Announcing Apache Pinot 1.0™

By: Hubert Dulay, Mayank Shrivastava, Neha Pawar

September 19th, 2023 • 13 min read

What Makes a “1.0 Release?”

Apache Pinot has continuously evolved since the project’s inception within LinkedIn in 2013. Back then it was developed at a single company with a single use case in mind: to power “who viewed my profile?” Over the ensuing decade the Apache Pinot community expanded to be embraced by many other organizations, and those organizations have expanded its capabilities to address new use cases. Apache Pinot in 2023 is continuously evolving to address emerging needs in the real-time analytics community. Let’s look at how much innovation has gone into Apache Pinot over the years:

- Upserts — data-in-motion tends to stay in motion, and one of the cornerstone capabilities of Apache Pinot is upsert support to handle upsert mutations in real-time.
- Query-time Native JOINs — it was important to get this right, so that they were performant and scalable, allowing high QPS. This we will discuss in more detail below.
- Pluggable architecture — a broad user base requires the ability to extend the database with new customizable index types, routing strategies and storage options
- Handling Semi-Structured/Unstructured Data — Pinot can easily index JSON and text data types at scale.

- Improving ANSI SQL Compliance — to that end, we've added better NULL handling, window functions, and as stated above, the capability for native JOINs.

With all of these features and capabilities, Apache Pinot moves farther and farther from mere database status, and becomes more of a complete platform that can tackle entire new classes of use cases that were beyond its capabilities in earlier days.

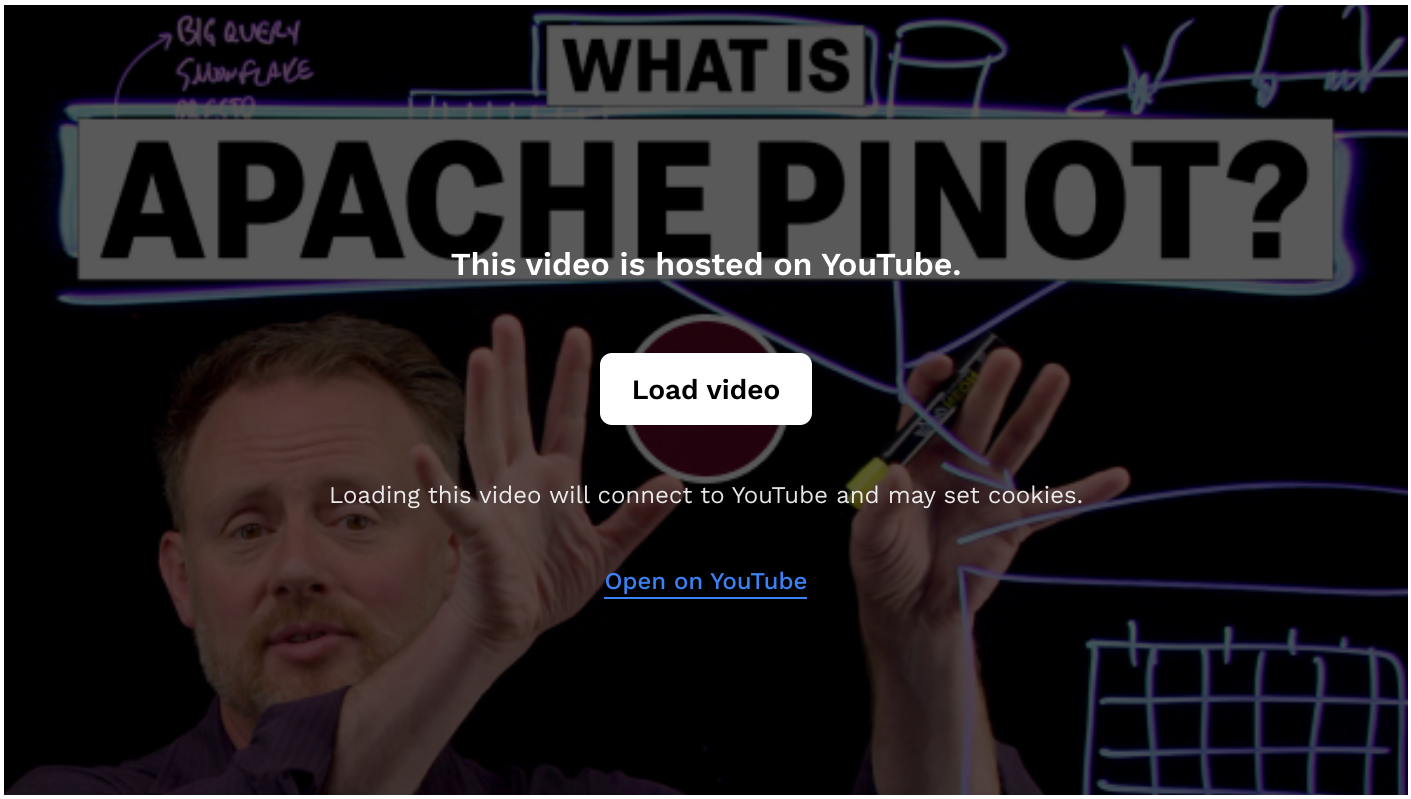
First let's look at what Apache Pinot 1.0 itself is delivering. The first foundational pillar of what makes something worthy of a "1.0" release is software quality. Over the past year, since September 2022, engineers across the Apache Pinot community have closed over 300 issues to provide new features, optimize performance, expand test coverage, and squash bugs.

Features are also a key thing that makes a new release worthy of "1.0" status. The most critical part of the 1.0 release is undoubtedly the Multi-Stage Query Engine, which permits Apache Pinot users to do performant and scalable query-time JOINs.

The original engine works very well for simpler filter-and-aggregate queries, but the broker could become a bottleneck for more complex queries. The new engine also resolves this by introducing intermediary compute stages on the query servers, and brings Apache Pinot closer to full ANSI SQL semantics. While this query engine has been available within Apache Pinot already (since release 0.11.0), with the release of Apache Pinot 1.0 this feature is functionally complete.

(While you can read more below, check out the accompanying blog by Apache Pinot PMC Neha Pawar about using query-time JOINs here).

This post is a summary of the high points, but you can find a full list of everything included in the release notes. And if you'd like a video treatment of many of the main features in 1.0, including some helpful animations, watch here:



Otherwise, let’s have a look at some of the highlighted changes:

- Join Support - Part of the Multi-Stage Query Engine
- Improved Upserts - Deletion and Compaction Support
- Encode User-Specified Compressed Log Processor (CLP) During Ingestion
- NULL Support
- Pluggable Index Types [Index Service Provider Interface (SPI)]
- Improved Pinot-Spark Integration - Spark3 Compatibility

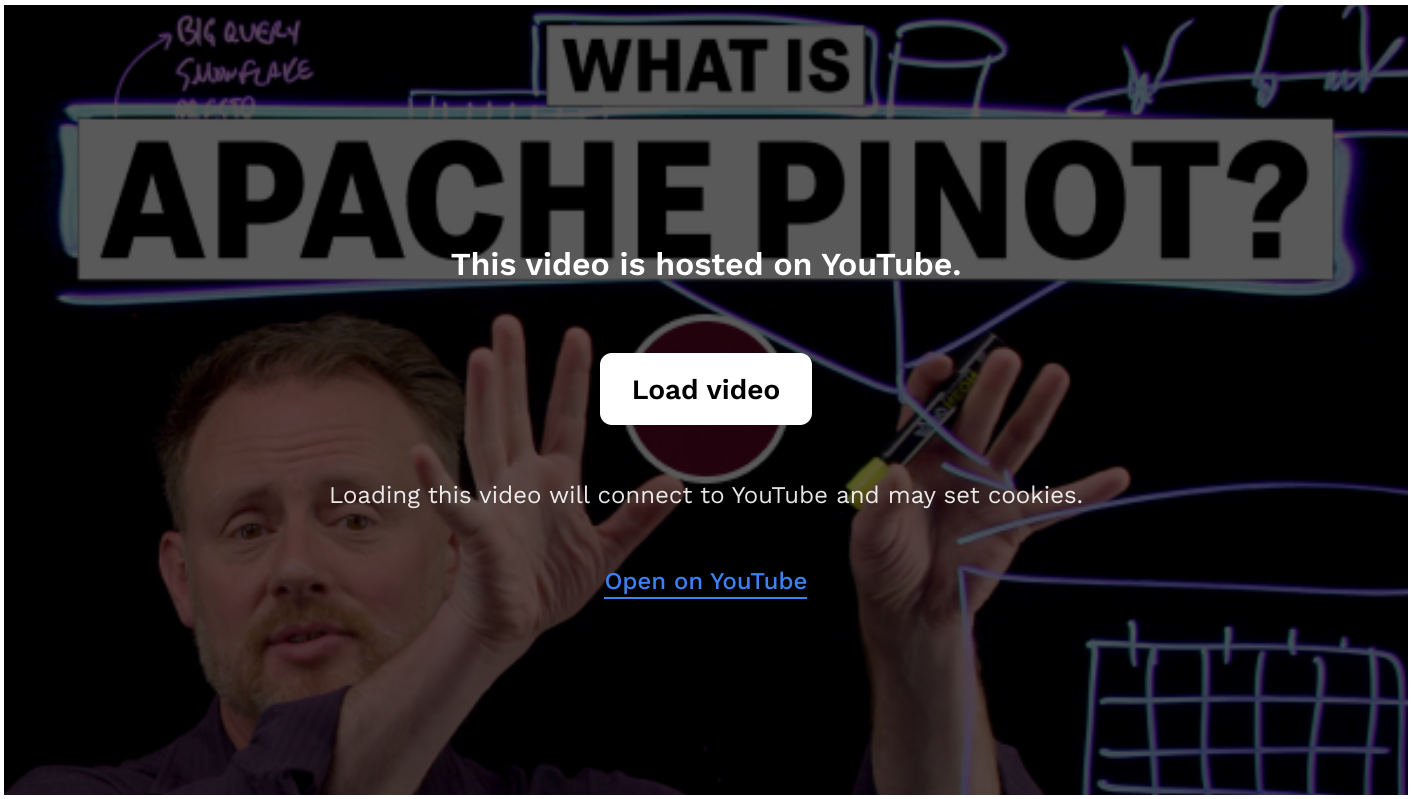
Join Support

Apache Pinot 1.0 introduces native query-time JOIN support equipping Pinot to handle a broad spectrum of JOIN scenarios providing full coverage from user-facing analytics all the way up to ad hoc analytics. Underpinning this innovation is the multi-stage query engine, introduced a year ago, which efficiently manages complex analytical queries, including JOIN operations. This engine alleviates computational burdens by offloading tasks from brokers to a dedicated intermediate compute stage. Additionally, a new planner supporting full SQL semantics enhances Pinot's analytical capabilities.

JOIN optimization strategies play a pivotal role in Apache Pinot 1.0. These include predicate push-down to individual tables and using indexing and pruning to reduce scanning which speeds up query processing, smart data layout considerations to minimize data shuffling, and query hints for fine-tuning JOIN operations. With support for all JOIN types and three JOIN algorithms, including broadcast join, shuffle distributed hash join, and lookup join, Apache Pinot delivers versatility and scalability. By significantly reducing query latency and simplifying architecture, Apache Pinot 1.0 is a game-changer for real-time OLAP systems.

For more detailed information on JOINs, please visit this blog post.

Discover How Uber is using Joins in Apache Pinot For a real-world use case, Uber is already using the new join capabilities of Apache Pinot at scale in production. You can watch this video to learn more.



Upsert Improvements

Support for upserts is one of the key capabilities Apache Pinot offers that differentiates it from other real-time analytics databases. It is a vital feature when real-time streaming data is prone to frequent updates. While upserts have been available in Apache Pinot

since 0.6.0, with 1.0 they include two major new enhancements: segment compaction and delete support for upsert tables.

Segment Compaction for Upsert Tables

Pinot's Upsert tables store all versions of a record ingested into immutable segments on disk. Older records unnecessarily consume valuable storage space when they're no longer used in query results. Pinot's Segment Compaction reclaims this valuable storage space by introducing a periodic process that replaces completed segments with compacted segments which only contain the latest version of the records.

```
"task": {
  "taskTypeConfigsMap": {
    "UpsertCompactionTask": {
      "schedule": "0 */5 * ? * *",
      "bufferTimePeriod": "7d",
      "invalidRecordsThresholdPercent": "30",
      "invalidRecordsThresholdCount": "100000"
    }
  }
}
```

The example above, bufferTimePeriod is set to "7d" which means that any segment that was completed over 7 days ago may be eligible for compaction. However, if you want to ensure that segments are compacted without any additional delay this config can be set to "0d".

invalidRecordsThresholdPercent is an optional limit to the amount of older records allowed in the completed segment represented as a percentage of the total number of records in the segment (i.e. old records / total records). In the example, this property is set to "30" which means that if more than 30% of the records in the completed segment are old, then the segment may be selected for compaction.

invalidRecordsThresholdCount is also a limit similar to the previous property, but allows you to express the threshold as a record count. In the example above, this property is

set to “100000” which means that if the segment contains more than 100K records then it may be selected for compaction.

Read more about the design of this feature.

DELETE Support for Upsert Tables

Apache Pinot upsert tables now support deleting records. Supporting delete with upsert avoids the need for the user to explicitly filter out invalid records in the query. SELECT _FROM table WHERE deleted_column != true becomes as simple as SELECT _ FROM table. Pinot will only return the latest non-deleted records from the table. This feature opens up the support to ingest Change Data Capture (CDC) data like Debezium where the changes from a source (typically, mutable) will contain DELETE events.

Deletes itself is implemented as a soft-delete in Apache Pinot with a dedicated boolean column that serves as a delete marker for the record. Pinot automatically filters out records that are marked in this column. For more details, please see the documentation.

NULL Value Support

This feature enables Postgres compatible NULL semantics in Apache Pinot queries. The NULL semantics are important for usability for full SQL compatibility which many BI applications like Tableau rely upon when invoking queries to render dashboards. Previously in Pinot, we could not represent NULL. The workaround was to use special values like Integer.MIN_VALUE to represent NULL. Now Pinot 1.0 has full support to represent NULL values. By adding NULL support, Pinot 1.0 has increased the Tableau certification pass rate by 90%.

Here are some examples of how NULLs will work in Pinot 1.0.

Aggregations

Given the following table below, aggregating columns with NULL values will have this behavior.

col1	col2
1	NULL
2	NULL
3	1

Since col1 does not contain NULL values, all the values are included in the aggregation.

```
select sum(col1) -- returns 6
select count(col1) -- returns 3
```

In the select statement below, the NULL values in col2 are not included in the aggregation.

```
select sum(col2) -- returns 1
select count(col2) -- returns 1
```

Group By

Pinot now supports grouping by NULL. In the example below, we are grouping by col1 which contains a NULL value. Given the table below, grouping by columns with NULL value will have this behavior.

col1
a
NULL
b
a

The following select statement will output the following result.

select col1, count(*) from table group by col1

col1	count()
a	2
b	1
NULL	1

Sorting

Pinot now allows you to specify the location of NULL values when sorting records. The default is to act as though NULLs are larger than non-NULLs.

Given this list of values, sorting them will result in the following.

`values: 1, 2, 3, NULL`

Example 1:

NULL values sort BEFORE all non-NULL values.

SQL:

```
select col from table order by col NULLS FIRST
```

`RESULT: NULL, 1, 2, 3`

Example 2:

NULL values sort AFTER all non-NULL values.

SQL:

```
select col from table order by col ASC NULLS LAST
```

`RESULT: 1, 2, 3, NULL`

Example 3:

Default behavior is NULL LAST.

SQL:

```
select col from table order by col
```

`RESULT: 1, 2, 3, NULL`

Index Pluggability

Today, Pinot supports multiple index types, like forward index, bloom filter, and range index. Before Pinot 1.0, index types were all statically defined, which means that in order to create a new index type, you'd need to rebuild Pinot from source. Ideally that shouldn't be the case.

To increase speed of development, Index Service Provider Interface (SPI), or index-spi, reduces friction by adding the ability to include new index types at runtime in Pinot. This opens the ability of adding third party indexes by including an external jar in the classpath and adding some configuration. This opens up Pinot indexing to lower-friction innovation from the community.

For now, SPI-accessible indexes are limited to single field index types. Features like the star-tree index or other multi-column approaches are not yet supported.

Apache Pinot Spark 3 Connector and Passing Pinot Options

Apache Spark users can now take advantage of Pinot's ability to provide high scalability, low latency, and high concurrency within the context of a Spark 3 cluster using the

Apache Pinot Spark 3 Connector.

This connector supports Apache Spark (2.x and 3.x) as a processor to create and push segment files to the database and can read realtime, offline, and hybrid tables from Pinot.

Now you can merge your streaming and batch datasets together in Spark to provide a full view of real-time and historical data for your machine learning algorithms and feature stores.

Performance Features

- Distributed, parallel scan
- Streaming reads using gRPC (optional)
- Column and filter push down to optimize performance
- Support for Pinot's Query Options that include: maxExecutionThreads, enableNullHandling, skipUpsert, etc.

Usability Features

- SQL support instead of PQL
- Overlap between realtime and offline segments is queried exactly once for hybrid tables
- Schema discovery - If schema is not specified, the connector reads the table schema from the Pinot controller, and then converts to the Spark schema.

Here is an example that reads a Pinot table, by setting the format to "pinot" spark will automatically load the Pinot connector and read the "airlinesStats" table. The queryOptions property allows you to provide Pinot Query Options.

```
val data = spark.read
  .format("pinot")
  .option("table", "airlineStats")
  .option("tableType", "offline")
  .option("queryOptions", "enableNullHandling=true,maxExecutionThreads=1")
  .load()
```

```
  .sql("SELECT * FROM airlineStats WHERE DEST = 'SFO'")

data.show(100)
```

Petabyte-Scale Log Storage and Search in Pinot with CLP

Compressed Log Processor (CLP) is a tool capable of losslessly compressing text logs and searching them in their compressed state. It achieves a better compression ratio than general purpose compressors alone, while retaining the ability to search the compressed log events without incurring the performance penalty of fully decompressing them. Part of CLP's algorithm was deployed within Uber to compress unstructured Spark logs, as they are generated, achieving an unprecedented compression of 169×.

Log events generated as JSON objects with user-defined schemas, meaning each event may have different keys. Such user-defined schemas make these events challenging to store in a table with a set schema. With Log Storage and Search in Pinot with CLP, users would be able to:

- Store their log events losslessly (without dropping fields)
- Store their logs with some amount of compression
- Query their logs efficiently

The CLP ingestion pipeline can be used on log events from a stream, such as JSON log events ingested from Kafka. The plugin takes two inputs: a JSON record and a list of fields to encode with CLP.

The fields to encode can be configured as shown:

```
{
    ...
    "tableIndexConfig": {
        ...
        "streamConfigs": {
            ...
```

```
            "stream.kafka.decoder.class.name": "org.apache.pinot.plugin.inputforma
            "stream.kafka.decoder.prop.fieldsForClpEncoding": "<field-name-1>,<fi
        }
    }
}
```

`<field-names-1 and 2>` are a comma-separated list of fields you wish to encode with CLP.

You can read the design document for more details into why and how this feature was implemented.

Summary

Apache Pinot's evolution is expressly due to the humans behind the code, and in reaching 1.0 release status it is proper and fitting to give credit to the open source project's key committers. Since its early days, over three hundred contributors have produced more than 1.3 million source lines of code (SLOC).


Image test

The introduction of Apache Pinot 1.0 represents an exceptional stride forward in real-time online analytical processing (OLAP) capabilities, marking a watershed moment in the evolution of real-time analytics systems. This release redefines the limits of what can be achieved in the realm of instant data analysis, presenting a game-changing solution for organizations seeking high throughput and low latency in their OLAP queries. If you would like to get started with Apache Pinot 1.0, you can check out the documentation, and download it now.

Resources

If you want to try out Apache Pinot, the following resources will help you get started:

[Download page]

[Getting started]

[Join our Slack channel]

[See our upcoming events]

[Follow us on social media]

Join our newsletter

Your email address