messaging systems to capture data from a wide range of sources, including IoT-connected devices, financial trades, the stock market, mobile applications, web applications, etc. This high-throughput messaging system ensures that every data file is transferred to the destination with zero data loss. The data is then written into databases for further processing.

Apache Kafka is a distributed streaming platform used for high-throughput messaging applications. Kafka is used for building real-time data pipelines and streaming applications. It is horizontally scalable, fault-tolerant, and wicked fast. It runs in production across all industries. Kafka acts as a broker between *producers* (processes that publish their records to a communications channel, which is called a topic) and *consumers* (processes that subscribe to one or more topics). Kafka is capable of handling terabytes of messages without any loss of performance, which is the reason it's become the first solution considered for building streaming applications.

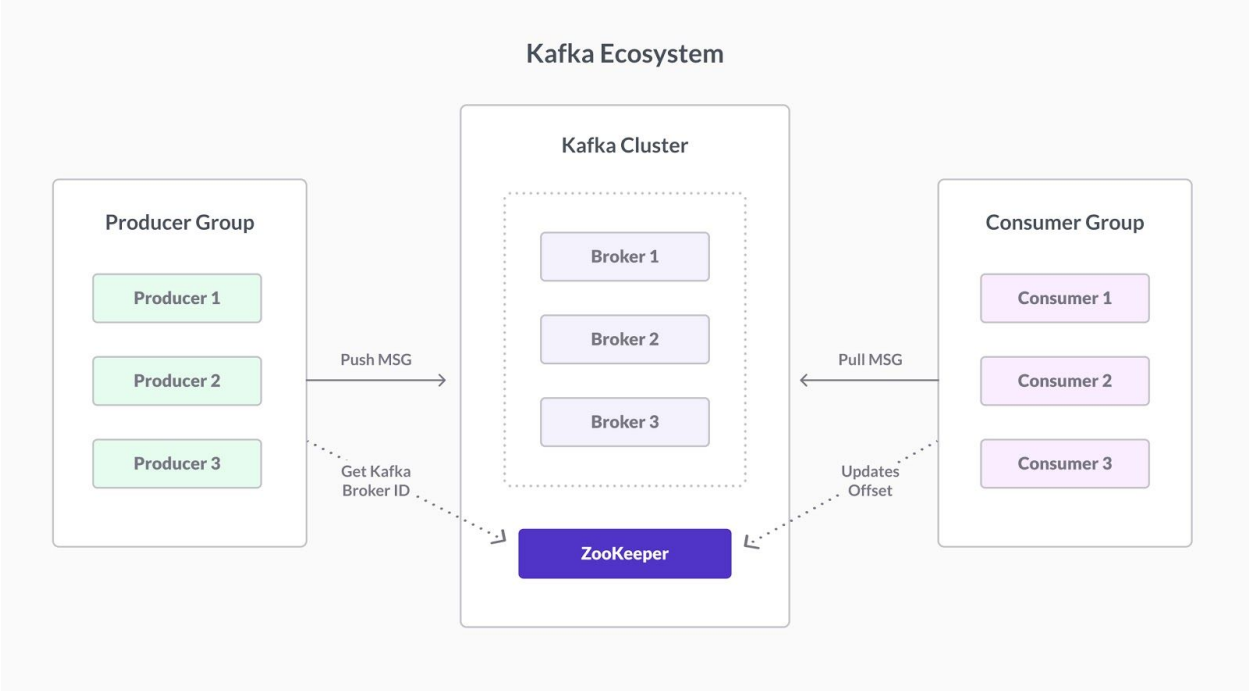


Fig 2. Apache Kafka ecosystem

Apache Kafka is primarily built with a distributed approach, so also to allow both the producers and the consumers to scale out horizontally - that is, by adding more servers to the cluster. In addition, Kafka's efficient use of memory, along with distributed commit logs stored on disk, ensures very high durability - as the messages persist on to disk as fast as possible. These characteristics enable Kafka to deliver superior performance for real-time applications.

More details on the fundamental concepts of Apache Kafka can be found at the official Apache site for Kafka.

2.2 Spark: Streaming Processing Platform for Data Transformation

In a streaming application, the data transformation layer converts the raw data into a meaningful format which is more usable for analytics and data exploration. The transformation layer can do various other tasks, including such as filtering of data, data aggregation, and data enrichment.

Apache Spark is one of the most popular stream processing platforms, designed for data transformation and used across all industries. Spark is a distributed, memory-optimized cluster computing framework that can add a great deal of value to real-time streaming applications. Spark Streaming brings Apache Spark's language-integrated API to stream processing, letting you write streaming jobs the same way you write batch jobs. It supports several popular programming languages: Java, Scala, and Python. Spark also has a native streaming library and programming interfaces that makes data processing and transformation easier.

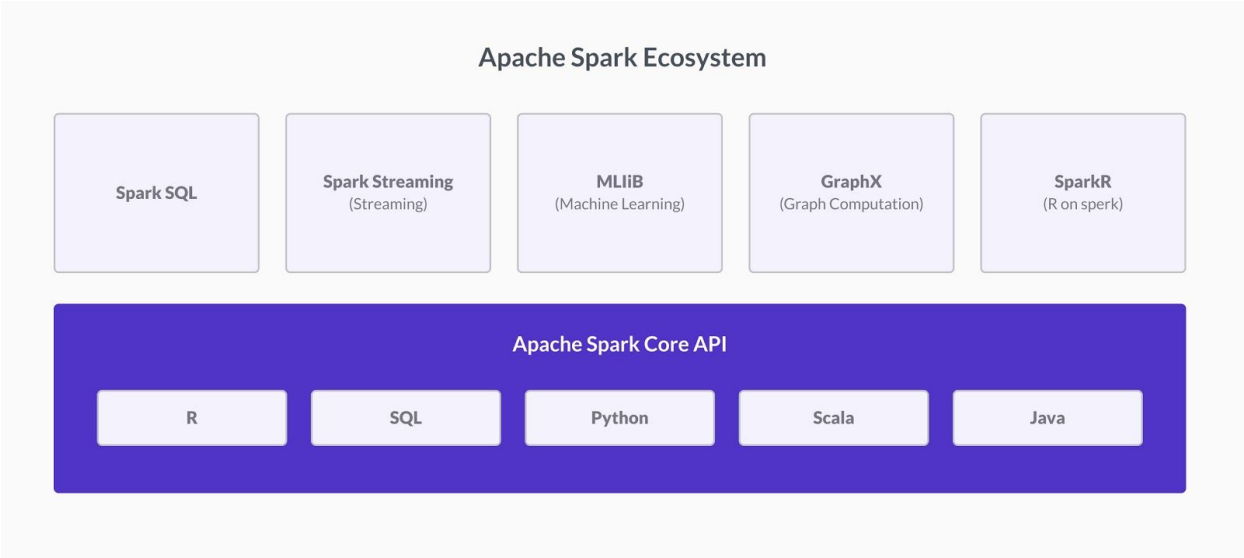


Fig 3. Apache Spark ecosystem

Spark streaming lets you leverage the same code you use for batch processing, join streams of newer data against historical data, or run ad hoc queries against the stream state. You can build scalable, fault-tolerant, interactive streaming applications with the Spark platform. While building real-time data pipelines, Spark does the transformation part by extracting raw data from sources (such as Kafka), running machine learning models or transforms on raw data, and then moving those enriched or augmented datasets to a persistent database (or datastore) for further analysis.

One fairly widely-used feature of Spark which we should comment on here is Spark SQL, a query language that you can use to query Spark streams. This query language is not ANSI SQL, but instead is SQL-like. It doesn't do much of what true SQL does, nor does it benefit from the decades of performance optimization that relational databases, which generally support true SQL, inherit. Later in this whitepaper, we will describe SQL pushdown, which allows the MemSQL database to efficiently execute commands that would otherwise be run, more slowly and with more contention with other processes, in Spark SQL.

More details on the fundamental concepts of Apache Spark can be found at the official Apache site for Spark.

2.3 Persistent Database or Datastore

Real-time data pipelines for streaming applications demand persistent database technologies that are capable of handling high-throughput data capture, ingestion, and processing. Some of the key characteristics shared by the streaming technologies used for real-time applications include: (i) memory-optimized store for ultra-fast ingest, (ii) distributed architecture capable of scaling out horizontally, (iii) real-time analytics capability, and (iv) massively parallel processing capability.

Neither the Kafka platform nor the Spark platform offers any persistent storage engine (e.g. database or datastore), which is why there arises a need for an operational database which is capable of storing data in the form of objects or records, with very high durability (meaning a very low likelihood of permanent data loss). An operational database must ensure the following key things to empower any real-time application: (i) *persistence* - saving all its information to disk storage for durability, (ii) *high availability* - data is always highly available by maintaining a readily available copy of all data, (iii) *fast failover* - fast and automatic switching to the copy, without downtime, in case of any unexpected hardware failures. A persistent database in a real-time data pipeline can be considered as a permanent datastore from where we can access data instantly for a variety of use cases such as real-time analytics, historical analysis, etc.

A memory-optimized operational database can manage and store real-time data in the most efficient way. Moreover, it enables persistence for both real-time and historical data, and the ability to query both within a single system. The data from the transformation

layer can be rapidly ingested into a persistent database (or datastore) which is used as a resource for deriving real-time insights on rapidly moving data.

3. Building Real-Time Data Pipelines with MemSQL

3.1 MemSQL Overview

MemSQL is an operational database built for performing both transactions and analytics to support the demands of modern applications, analytical systems, and AI and machine learning at scale. MemSQL uses a cloud-native, distributed architecture to deliver maximum performance and elastic scale. Note that "cloud-native" does not mean "cloud-only"; in fact, "cloud-native" infrastructure and apps are completely flexible, being able to run on any cloud or on-premises. MemSQL accomplishes this by offering both multi-cloud and hybrid options, ranging from a database-as-a-service (MemSQL Helios), to Kubernetes-based hybrid and private deployments (using the MemSQL Kubernetes Operator), to traditional on-premises installations on virtual machines (VMs) or commodity hardware.

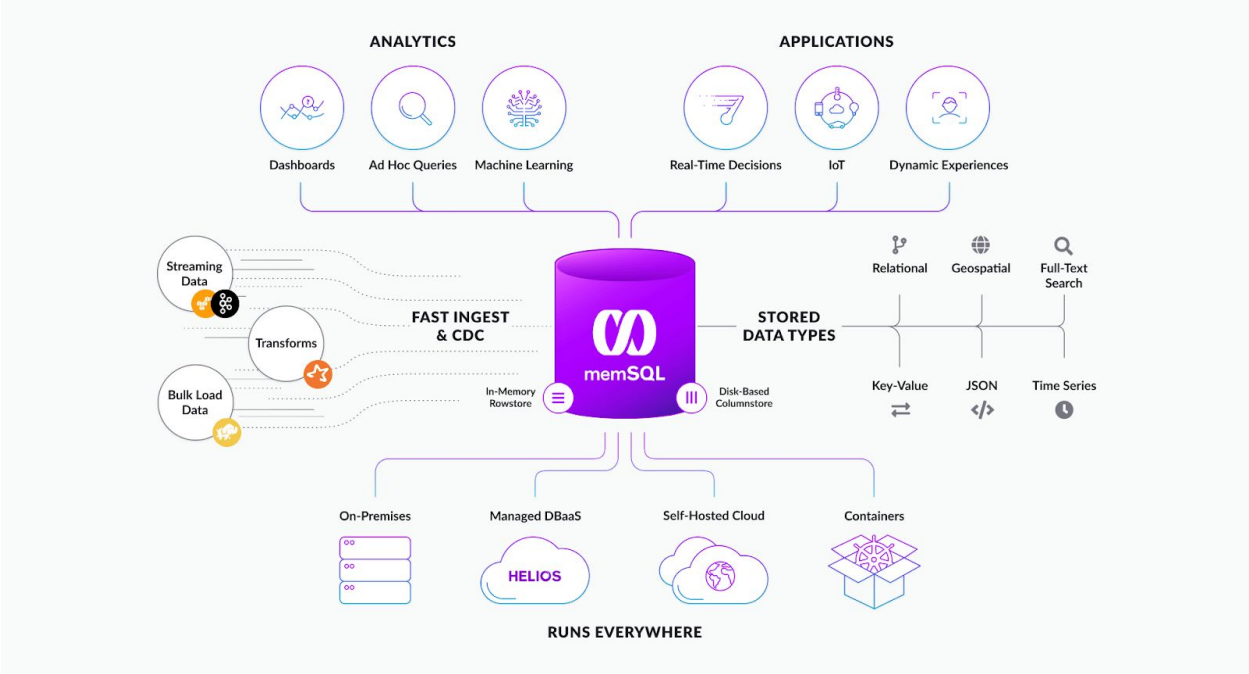


Fig 4. MemSQL Architecture

MemSQL can ingest millions of events per second, with support for ACID transactions, while simultaneously supporting ad hoc SQL queries, business intelligence (BI) tools, , applications, machine learning model queries, and AI queries on trillions[1] of data rows. MemSQL can support running transactional and analytical workloads with fast ingest and while supporting high concurrency, all while supporting the standard ANSI SQL query language. You can read this technical whitepaper for a deep dive into the concepts behind the MemSQL data platform.

MemSQL's database-as-a-service offering is called MemSQL Helios. MemSQL Helios gives you the full capabilities of self-managed MemSQL database software without the operational overhead and complexity of managing it yourself. MemSQL Helios provides a resilient database with cloud-agnostic deployment support on AWS and Google Cloud Platform (with support for Azure coming soon, and others to follow).

[1] MemSQL Processing Shatters Trillion Rows Per Second Barrier
https://www.memsql.com/blog/memsql-processing-shatters-trillion-rows-per-second-barrier/

With MemSQL Helios, cluster provisioning, cluster management, deployment, upgrades, alerting, and troubleshooting are all handled by MemSQL. This greatly reduces operational expenses, by shifting the database administration (DBA) tasks needed to operate your database from your organization to MemSQL. Support interactions are also simplified, and, for those questions which do arise, solving the problem is likely to be faster and easier than with self-managed MemSQL software.

Just as importantly, MemSQL is offered at a price point dramatically lower than traditional database vendors, while our ultra-efficient query engine means that operational costs for MemSQL also tend to be lower than the proprietary offerings from the cloud service providers - including open source offerings, for which the license required to run the software is "free."

3.2 How MemSQL Keeps Your Data Safe

MemSQL is built with enterprise-ready durability features. The MemSQL architecture enables organizations to keep their data safe without introducing latency. MemSQL has three mechanisms for ensuring not only disk-based durability, but that your database and business stay online: logs and snapshots to disk for durability, redundancy for high availability, and cross-datacenter replication for disaster recovery.

3.2.1 Persistence, Data Durability, and High Availability in MemSQL

An operational database must have reliable persistence and high availability mechanisms for the data. MemSQL ensures this persistence with its ability to store information durably and resilient to any unexpected hardware failures.

MemSQL runs with full *durability* enabled. The transactions are committed to the transaction log on disk and later compressed into full-database snapshots. If data needs to be restored, the database software restores the most recent snapshot, then plays back the

logged transactions that occurred since that snapshot. This process is quite fast and does not cause noticeable downtime. Hence, durability is ensured through persistent logs and periodic snapshots.

MemSQL ensures *high availability* by storing a redundant copy of data within a cluster. The paired leaf nodes replicate data to one another, and can be configured to do so as synchronous replication - which is the safer method, and only adds a small performance hit - or as asynchronous replication, which is even faster. When a leaf node goes offline, MemSQL automatically fails over to its replication partner.

MemSQL supports fully automatic *cross-datacenter replication* that can be provisioned with a single command. The secondary (replica) cluster stores a read-only copy of data asynchronously replicated from the primary cluster. In the event of a major failure in the primary cluster, MemSQL allows you to promote the secondary cluster, immediately making it a "full" MemSQL cluster. In addition to providing disaster recovery assurance, the secondary cluster can also be used to support heavy read-only workloads, which enables higher performance for reads, which take place against a copy of the cluster that is not actively being written to, and for writes - the "live" cluster, against which writes occur, is made less busy by offloading many reads to the secondary cluster.

3.2.2 System of Record Capabilities in MemSQL 7.0 (and above)

System of record capability is the holy grail for transactional databases. Companies need to run their most trusted workloads on a database that has many ways to ensure that transactions are completed and to back up completed transactions, with fast and efficient restore capability. MemSQL 7.0 (and above versions) includes new features that deliver very fast synchronous replication – including a second copy which is made as part of the initial write operation, atomically – and incremental backup, which offers increased flexibility and reliability.

With these features, MemSQL 7.0 offers a viable alternative for Tier 1 workloads that require a system of record capability. When combined with the speed improvements and operational efficiencies provided by MemSQL SingleStore, and MemSQL's long-standing ability to combine transactions and analytics on the same database software, MemSQL 7.0 now offers unprecedented design and operational simplicity, lower costs, and higher performance for a wide range of workloads.

3.3 Building Kafka Pipelines with MemSQL

MemSQL Pipelines is a built-in component of MemSQL database software. Pipelines can extract, transform, and load external data, without the need for third-party tools or middleware. MemSQL Pipelines is a native feature that natively ingests real-time data from external sources. The Pipelines feature is robust, scalable, highly performant, and supports fully distributed workloads.

MemSQL Pipelines can extract data directly from Apache Kafka. In addition, it can also extract data from Amazon S3, Azure Blob, Filesystem, Google Cloud Storage, and HDFS data sources.

Kafka and MemSQL share a similar distributed architecture that makes Kafka an ideal data source for building real-time or near real-time pipelines with MemSQL. Kafka topics can be paired one to one with MemSQL leaf nodes for very fast ingest, processing, and availability of both new and historical data for analytics and apps.

Both Kafka and MemSQL support "exactly-once" updating, which ensures that data, once placed in a Kafka topic, then processed by MemSQL, is neither lost, nor sent through or processed more than once. MemSQL Pipelines natively support not only relational data, but also the JSON, Avro, and CSV data formats, as does Kafka.

To create and interact with a Kafka pipeline quickly, using the MemSQL Pipelines feature, follow the instructions in this link.

3.4 Advantages of the MemSQL Spark 3.0 Connector

Note: The MemSQL Spark Connector 3.0 is a beta release to be used for testing and development, and is not intended for production use. The GA version will be available in the near future.

The MemSQL Spark Connector 3.0 allows you to connect your MemSQL database to Spark stream processing. Spark excels at iterative computation and includes numerous libraries for statistical analysis, graph computations, and machine learning. However, Spark SQL, as mentioned above, is not true ANSI SQL, and is not nearly as fast or efficient at query processing as MemSQL.

There are two key data bottlenecks in Spark-oriented processing against other data layers that the combination of the MemSQL database and the MemSQL Spark Connector 3.0 solves: (1) slow query responses and (2) slow data loads.

> To address the first bottle neck: slow query responses, MemSQL Spark Connector supports SQL pushdown - a feature which allows queries in Spark SQL to be processed as native SQL queries in the MemSQL database, with great gains in speed, and less use of system resources. As datasets grow in AI/ML scenarios, the query execution efficiency and query optimization that MemSQL provides grow in importance. Additionally, an order of magnitude performance gain is achieved when the SQL queries being pushed down utilize MemSQL's memory-optimized columnstore with compressed data and segment elimination.

The efficiency of your Spark SQL for ML training operations can be measured by viewing the query plan used by invoking `train.explain()` from your Spark shell. This command shows the Spark execution tree for the query plan identifies where SQL pushdown operations are occurring for each part of your Spark SQL statement. Also, the MemSQL Spark Connector improves query performance and query optimization on every Spark SQL statement. This is especially important in multi-pass Spark SQL operations where a series of SQL statements are executed. The performance gains provided by MemSQL for machine learning algorithms which perform multi-pass data querying can be as much as 100x faster. This performance advantage compounds with the number of passes of the algorithm.

The MemSQL Spark Connector can also address the second data bottleneck: slow data loads. By taking advantage of the fact that both Spark and MemSQL are distributed, the Spark Connector makes multiple, parallel connections to MemSQL's multiple data nodes, called Leafs, to read data which vastly improves throughput for MLOps scenarios. This allows your distributed Spark cluster to execute parallel reads directly against each of the distributed data nodes in the MemSQL cluster. This provides another order of magnitude performance increase.

You can leverage MemSQL and Spark together to accelerate workloads by taking advantage of computational power of Spark in tandem with the fast ingestion, persistent storage, fast query processing, and high concurrency for queries from multiple sources that MemSQL has to offer.

The Spark connector is implemented as a native Spark SQL plugin and supports Spark's DataSource API. The Spark connector makes it possible to integrate Spark with MemSQL for a wide range of use cases, such as a real-time data analytics pipeline, scoring machine

learning models, predictive analytics etc. The MemSQL Spark Connector 3.0 supports both data loading and extraction from database tables and Spark DataFrames.

Please check this link to learn more about - how to configure and start using the MemSQL Spark Connector 3.0.

You can download the Spark Connector 3.0 from its GitHub repository and from Maven Central. The group is `com.memsql` and the artifact is `memsql-spark-connector_2.11`.

Finally, check out the Spark Connector videos on MemSQL's YouTube Channel.

4. Customer Success Stories

4.1 Large Energy Company in US Using MemSQL for Preventative Maintenance

Use Case

A large energy company uses MemSQL for upstream processing, which is a solid use case of predictive analytics in a machine learning application. The company delivers preventive maintenance for their oil drills across the globe, including expensive drill bits, using this data platform. The preventive maintenance approach helps them to reduce failures and avoid breakdowns to their equipment.

Problem Statement

Design a highly scalable data platform which is capable of continuously ingesting raw data from a variety of sensors and which allows predictive machine models to run in parallel to determine the scoring for oil drill health, all in real time.

Key Considerations for Using MemSQL

- Can query while ingesting data
- Highly scalable
- Ultra-fast ingestion
- Massively parallel processing data architecture
- Performs ML scoring and data ingestion in parallel

Reference Architecture for Predictive Maintenance

Oil drills are deployed across different locations around the globe. These drills are equipped with various sensors (such as vibration, direction, and temperature) and generate data continuously. The raw data generated gets pushed into a Kafka queue as a first step. Data is then pulled from the Kafka queue into a Spark cluster, where a Predictive Model Markup Language (PMML) model calculates the health of a drill based on real-time data. The scored data then lands in MemSQL to power a real-time operational dashboard, where the drill operators can make informed judgments based on events or anomalies detected by processing real-time data.

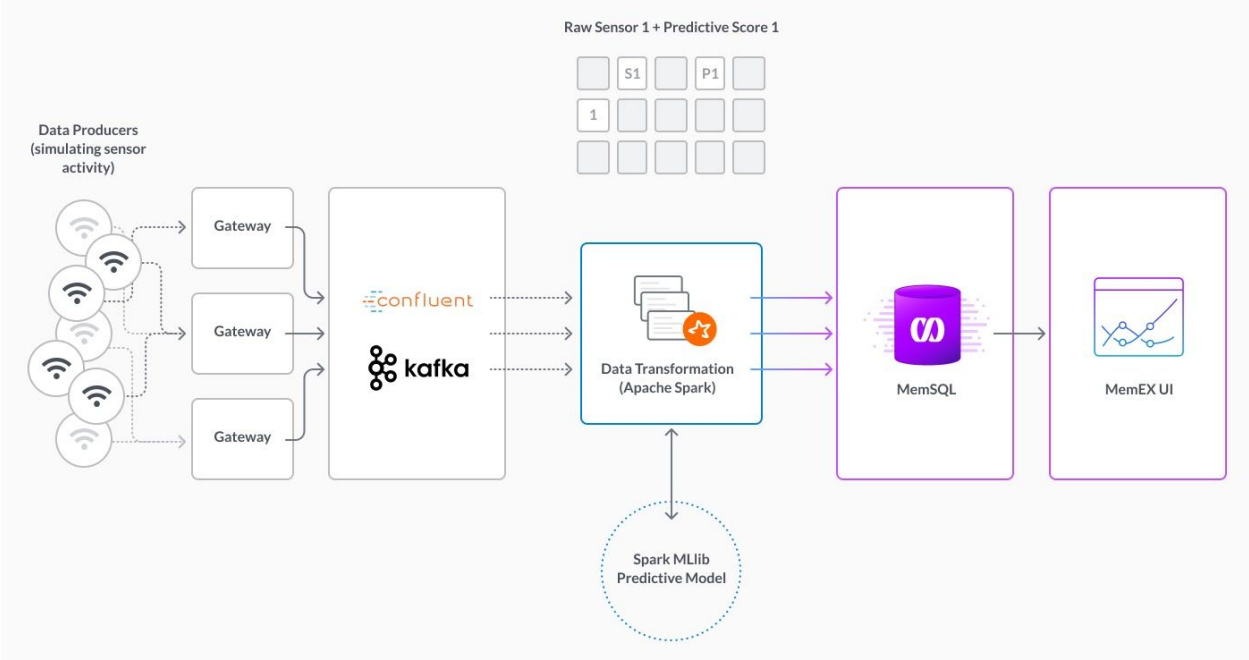


Figure 5. Reference Architecture for Predictive Maintenance

Positive Business Outcomes and Benefits with MemSQL

The following benefits and positive business outcomes are achieved by using MemSQL as part of the solution.

- **Maintain Equipment Uptime.** Analyze millions of equipment condition readings and event updates, with real-time monitoring
- **Reduced Operating Costs.** Analyze millions of equipment conditions and events with real-time monitoring
- **Predict Failures.** Prevent high-risk conditions with real-time predictive scoring
- **Real-Time Visibility.** Each department/region now has the ability to assess current business and oil production to forecast budgets from their computers or mobile devices - at any time of the day
- **Strong earnings and profitability.** as shown in their market capitalization and stock price

4.2 Major Financial Service Provider Improving Risk Management Performance with MemSQL

Use Case

In this customer success story, we describe how one of the major financial services providers in the US was able to successfully improve the performance and ease of development of their risk management decision-making system with MemSQL.

Problem Statement

Design a solution to meet the growing concurrency and performance requirements for their risk management decision-making system.

Key Considerations for Using MemSQL

The following considerations helped to drive the use of MemSQL as part of the solution:

- **Scalable.** The underlying database used must be highly scalable, so as to provide arbitrarily large capacity wherever needed.
- **Streaming-ready.** Ability to integrate easily with streaming platforms like Kafka and process this high-velocity data in a smart and efficient way.
- **Flexible.** Ability to deploy in any public cloud, on-premises, in a container or virtual machine, or in a blended environment, so as to mix and match strengths as needed to meet requirements.
- **High concurrency.** The database powering analytics must support a large number of simultaneous users, with good responsiveness for all.
- **SQL compatibility.** Scores of popular business intelligence tools involved in the decision-making systems use SQL.
- **Real-time analytics.** Ability to operate on rapidly moving data ("query while you ingest") is crucial for the decision-making system.

Reference Architecture

In order to meet the business requirements, the company implemented the newly simplified architecture (Fig. 5) by leveraging the following modern solutions:

- **Kafka as a high-throughput messaging platform.** The company standardized on Kafka for messaging, speeding data flows and simplifying operations.
- **Analytics database consolidation to MemSQL.** A single data store running on MemSQL was chosen as the engine and source of truth for analytics.
- **Spark as a data transformation layer**. Data scientists leverage the power of MemSQL and Spark together for data exploration.

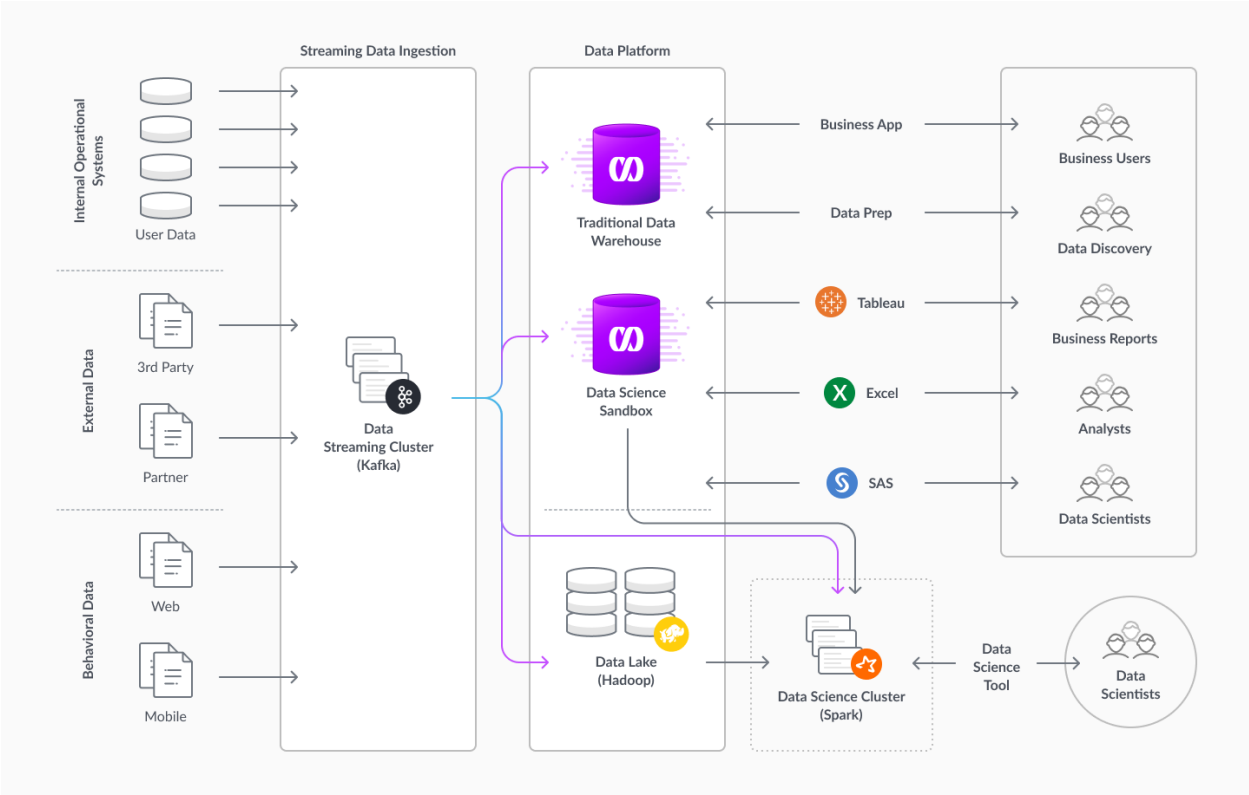


Fig 6. Reference architecture for risk management

Before MemSQL, multiple (ETL) processes moved data between the operational database and the analytics data warehouse, which added to the overall data latency and complexity. Other ETL processes are also typically used to load additional data sources into the data warehouse, adding more delays. After implementing MemSQL, the Kafka cluster streams all the data into one MemSQL database instance for analytics, and also into a second database instance which serves as a performant data science sandbox. The Hadoop/HDFS data lake, serving as long-term cold storage, retains all the data. This Hadoop data lake was a dedicated platform for data science-related activities on cold data, with data scientists being the primary users.

With MemSQL, the customer was able to load data in as soon as it became available, rather than doing batch uploads on a nightly basis (the “before MemSQL” approach).

Using MemSQL led to better query performance, with query results that include the latest data, not only historical data, as had been the case previously. The customer has achieved greater performance, better uptime, and simpler application development. Risk managers have access to much more recent data. Also, users were able to achieve sub-second response time for complex analytic queries. The risk management users, analytics team, and data scientists shared a wide range of benefits after implementing MemSQL.

Positive Business Outcomes and Benefits with MemSQL

The following benefits and positive business outcomes accrued from using MemSQL:

- Dramatic cost savings due to the need for fewer servers and compute cores, and less RAM
- Less coding and less complexity for new apps
- Lower TCO
- Cloud connectivity and flexibility
- More analytics users supported
- Ultra-fast analytics on streaming data
- Augmented data science results

Please refer to this detailed case study link to learn more about this customer success story.

5. Conclusion

Modern organizations have started adopting a data-driven approach to improve their business performance and decision-making capabilities while achieving enhanced customer experience. Agile businesses need to implement real-time data pipelines so decision makers can refine strategies quickly. Streaming real-time data pipelines can play a major role in achieving this goal by providing instant access to analytics, run against

real-time data. An operational database connected to a real-time data pipeline must be capable of delivering real-time insights and experiences through operational analytics.

The reference architectures mentioned in the previous chapters refers to a widely accepted approach to build real-time data pipelines across all the industries. The most common data pipeline approach followed by our customers includes: (i) Kafka - as a high-throughput message broker, (ii) Spark - as a transformation layer that can process and enrich the data in micro batches, and (iii) MemSQL - as an operational database that can deliver ultra-fast and highly scalable data reporting and analytics across all of your operational data, to include streaming real-time data and historical data.

The customer stories discussed showcase how well our customers are leveraging MemSQL with modern streaming technologies like Kafka and Spark for successfully building real-time data pipelines. Adopting a data-driven approach with MemSQL can drive successful business performance, which is something enterprises are constantly striving to achieve.

- Download and experience our on-premises free trial version at https://www.memsql.com/download/
- With MemSQL Helios (database-as-a-service) now publicly available, we hope that you can experience it for yourself, and share your success story. Test drive MemSQL Helios at memsql.com/helios