Griddable.io and Apache Kafka

Summary

Several open source projects were initiated at LinkedIn to create cloud-scale data integration including both Kafka, for event-based messaging and stream processing, and Databus, for scalable transactional data replication. Griddable.io and its core Databus technology provides several advantages in building distributed data integration pipelines with guaranteed consistency:

- *Timeline transactional consistency with the data source*
- *Extensible declarative policy engine*
- *Resilient pull-based architecture*
- *Unified APIs and data model for easy extensibility*

Today's digital business imperative means more data in more places across more platforms connected at ever increasing speed. This puts new pressures on traditional enterprise data to migrate and operate across hybrid clouds, feed continuously up-to-date copies into real-time applications, and connect to globally distributed edge computing. A new generation of distributed data integration technologies is now available as modern alternatives to traditional replication, message bus, and ETL solutions to provide more a flexible, highly connected approach that allows data transformations and processing in transit between sources and destinations.

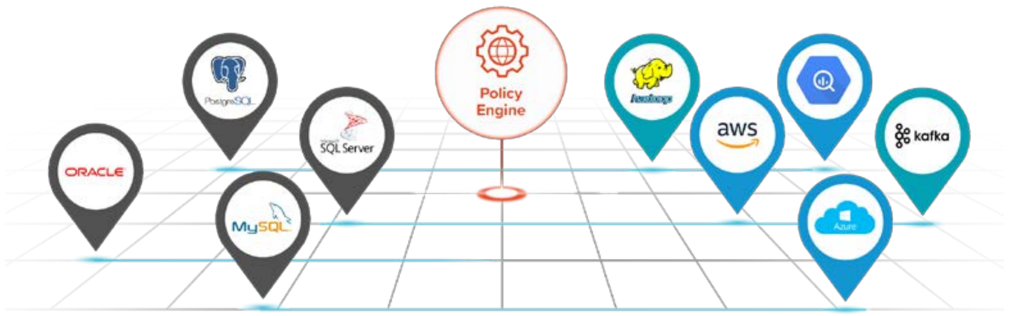
Apache Kafka

Apache Kafka is a popular open source platform for event-based messaging and stream processing.

Griddable.io

Griddable.io is the industry's first *smart grid for enterprise data* which brings a new approach optimized for synchronized data integration across any cloud and database platform. Griddable.io is purpose built for hybrid clouds and can also be installed on top of on-premise Oracle GoldenGate. Key attributes are:

- **A resilient scale-out grid** architecture that supports any topology and heterogeneous databases
- **Intelligent in-line grid services** that connect end-points and selectively filter, mask, or transform data while guaranteeing transaction consistency
- **Flexible grid policies that are easy to setup & change** using a declarative policy language, eliminating the need for complex custom coding or highly specialized expertise across a myriad of single function tools



The griddable.io smart grid for enterprise data guarantees transaction consistency across any number of sources and destinations

System properties: reliability, consistency, and availability

Griddable.io

Timeline consistency – Griddable.io smart grids provide timeline transactional consistency with a source database. All components in the grid see the sequence of change events as committed in the source database.

Pull-based architecture – All components on the grid pull data from upstream components. They are responsible for maintaining their own state of what has been successfully processed. Due to the timeline consistency property of the grid, that state is portable across other components so fail-over within a cluster or across clusters can be transparent.

Unlike push models, Griddable.io's pull model does not require expensive and complex distributed agreement protocols to guarantee reliability and consistency. Each component pulls transactions and change events in exactly the order that they happened in the source database. Keeping track of what has been successfully processed requires simple update of only its own state.

Source transactions capture – Data source transactions and transactional processing are a fundamental part of the data synchronization stream. Only committed change data is exposed to participants in the grid.

Consumer plugins are transaction-aware by default. The client library that runs the consumer plugins manages the state transactionally if the destination database supports, guaranteeing exactly once semantics. If the destination database does not support transactional updates, the client library provides at-least-once semantics.

Apache Kafka

Best effort reliability – By default, Kafka provides "best effort" reliability. Guaranteeing reliability for producers and consumers requires configuration and code changes: enabling "required.acks=all" mode for producers and manual updating of checkpoints for consumers.

Mixed push/pull model – Kafka utilizes a mixed push/pull model. Consumers pull data and maintain their state (also known as the Kafka offset). Producers push data which creates several challenges to achieve consistency. Enabling "required.acks=all" mode provides reliability of writes (i.e. at least one semantics) but does not guarantee ordering, which requires:

a) writing to a single partition, which has performance implications because it precludes scaling of write traffic by increasing the number of partitions
b) maintaining a single producer, which means that the availability of the producer needs to be ensured. One way to achieve this is through clustering with leader/stand-by model. Special care needs to be taken to avoid "split personality" problems where intermittently there are more than one leaders (e.g. due to network partitioning or garbage collection storms) which can corrupt the order of data written to Kafka.
c) exactly-once mode for producers (KIP-98). Item (b) Item (c) requires a complex distributed agreement protocol (like 2PC) which may affect performance or availability.

Source transactions overlay format - Capturing transactions in the event stream requires developing a custom event format that captures source transactions boundaries because Kafka by design is an event-based message bus. Using such custom format means that the existing ecosystem of source and target connectors cannot be utilized.

In addition, transactional processing of events by consumers requires the implementation of a custom Kafka consumer library that understands the source transactions overlay format, and updates the state (the checkpoint) transactionally. Such transactional processing of events will require complex and resource-intensive buffering of events (either by the standard Kafka consumer library or the transactional processing add-on) to filter out rolled-back Kafka transactions (see KIP-98).

Maintaining easy-to-use data integration

Griddable.io

Griddable.io architecture – Built on a functional model of processing where state is determined by the last successfully processed transaction in the source commit sequence (aka the source logical clock) and the policies applied. Policies can be applied on the fly without need for data materialization. This provides great flexibility for how data is read and processed. For example, a consumer that has last processed transaction *T1* and requires the application of policies P1 and P2. The consumer can

(i) stream events from the data source and apply the policies P1 and P2 in-line,
(ii) stream from another component (such as a relay) that has already cached the source change events and it can then apply P1 and P2,
(iii) stream from a component that has cached events with P1 already applied so that it needs to apply on P2, and so on.

Thus, the architecture supports the evolving the grid topology (e.g. by scaling the existing clusters or adding new clusters) to provide greater scalability, redundancy, moving data closer to consumer and pre-computing the results of common policies without affecting downstream services. Such components can also transparently employ persistence for increased retention of the data or fast in-memory processing for lower latency.

Extensible declarative policy engine – allows easy control of what data can and cannot be synchronized with powerful in-line data filtering and transformations.

Plugin model – based on common APIs and event/transaction data model. Plugins can be used to enhance the grids with support for new types of data sources, destinations or policy filters and transformations. The common APIs and data model guarantee that plugins can be easily combined to address a wide range of use cases with no coding.

Automated Policy-Based Cloning – the creation of a synchronized consistent copy from the data source to the destination is typically a two-phase process. First, a full copy of the source database is created. Second, incremental changes are continuously replicated to keep the copy synchronized. With griddable.io's smart grids, those two phases are fully integrated with the policy engine and the switch from the first to the second one is fully automated.

Web-based UI – easy deployment, configuration and a command-line interface for automation.

Apache Kafka

K/Connect, K/Streams, KSQL – Kafka provides APIs for integration with data sources and destinations (K/Connect) and for stream processing (K/Streams and KSQL). Those APIs do not standardize on the event value format and have varying guarantees in terms of consistency, support for streaming versus batch change capture and so on. All of these make it hard to integrate them without having to write and maintain additional code.

K/Streams can be extended with KSQL, a declarative language to define processing on the events from Kafka. Like SQL, KSQL is meant to process events from one or a small number of specific topics. It is unsuitable for defining policies that can be applied on a large or changing number of topics.

Intermediate materialization – In-line processing in above systems can be performed only in a single node. If distributed processing is required, intermediate data needs to be materialized in a new Kafka topic. There are important cases where intermediate materialization may be required such as moving the data in a different data center to achieve locality of access or to pre-compute common processing.

The above materialization has several implications. First, it introduces a new point of push semantics which requires complex and expensive exactly-once pushes. Second, it creates another persistent copy (or more accurately, three copies) of the data. Finally, it introduces new event sequence numbering which is hard to map to the original sequence.

The last implication is particularly important because the state of downstream components is tied to that sequence numbering. If the Kafka cluster or topic become partitioned, unavailable or corrupted, it is very difficult to fail over to a different cluster/topic because the state is not portable. In a complex data integration pipeline, if a single step becomes unavailable or corrupted, the entire pipeline is affected.

Building flexible data integration solutions

Griddable.io	Apache Kafka
Griddable.io's smart grids have their origin in Databus, an open-source project developed at LinkedIn to provide a scalable, replication bus for change data from LinkedIn's OLTP databases. Databus allowed several production consumer services to subscribe to a consistent stream of transaction data from the source database. Griddable.io provides a complete product for creating and maintaining distributed synchronized copies of enterprise databases.	Kafka was also developed at LinkedIn to capture event-based tracking and log data not persisted in the OLTP data stores. It provides an easy to use, high-throughput, durable storage for event-based messages with a streaming pub/sub API. Kafka is a project that defines a set of APIs for connectors (such as K/Connect) and event processing (such as K/Streams & KSQL). There are existing community-provided implementations for these APIs but the customer is expected to assemble an end-to-end solution.

Integration of griddable.io and Apache Kafka

Griddable.io smart grids and Apache Kafka can play complementary roles. Kafka can act as one of the sources and/or destinations for a grid. This can be useful where transactional consistency is not required and can utilize existing Kafka connectors for event-based processing. That allows the re-use of the same grid infrastructure for both transactional and non-transactional sources and destinations. Further, companies can utilize Griddable.io's ease of setup and powerful policy engine to control what data gets exposed to the downstream Kafka consumers.

griddable.io

2540 North First Street, Suite 201
San Jose CA 95131 USA
Phone 669.284.2143
www.griddable.io