Streambased: A Composable real-time and analytical data platform

Tom Scott

streambased.

White paper | March 2026

Streambased: A Composable real-time and analytical data platform

Tom Scott, Streambased

Streambased is a modern, data serving solution architectured to combine the industry standard Apache Kafka event streaming and Apache Iceberg analytical data platforms.

Streambased provides a unified view of data in both systems to create a seamless, time bound data set that spans all most recent data in Apache Kafka and log term retained data in Apache Iceberg. Streambased serves this dataset in a format suitable for consumption by either Apache Kafka clients or Apache Iceberg engines, a service not delivered by existing frameworks such as KsqlDB or Apache Flink. This paper presents Streambased from a user's perspective and gives an overview of its architecture and main components.

1. Introduction

Streambased is a fully-integrated and accelerated data platform designed specifically to leverage the operational data capabilities of Apache Kafka and the analytical data capabilities of Apache Iceberg in a transparent manner. Streambased's goal is to provide applications with datasets that feature the, often contradictory, features of:

- **Low Latency** - <1 second between data point creation and availability for processing.
- **High volume** - access to large scale historical data.
- **Single source of truth** - no intermediate copies during data ingestion and processing.
- **High Performance** - takes advantage of indexes, partitioning and other data standard acceleration techniques

Streambased is currently available as a set of docker containers to be deployed into existing Kubernetes environments or as a managed service via streambased cloud.

Streambased leverages the existing Apache Kafka and Apache Iceberg projects to provide best of breed handling of data ingest and data storage. Much has already been written on these systems and this paper will not cover these projects, for background information on Kafka and Iceberg please visit https://kafka.apache.org/ and https://iceberg.apache.org/ respectively. Instead this paper will focus on the additions to Apache Kafka and Apache Iceberg that Streambased provides in order to enable cross system access to Iceberg engines and Kafka clients.

Streambased maintains compatibility with the entire Apache Kafka ecosystem, it is capable of taking advantage of open source components such as Confluent's Schema Registry for catalog management and also leverage proprietary implementations of the Kafka protocol (RedPanda, WarpStream etc.) as well as the original open source distribution. On the Iceberg side Streambased is compatible with all major catalog providers (Polaris, AWS Glue, Nessie and many more via the generic REST catalog interface) and supports the latest features available in Iceberg 3.

As with the underlying data systems it enhances, Streambased implements a distributed architecture based on daemon processes that are responsible for all aspects of query execution. These daemons have minimal coupling to the Kafka/Iceberg infrastructure and operate using only the public APIs of both. Streambased maintains a principle of reuse of Kafka and Iceberg ecosystem components where possible and so requires no separate access management infrastructure, instead leveraging to well established Apache Kafka and Iceberg security principles already in use for data applications.

The result is a scope of data access that far exceeds that of either underlying system and a reduction in required infrastructure, maintenance, and cost. Depending on the particular workload, Streambased also offers considerable performance improvements via database staples like indexing, caching and partitioning.

This paper discusses the services Streambased provides to the user and then presents an overview of its architecture and main components.

It will also briefly explore the performance enhancements possible due to the Streambased approach. These become especially apparent under workloads that Kafka and Iceberg are traditionally bad at. As we will demonstrate later for row lookups by message attribute (SELECT * FROM transactions WHERE transactionId='XXX'), Streambased is up to 30x faster than today's alternatives.

The remainder of this paper is structured as follows: the next section gives an overview of Streambased from the user's perspective and points out how it differs from other approaches to Kafka/Iceberg based data systems.. Section 3 presents the overall architecture of the system. Section 4 presents the frontend component, which includes query optimisation, Section 5 presents the backend component, which is responsible for view creation, protocol translation and schema enforcement.. Section 6 briefly evaluates the performance of Streambased with reference to industry leading Kafka -> Iceberg solutions and Section 7 concludes

2. User View of Streambased

Streambased was specifically targeted for integration with existing applications/tooling that integrate with Apache Kafka or Apache Iceberg. To achieve this Streambased presents 2 components:

1. Streambased I.S.K. (Iceberg Service for Kafka) presents 2 interfaces: An Iceberg RESt catalog and a S3 compatible storage service.
2. Streambased K.S.I. (Kafka Service for Iceberg) functions as a Kafka proxy, intercepting requests/responses from Kafka clients and enriching them with Streambased functionality.

Both components are configured in a way that is consistent with the underlying systems Streambased accesses. For instance, applications utilising K.S.I. will provide the regular set of Kafka connection parameters well understood in the ecosystem. Likewise, an Iceberg engine (e.g. Apache Spark) would connect using the same set of parameters as it would to any other Iceberg based data source.

Streambased uses a Kafka Schema Registry to determine table population and structure. This component is typically pre-existing for the operational domain and so no extra effort is required to create a table for Streambased usage. As there is no overhead in making tables available, Streambased will make every Kafka topic and Iceberg table (subject to authorisation) available to a user immediately on startup.

2.3 Stream <> Table duality

Streambased maps topics (the logical unit of data grouping in Kafka) to tables (the logical unit of data storage in BI tooling) and vice versa using the following simple principles:

These mappings ensure a coherent and easily understandable path for data structure from the operational domain into the analytical domain.

3. Architecture

Streambased is a horizontally scalable engine that runs either co-located or separate to the underlying Kafka/Iceberg infrastructure that serves the base data. Streambased is designed to be horizontally scalable to limits defined only by the underlying systems. For instance, a Streambased cluster could be created with 100s of nodes but, if the underlying Kafka cluster consists of only 3 nodes it is likely this underlying cluster would not be able to satisfy the load put upon it by the Streambased cluster. Streambased interacts with the Kafka cluster using only the public Kafka API and, this means the serving engine (Streambased), is completely decoupled from the storage engine (Kafka).

A Streambased deployment is separated by the type of client it serves. Streambased K.S.I is responsible for serving Kafka clients, I.S.K. is responsible for serving Iceberg engines.

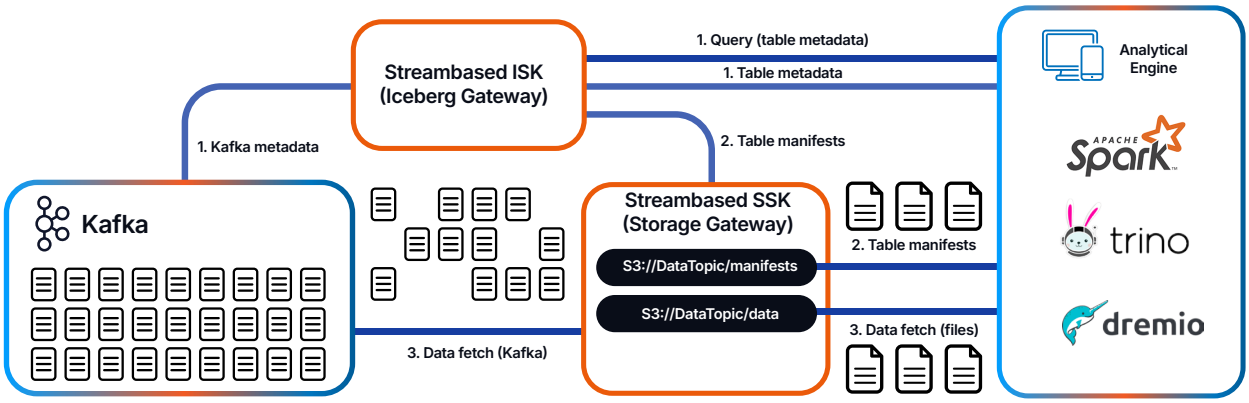
Kafka Unit	Streambased Unit	Description
Topic	Table Schema	As logical collections of data points both topics and tables are equivalent.
Message	Row	An individual message within a topic can be viewed as a row within a table. When combined with Schema Registry a repeating structure is enforced on them so they become equivalent.
Message field	Column	As above, using Schema Registry with Kafka messages ensures that they exhibit a particular structure. This structure is typically name:value:type and as such can easily be mapped to the traditional RDBMS column layout

3.1 The I.S.K. flow

Streambased I.S.K. serves a dataset made up of Apache Kafka and Apache Iceberg data to Iceberg engines. The flow that serves Iceberg data is essentially a passthrough to the underlying Iceberg cluster. On top of this an extra Iceberg snapshot is created that represents real-time data from Kafka. In detail this looks like:

1. **The Iceberg engine requests table metadata** - this request is served by the Streambased Iceberg REST catalog implementation and metadata is created by translating Kafka topic metadata into a format suitable for Iceberg engine consumption
2. **The Iceberg engine then requests manifest files** - these are once more metadata that is translated from Kafka but this time is served from the I.S.K. S3 compatible filesystem.
3. **Data file fetch** - The Iceberg engine requests data files. Via the I.S.K. S3 API. These data requests are mapped to consumer fetch requests in Kafka and the resultant fetches are served to the Iceberg engine.

Figure 1.1 shows the interactions between components that make up this flow.

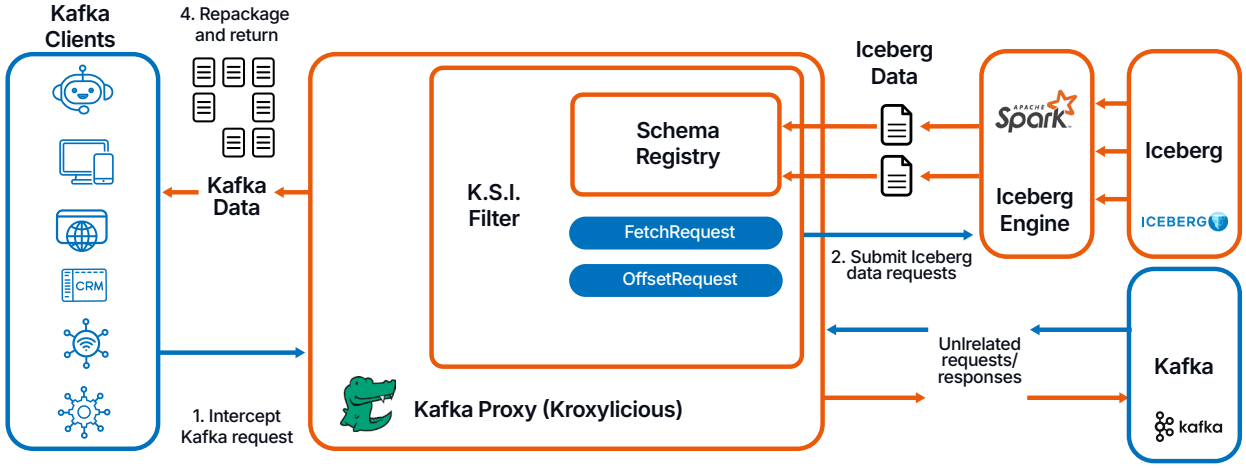


3.2 The K.S.I. flow

The K.S.I. flow serves Kafka clients from a dataset made up of recent data in Apache Kafka and historical data in Apache Iceberg. K.S.I. is implemented as a Kafka proxy that intercepts requests and responses to/from an underlying Kafka cluster. As K.S.I. is implemented as a proxy, any requests that are not relevant to Streambased are simply passed through to the underlying cluster in a transparent manner. The flow is as follows:

1. The client requests data from Kafka. If the data is available from Kafka the request is simply forwarded.
2. If the data is not available in Kafka K.S.I. reinterprets the request as an Iceberg data fetch and submits it to the underlying Iceberg store via an embedded or external engine.
3. The results of the Iceberg query are reformatted from table rows into streaming events.
4. The collection of events is packaged into a response and returned to the Kafka client

Figure 1.2 shows the interactions between components that make up this flow



Streambased's architecture cleanly separates compute from storage while unifying real-time and historical data access across Kafka and Iceberg. Through its dual flows I.S.K. for Iceberg engines and K.S.I. for Kafka clients, Streambased transparently translates between streaming and tabular paradigms, enabling each client type to interact with a logically continuous dataset.

This design establishes the foundation for a unified data plane that preserves native client semantics without requiring changes to existing applications.

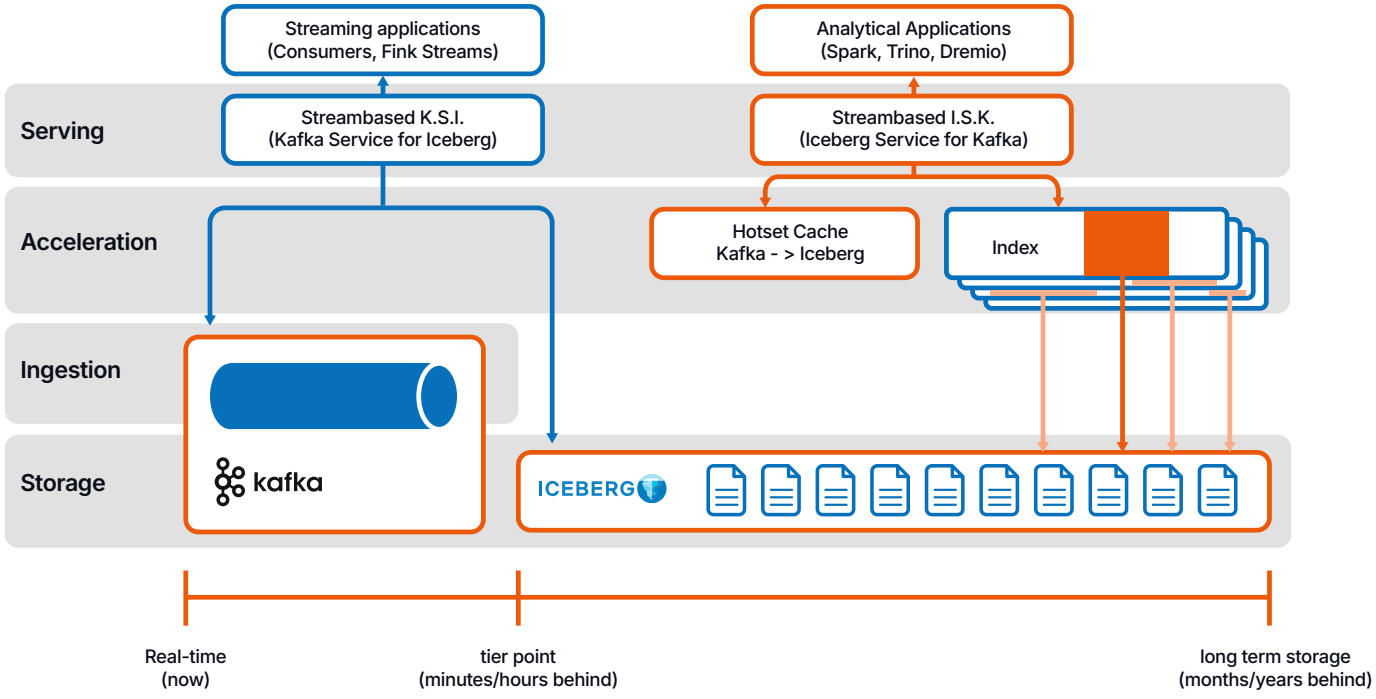
The final component of a Streambased deployment is the Streambased Indexing service (Hyperstream). This is an optional external service that can be used to compute and utilise indexing in Iceberg to accelerate analytical queries.

This service is surfaced as a REST API that applications can access providing the following functions:

1. Index Creation - Given a set of parameters, this API will create or update indexes over a composed dataset offered by I.S.K./K.S.I. These indexes are stored as Iceberg tables and publicly addressable by Iceberg engines.
2. Query Acceleration - This API interprets a prospective SQL query that will be submitted to a Streambased dataset and introduces new joins and clauses in order to utilise available indexes. This usually results in greatly improved query performance (see section 6 for metrics).

For detail on how indexing is implemented see the indexing section (4).

Figure 1.3 shows the architectural components of Streambased



4. Indexing

Performance enhancements offered by Streambased are focused around it's indexing technology. Streambased indexes do not physically modify the layout of the underlying data, instead they provide a translation layer where inefficient access patterns can be mapped to patterns that can be executed far more efficiently.

Apache Iceberg has the concept of partitioning, that allows data to be split into sections of similar data (for example, retail data may be partitioned by product category). When a query relates to these sections it can quickly eliminate irrelevant sections and greatly improve query performance. For instance, given our product category example above, a query with the clause: WHERE product_category = 'electricals' would not need to read product categories of clothing, bathroom or groceries in order to satisfy the query, greatly improving performance and resource usage.

In a Streambased composed dataset the data is by default partitioned by Kafka Offset/Partition. This makes it efficient to be read back via K.S.I. but it is unlikely to match analytical query clauses (and allow them to take advantage of indexing).

4.1 Predicate pushdown

Predicate pushdown is the process of extracting constraints present in an incoming SQL query and applying them at the data read stage of the query. The principle is that applying these constraints can eliminate a lot of the expensive read operations that make up the majority of time spent in large volume analytical queries.

In Streambased predicate pushdown leverages Apache Iceberg's partitioning to eliminate unnecessary reads. The flow is as follows:

1. Predicates parsed in the incoming SQL are extracted. For instance, given the query:

   SELECT *
   FROM transactions
   WHERE accountId = 1001001

   Streambased would extract the predicate associated with the accountId column.

2. Extracted predicates are compared against indexing information. If an accountId index is available, Streambased will extract all index information related to the key: 1001001

3. Streambased index information maps the key to the partitioning scheme of the underlying Iceberg table. In this case it would determine which Kafka Offset/Partition range is relevant to accountId: 1001001

4. Extra, partitioning specific, predicates are added to the original SQL query so that:

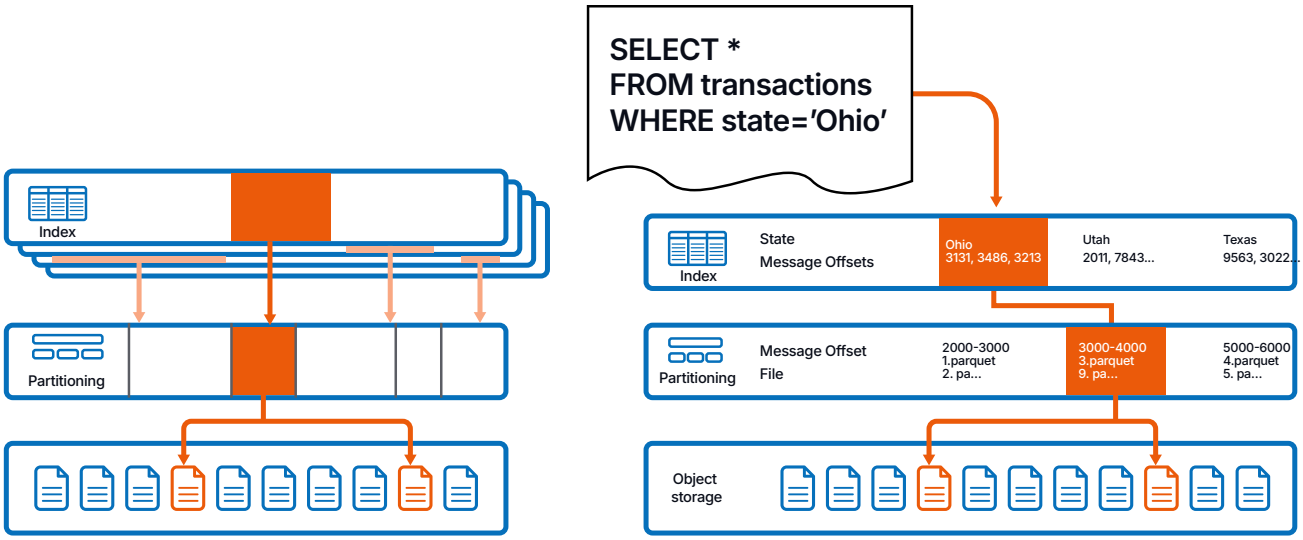
SELECT *
FROM transactions
WHERE accountId = 1001001

Becomes:

SELECT *
FROM transactions
WHERE accountId = 1001001
AND kafka_partition = 0
AND kafka_offset > 1000
AND kafka_offset < 2000

With these new predicates read performance is massively increased. Let's imagine that our example datset consists of 500,000 kafka offsets. Without indexing, this query would read all 500k records, as can be seen above, with indexing it reads only 1k!

Figure 1.4 shows the index mapping process graphically.



4.2 Graceful Index Miss Degradation

Streambased works on the principle of acceleration not transformation. This means that the extra index information created and stored by Streambased is not critical to obtaining the correct result for any given query but can be critical to that query performing in a way that is suitable for its application.

To give an example of this, imagine that the Streambased Indexing component has failed (this component is resilient by replication and so this is unlikely), Streambased I.S.K. can still accept and execute queries but the performance of queries in this degraded state are dramatically slower than those in "business as usual" state. This is because Streambased is no longer skipping any irrelevant partitions as indexing information is not available.

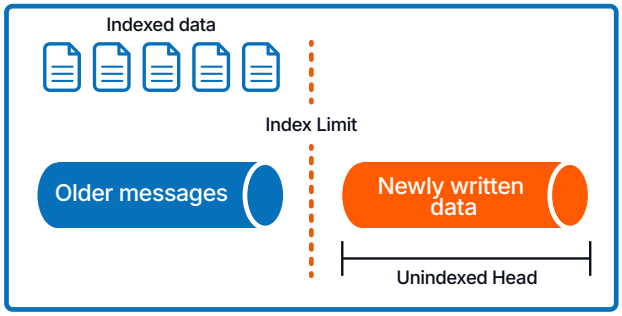
In fact, query performance will degrade gracefully from very high performance, fully indexed queries through lower performance partially indexed queries down to the unindexed, lowest performance queries discussed above.

To assert this with an example: let's say that our Iceberg view contains offsets 0 to 5000. Streambased indexing information exists for offsets 0-3000 but there is no indexing information for the 3001-5000 section.

No matter what query we submit, Streambased will always read the 3001-5000 section as it can not be certain whether data in this section is relevant to the query or not. In addition to this read, Streambased may read some, all or none or the 0-3000 section as it can be certain of the relevance of partitions in this section.

This principle is key when working with Streaming data as new data is expected continuously and any querying system should be able to answer queries on the new data with a minimum of latency.

Figure 1.5 shows index coverage split across historical and newly written data.



4.3 Thresholding

The performance increase afforded by utilising Streambased indexing can be dramatic, but so can the amount of pre-computed data that is required to achieve it. Streambased ensures that all indexing information is calculated asynchronously so as not to affect the operational capabilities of the Apache Kafka or Apache Iceberg cluster it is representing. It also ensures that administrators can balance the storage requirements for index data against the performance requirements for a given workload. This is typically implemented using thresholding.

Thresholding is the imposition of a limit on the total size of indexing information for a given table. This provides easy to predict storage requirements for a Streambased cluster that scale in step with the underlying Kafka cluster. In the event that a threshold is reached, index computation will stop and the index will remain incomplete until suitable space has been freed (by a background cleaning process).

5. Conclusion

In this paper we presented Streambased, a composable real-time and analytical data platform that unifies Apache Kafka and Apache Iceberg into a single, logically continuous data plane. By introducing the dual-service model of K.S.I. (Kafka Service for Iceberg) and I.S.K. (Iceberg Service for Kafka), Streambased enables streaming clients and analytical engines to operate over the same dataset without copying, transforming, or relocating data.

We demonstrate that the traditional boundary between operational and analytical systems is largely architectural rather than fundamental. Through transparent protocol translation, metadata synthesis, and optional indexing acceleration, Streambased allows Iceberg engines to query live Kafka data and Kafka clients to access historical Iceberg data as if it were still present in the log. This unified approach preserves native client semantics while dramatically expanding the accessible data horizon for both paradigms.

A key contribution of Streambased is its indexing architecture, which accelerates analytical workloads without modifying underlying data layout or interfering with ingestion performance. By mapping high-level predicates to partition-aware constraints, Streambased achieves significant performance gains (up to 30× in tested scenarios) while degrading gracefully in the absence of index coverage. This principle of acceleration without transformation ensures correctness and availability are never dependent on auxiliary structures, an essential property in continuously evolving streaming environments.

Importantly, Streambased achieves these capabilities without introducing a new storage layer, access control model, or proprietary protocol. It builds exclusively on the public APIs and ecosystem components of Kafka and Iceberg, maintaining compatibility with existing tooling, security models, and operational practices. As a result, organizations can deploy Streambased incrementally within their current infrastructure, reducing architectural complexity while extending system capability.

The broader implication of this work is that real-time and analytical data systems need not exist as separate architectural domains. By treating streams and tables as dual representations of the same underlying facts, Streambased establishes a unified data foundation in which data is recorded once, at its point of creation, and made immediately available for both operational and analytical use.

We anticipate that the historical separation between streaming systems and data lakes will continue to diminish. As workloads increasingly demand both low-latency responsiveness and deep historical context, converged architectures such as Streambased will become central to modern data infrastructure. By combining composability, performance acceleration, and architectural minimalism, Streambased offers a practical path toward that convergence and a foundation for the next generation of unified data platforms.