

Best Practices for Stream Processing with GridGain and Apache Ignite and Kafka



Alexey Kukushkin
Professional Services



Rob Meyer
Outbound Product Management

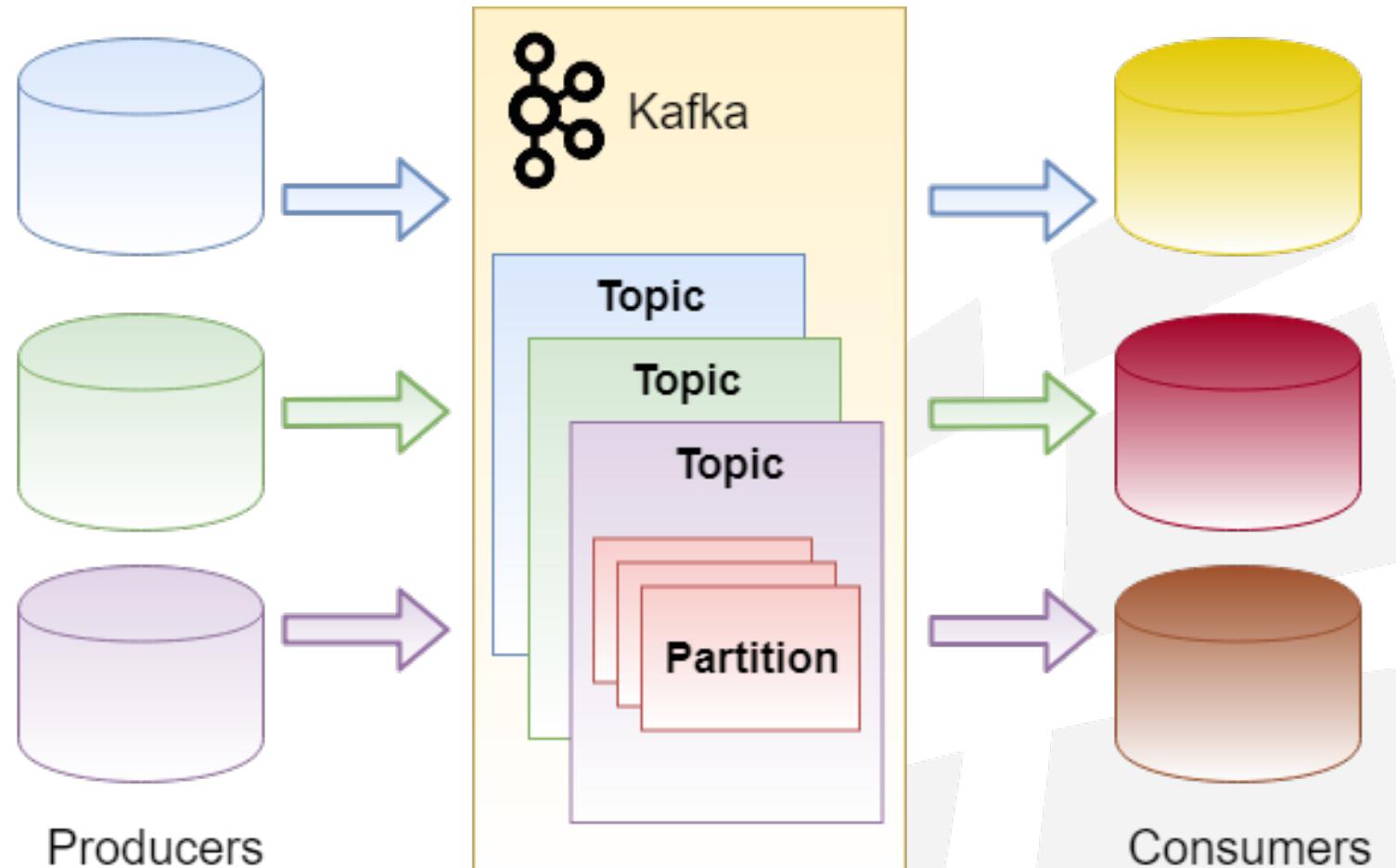
Agenda

- Why we need Kafka/Confluent-Ignite/GridGain integration
- Ignite/GridGain Kafka/Confluent Connectors
- Deployment, monitoring and management
- Integration Examples
- Performance and scalability tuning
- Q & A

Why we need Kafka/Confluent-Ignite/GridGain integration

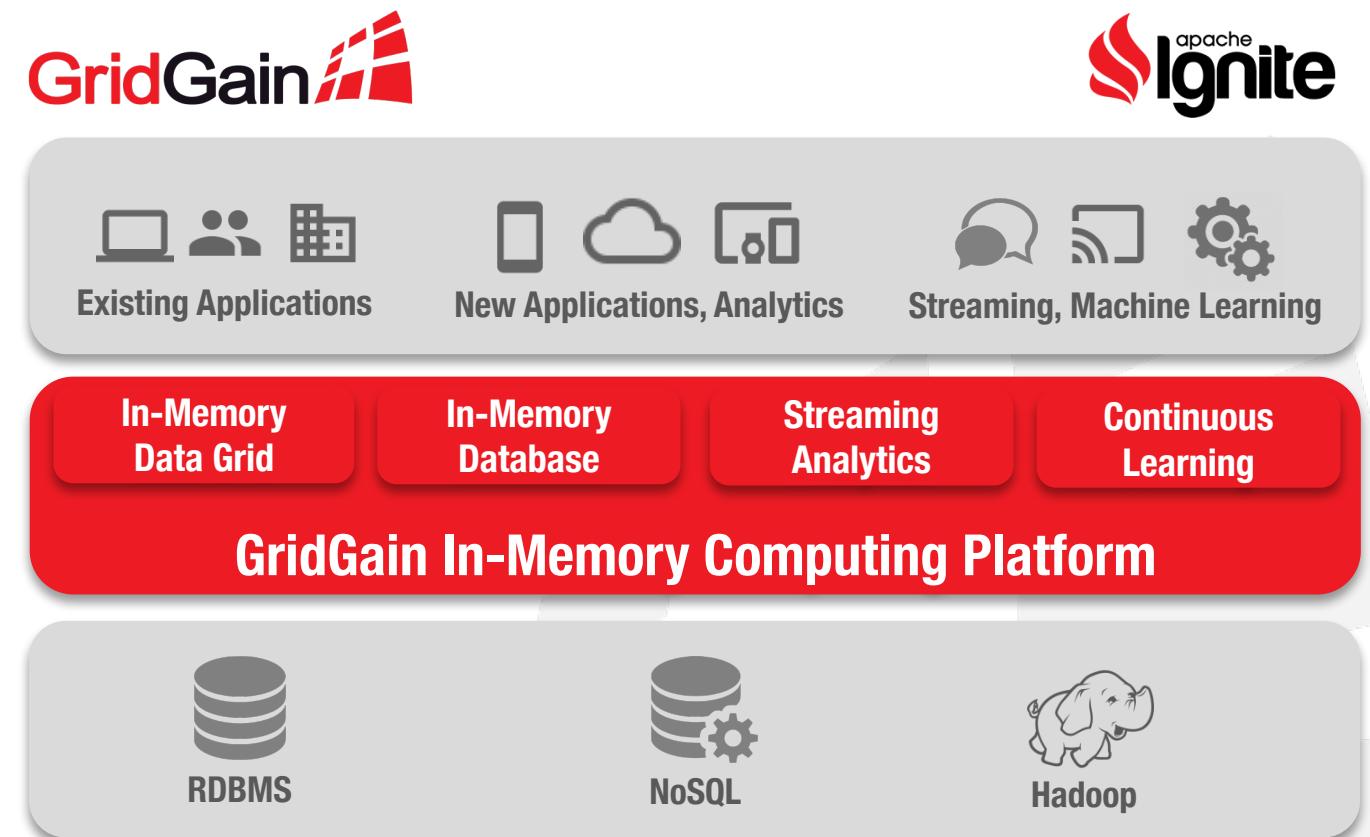
Apache Kafka and Confluent

- A distributed streaming platform:
- Publish/subscribe
 - Scalable
 - Fault-tolerant
 - Real-time
 - Persistent
 - Written mostly in Scala

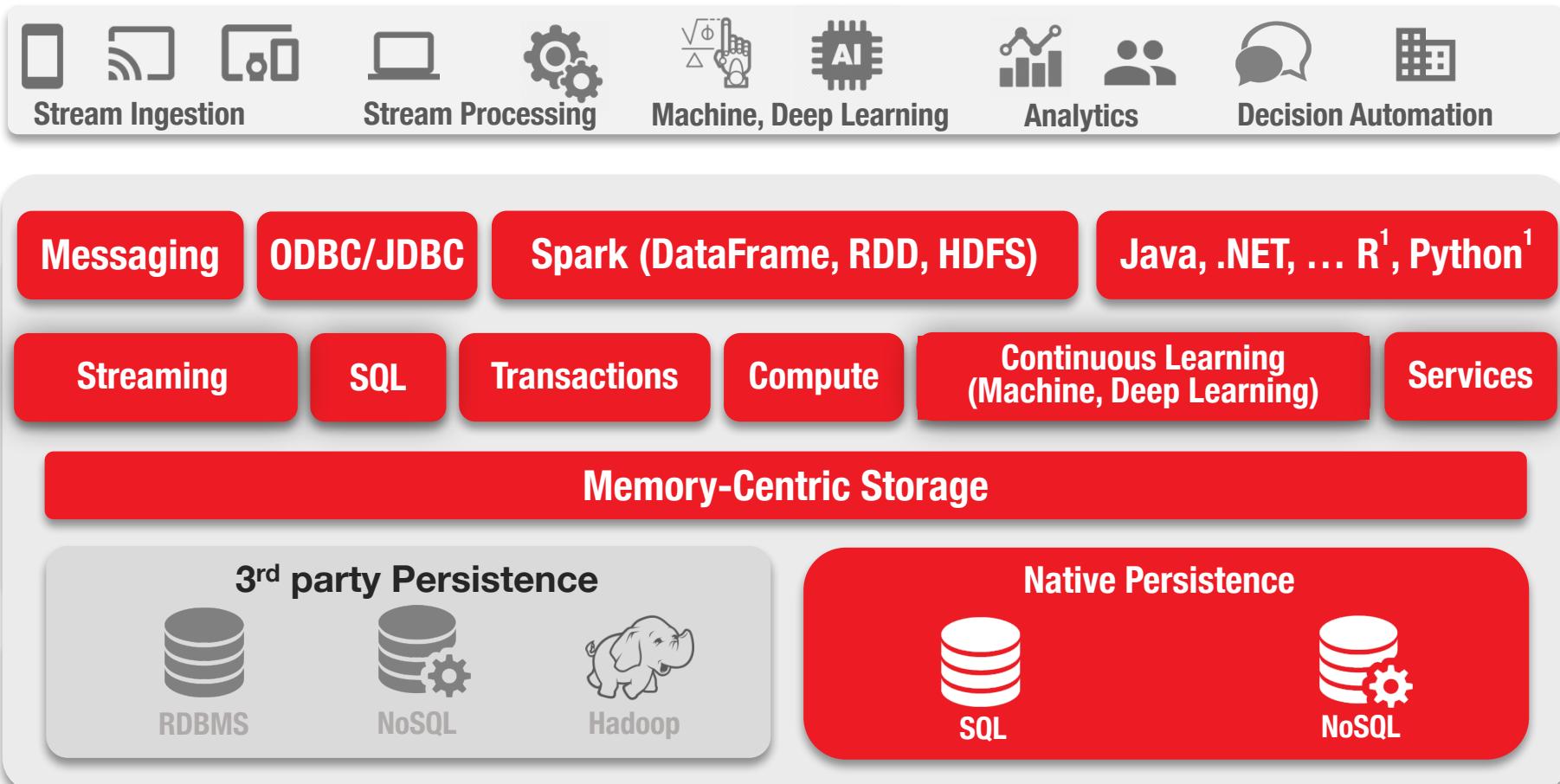


GridGain In-Memory Computing Platform

- Built on Apache Ignite
 - Comprehensive platform that supports all projects
 - No rip and replace
 - In-memory **speed**, petabyte **scale**
 - Enables HTAP, streaming analytics and continuous learning
- What GridGain adds
 - Production-ready releases
 - Enterprise-grade security, deployment and management
 - Global support and services
 - Proven for mission critical apps



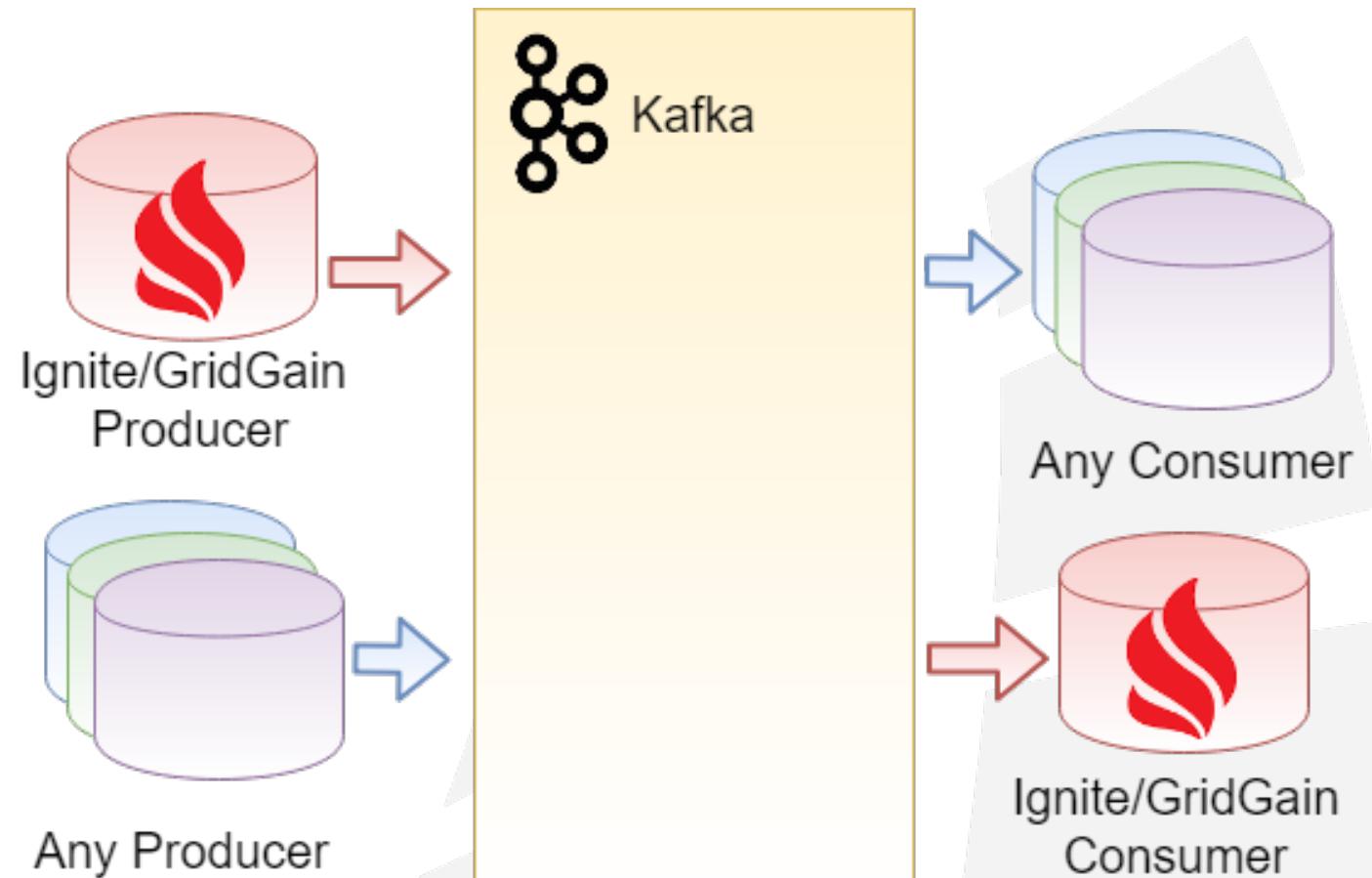
Streaming Analytics, Machine and Deep Learning



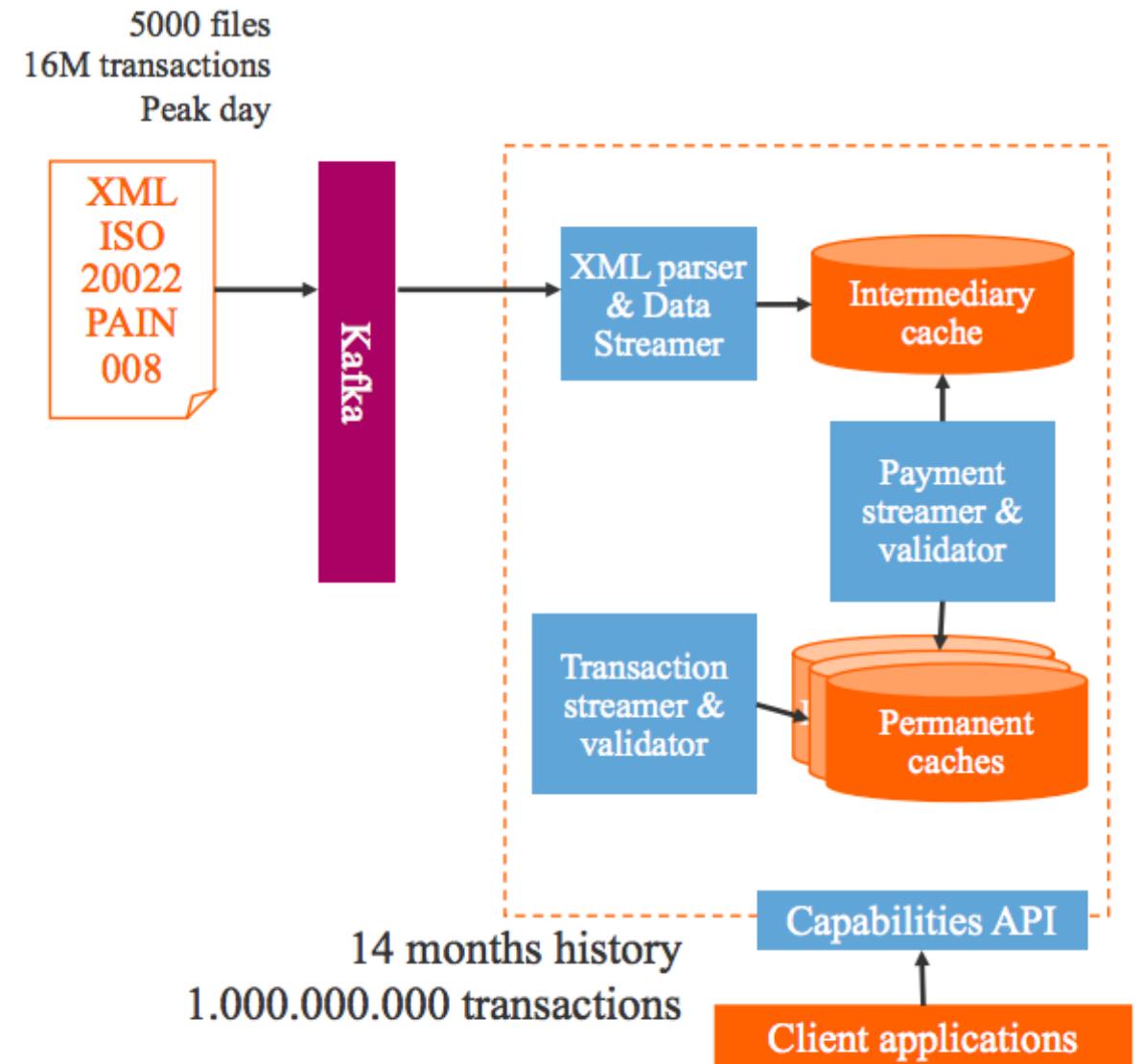
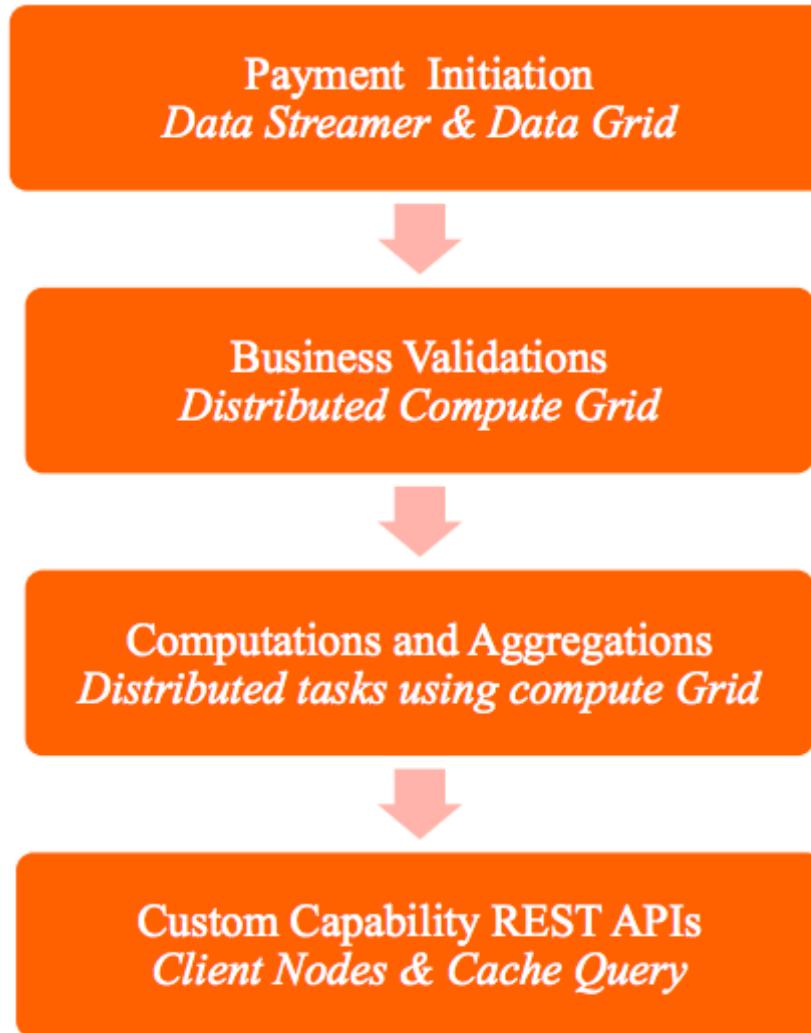
¹. R and Python developers currently invoke Java classes. Direct R and Python support planned.

Ignite/GridGain & Kafka Integration

- Kafka is commonly used as a messaging backbone in a heterogeneous system
- Add Ignite/GridGain to a Kafka-based system



SEPA DD



Ignite and GridGain Kafka Connectors

Developing Kafka Consumers & Producers

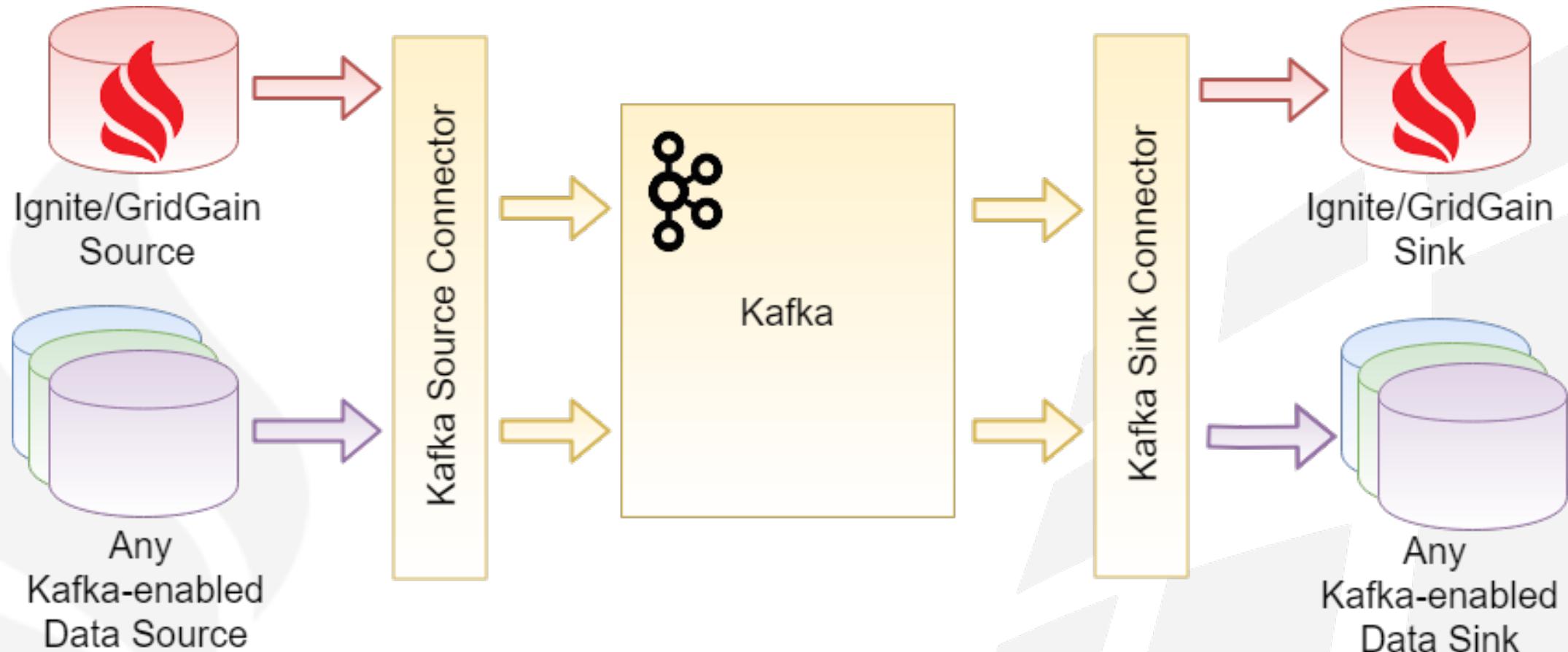
You can develop Kafka integration for any system using Kafka Producer and Consumer APIs but you need to solve problems like:

- How to use each API of every producer and consumer
- How Kafka will understand your data
- How data will be converted between producers and consumers
- How to scale the producer-to-consumer flow
- How to recover from a failure
- ... and many more

GridGain-Kafka Connector: Out-of-the-box Integration

- Addresses all the integration challenges using best practices
- Does not need any coding even in the most complex integrations
- Developed by GridGain/Ignite Community with help from Confluent to ensure both Ignite and Kafka best practices
- Based on Kafka Connect and Ignite APIs
 - Kafka Connect API encourages design for scalability, failover and data schema
 - GridGain Source Connector uses Ignite Continuous Queries
 - GridGain Sink Connector uses Ignite Data Streamer

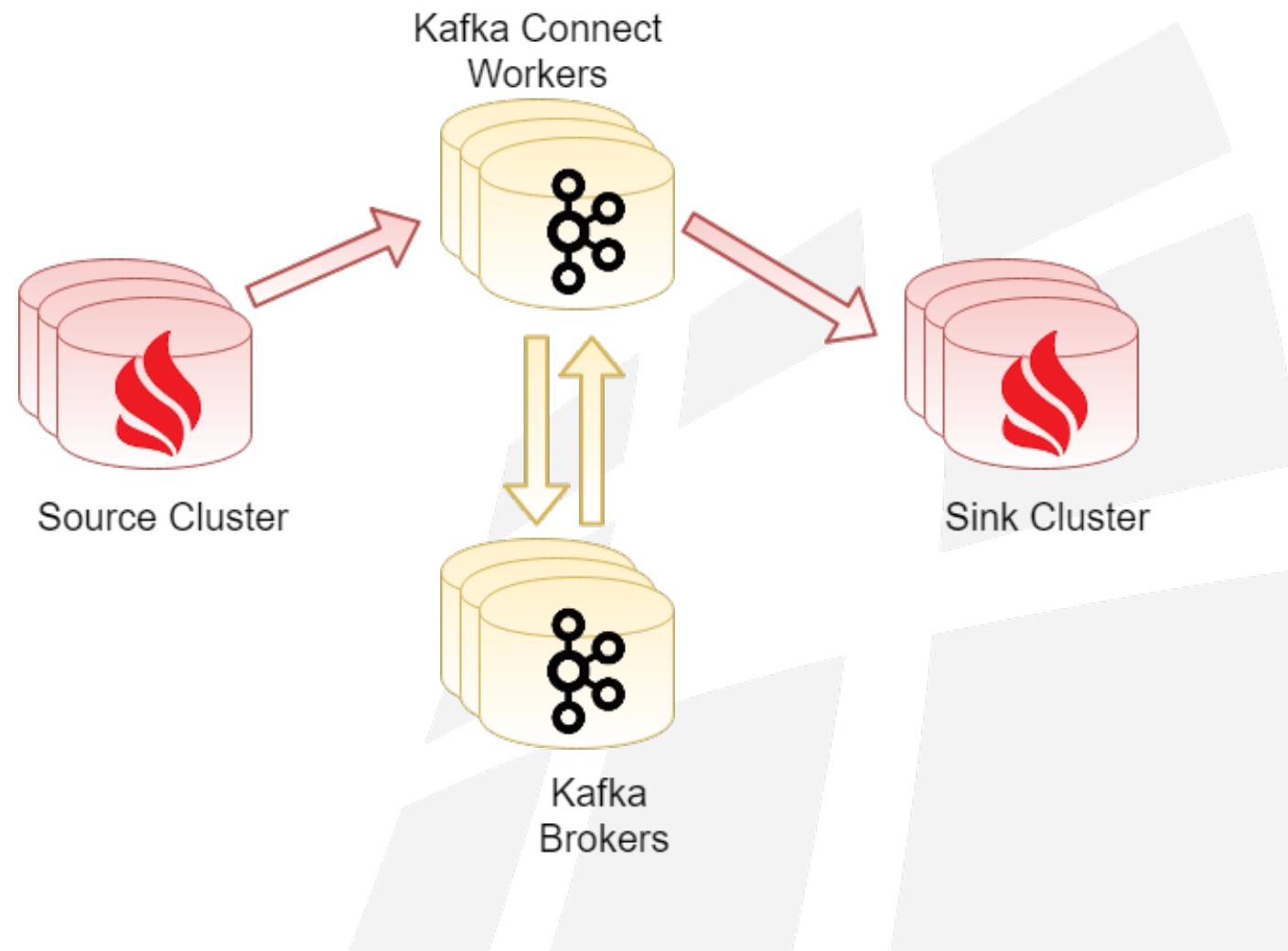
Kafka Source and Sink Connectors



Kafka Connect Server Types

In general, there are 4 separate clusters in Kafka Connect infrastructure:

- Kafka cluster
 - cluster nodes called **Brokers**
- Kafka Connect cluster
 - cluster nodes called **Workers**
- Source and Sink GridGain/Ignite clusters
 - **Server Nodes**



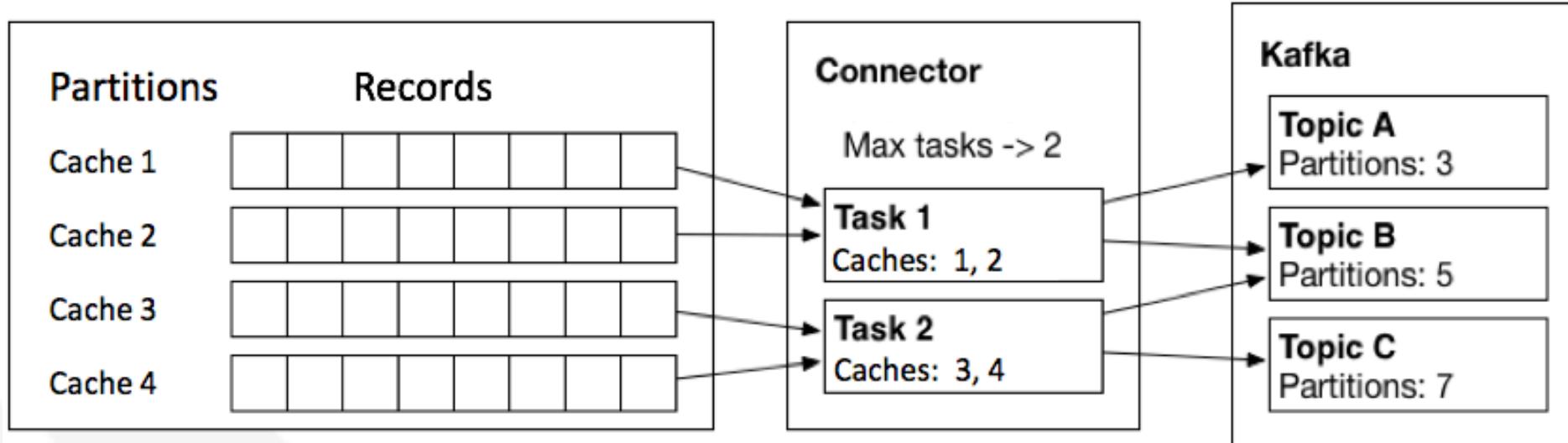
GridGain Connector Features

Two connectors independent from each other:

- GridGain Source Connector
 - streams data from GridGain into Kafka
 - uses Ignite continuous queries
- GridGain Sink Connector
 - streams data from Kafka into GridGain
 - uses Ignite data streamers

GridGain Source Connector: Scalability

Kafka Source Connector Model:



Scales by assigning multiple source **partitions** to Kafka Connect **tasks**.

For GridGain Source Connector:

- Partition = Cache
- Record = Cache Entry

GridGain Source Connector: Rebalancing and Failover

Rebalancing: re-assignment of Kafka Connectors and Tasks to Workers when

- A Worker joins or leaves the cluster
- A cache is added or removed

Failover: resuming operation after a failure

- how to resume after failure or rebalancing without losing cache updates occurred when Kafka Worker node was down?

Source Offset: position in the source stream. Kafka Connect:

- provides persistent and distributed source offset storage
- automatically saves last committed offset
- allows resuming from the last offset without losing data.

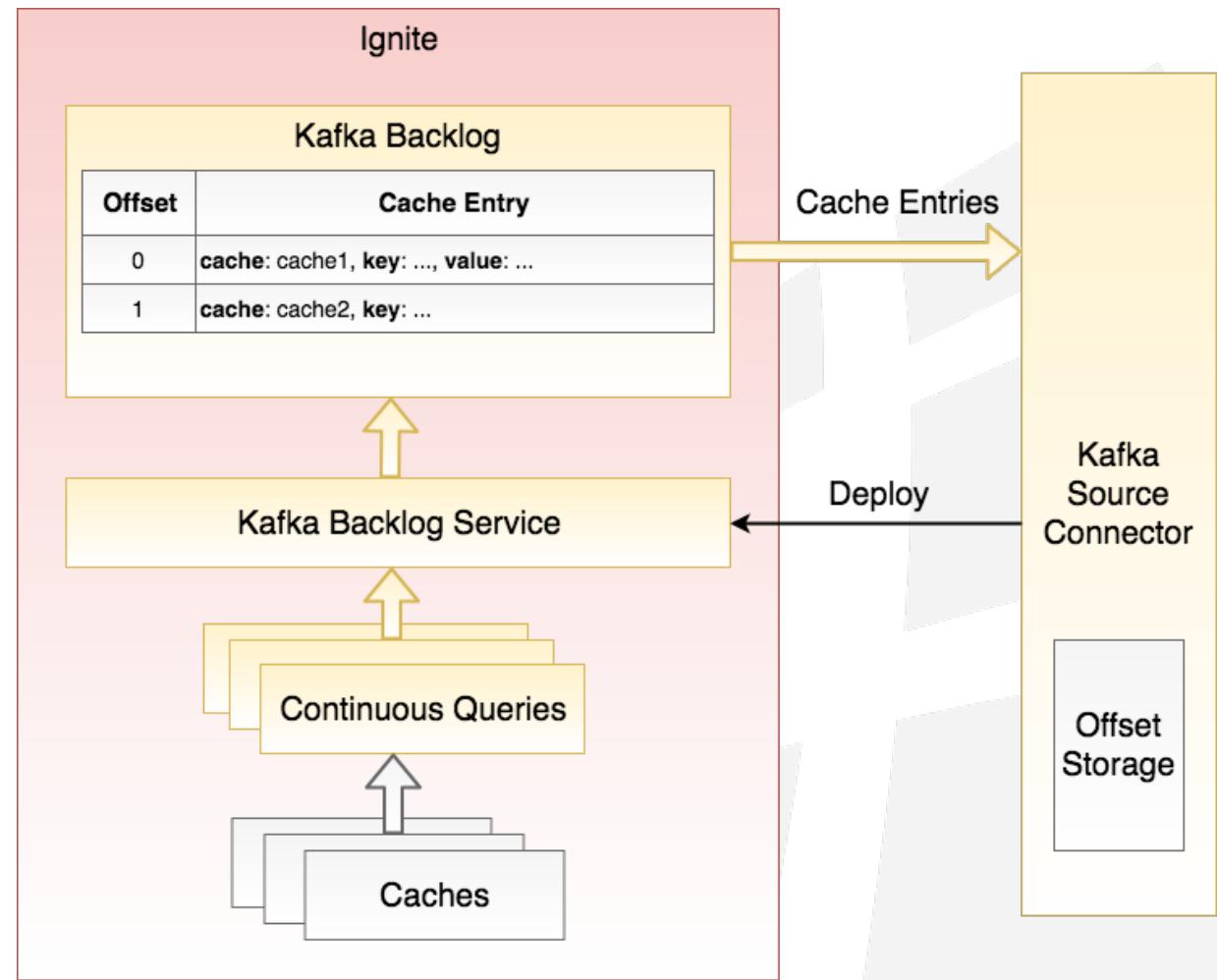
Problem: caches have no offsets!

GridGain Source Connector: Failover Policies

- **None:** no source offset saved, start listening to current data after restart
 - Cons: updates occurred during downtime are lost (“at least once” data delivery guarantee violated)
 - Pros: fastest
- **Full Snapshot:** no source offset saved, always pull all data from the cache upon startup
 - Cons:
 - Slow, not applicable for big caches
 - Duplicate data (“exactly once” data delivery guarantee is violated)
 - Pros: no data is lost

GridGain Source Connector: Failover Policies

- **Backlog:** resume from the last committed source offset
 - Kafka Backlog cache in Ignite
 - Key: incremental offset
 - Value: cache name and serialized cache entries
 - Kafka Backlog service in Ignite
 - Runs continuous queries pulling data from source caches into Backlog
 - Source Connector gets data from Backlog from backlog starting from the last committed offset
 - Cons
 - Intrusive: GridGain cluster impact
 - Complex configuration: need to estimate amount of memory for Backlog



GridGain Source Connector: Dynamic Reconfiguration

Connector monitors list of available caches and re-configures itself if a cache is added or removed.

Use `cacheWhitelist` and `cacheBlacklist` properties to define from which caches to pull data.

GridGain Source Connector: Initial Data Load

Use `shallLoadInitialData` configuration property to specify if you want the Connector to load the data that is already in the cache by the time the Connector starts.

GridGain Sink Connector

- Sink Connectors are inherently scalable since consuming data from a Kafka topic is scalable
- Sink Connectors inherently support failover thanks to the Kafka Connector framework auto-committing offsets of the pushed data.

GridGain Connector Data Schema

Both Source and Sink GridGain Connectors support data schema.

- Allows GridGain Connectors understand data with attached schema from other Kafka producers and consumers
- Source Connector attaches Kafka schema built from Ignite Binary objects
- Sink Connector converts Kafka records to Ignite Binary objects using Kafka schema

Limitations:

- Ignite Annotations are not supported
- Ignite CHAR converted to Kafka SHORT (same for arrays)
- Ignite UUID and CLASS converted to Kafka STRING (same for arrays)

Ignite Connector Features

- Ignite Source Connector
 - pushes data from Ignite into Kafka
 - uses Ignite Events
 - must enable `EVT_CACHE_OBJECT_PUT`, which negatively impacts cluster performance
- Ignite Sink Connector
 - pulls data from Kafka into Ignite
 - use Ignite data streamer

Apache Ignite vs. GridGain Connectors

Feature	Apache Ignite Connector	GridGain Connector
Scalability	Limited Source connector is not parallel Sink connector is parallel	Source connector creates a task per cache Sink connector is parallel
Failover	NO Source data is lost during connector restart or rebalancing	YES Source connector can be configured to resume from the last committed offset
Preserving source data schema	NO	YES
Handling multiple caches	NO	YES Connector can be configured to handle any number of caches
Dynamic Reconfiguration	NO	YES Source connector detects added or removed caches and re-configures itself

Apache Ignite vs. GridGain Connectors

Feature	Apache Ignite Connector	GridGain Connector
Initial Data Load	NO	YES
Handling data removals	YES	YES
Serialization and Deserialization of data	YES	YES
Filtering	Limited Only source connector supports a filter	YES Both source and sink connectors support filters
Transformations	Kafka SMTs	Kafka SMTs

Apache Ignite vs. GridGain Connectors

Feature	Apache Ignite Connector	GridGain Connector
DevOps	Some free-text error logging	Health Model defined
Support	Apache Ignite Community	Supported by GridGain, certified by Confluent
Packaging	Uber JAR	Connector Package
Deployment	Plugin PATH on all Kafka Connect workers	Plugin PATH on all Kafka Connect workers. CLASSPATH on all GridGain nodes.
Kafka API Version	0.10	2.0
Source API	Ignite events	Ignite continuous queries
Sink API	Ignite data streamer	Ignite data streamer

Deployment, monitoring and management

GridGain Connector Deployment

1. Prepare Connector Package
2. Register Connector with Kafka
3. Register Connector with GridGain

Prepare GridGain Connector Package

1. GridGain-Kafka Connector is part of GridGain Enterprise and Ultimate 8.5.3 (to be released in the end of October 2018)
2. The connector is in
\$GRIDGAIN_HOME/integration/gridgain-kafka-connect
 - (**GRIDGAIN_HOME** environment variable points to the root GridGain installation directory)
3. Pull missing connector dependencies into the package:

```
cd $GRIDGAIN_HOME/integration/gridgain-kafka-connect  
.copy-dependencies.sh
```

Register GridGain Connector with Kafka

For every Kafka Connect Worker:

1. Copy GridGain Connector package directory to where you want Kafka Connectors to be located
for example, into `/opt/kafka/connect` directory
2. Edit Kafka Connect Worker configuration (`kafka-connect-standalone.properties` **or** `kafka-connect-distributed.properties`) to register the connector on the plugin path:
`plugin.path=/opt/kafka/connect/gridgain-kafka-connect`

Register GridGain Connector with GridGain

This assumes GridGain version is 8.5.3

On every GridGain server node copy the below JARs into
\$GRIDGAIN_HOME/libs/user directory. Get the Kafka JARs from the Kafka
Connect workers:

- gridgain-kafka-connect-8.5.3.jar
- connect-api-2.0.0.jar
- kafka-clients-2.0.0.jar

Ignite Connector Deployment

1. Prepare Connector Package
2. Register Connector with Kafka

Prepare Ignite Connector Package

This assumes Ignite version is 2.6.

Create a directory containing the below JARs (find JARs in the
\$IGNITE_HOME/libs sub-directories):

- ignite-kafka-connect-0.10.0.1.jar
- ignite-core-2.6.0.jar
- ignite-spring-2.6.0.jar
- cache-api-1.0.0.jar
- spring-aop-4.3.16.RELEASE.jar
- spring-beans-4.3.16.RELEASE.jar
- spring-context-4.3.16.RELEASE.jar
- spring-core-4.3.16.RELEASE.jar
- spring-expression-4.3.16.RELEASE.jar
- commons-logging-1.1.1.jar

Register GridGain Connector with Kafka

For every Kafka Connect Worker:

1. Copy Ignite Connector package directory to where you want Kafka Connectors to be located
for example, into `/opt/kafka/connect` directory
2. Edit Kafka Connect Worker configuration (`kafka-connect-standalone.properties` or `kafka-connect-distributed.properties`) to register the connector on the plugin path:
`plugin.path=/opt/kafka/connect/ignite-kafka-connect`

Monitoring: GridGain Connector

Well defined Health Model:

- Numeric Event ID uniquely identifies specific problem
- Event severity
- Problem description and recovery action is available at
<https://docs.gridgain.com/docs/certified-kafka-connector-monitoring>

Configure your monitoring system to detect event ID in the logs and may be run automated recovery as defined in the Health Model

- Sample structured log entry (# used as a delimiter):
09-10-2018 19:57:35 # ERROR # 15000 # Spring XML configuration path is invalid: /invalid/path/ignite.xml

Monitoring: Ignite Connector

No Health Model is defined.

1. Run negative tests
2. Check Kafka and Ignite logs output
3. Configure your monitoring system to detect corresponding text patterns in the logs

Integration Examples

Propagating RDBMS updates into GridGain

Propagating RDBMS updates into GridGain

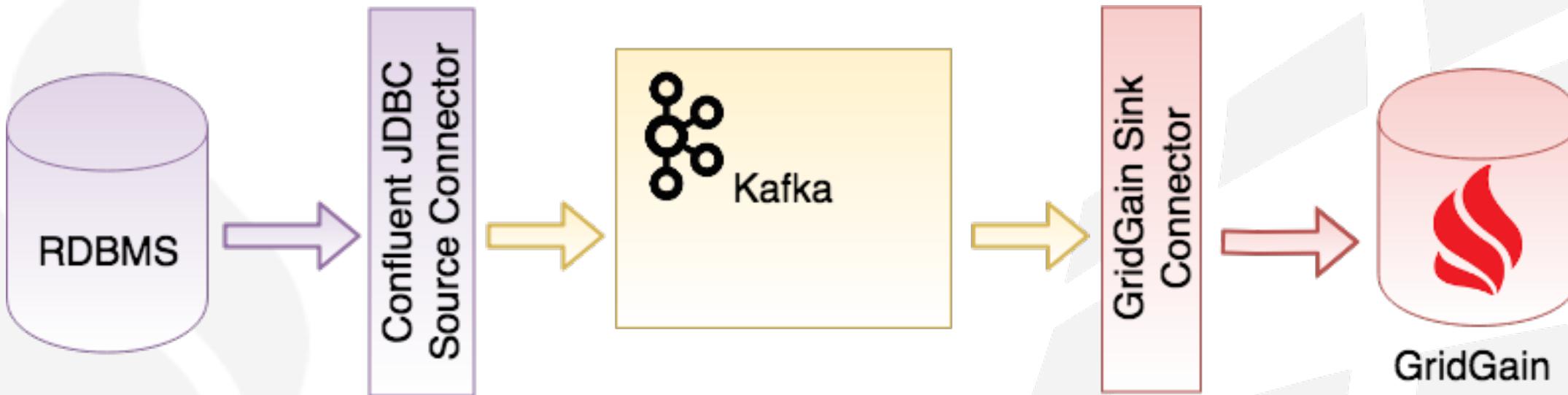
Ignite/GridGain has a 3rd Party Persistence feature (Cache Store) that allows:

- Propagating cache changes to external storage like RDBMS
- Automatically copying data from external storage to Ignite upon accessing data missed in Ignite

What if you want to propagate external storage change to Ignite at the moment of the change? - 3rd Party Persistence cannot do that!

Propagating RDBMS updates into GridGain

Use Kafka to achieve that without writing single line of code!



Assumptions

- For simplicity we will run everything on the same host
 - In distributed mode GridGain nodes, Kafka Connect workers and Kafka brokers are running on different hosts
- GridGain 8.5.3 cluster with GRIDGAIN_HOME variable set on the nodes
- Kafka 2.0 cluster with KAFKA_HOME variable set on all brokers

1. Run DB Server

We will use H2 Database in this demo.

We will use `/tmp/gridgain-h2-connect` as a work directory.

- Download H2 and set `H2_HOME` environment variable.
- Run H2 Server:

```
java -cp $H2_HOME/bin/h2*.jar org.h2.tools.Server -webPort 18082 -tcpPort 19092
TCP server running at tcp://172.25.4.74:19092 (only local connections)
PG server running at pg://172.25.4.74:5435 (only local connections)
Web Console server running at http://172.25.4.74:18082 (only local connections)
```

- In the opened H2 Web Console specify

JDBC URL:

`jdbc:h2:/tmp/gridgain-h2-connect/marketdata`

- Press Connect

Класс драйвера:	<code>org.h2.Driver</code>
JDBC URL:	<code>jdbc:h2:/tmp/gridgain-h2-connect/marketdata</code>
Пользователь Имя:	<code>sa</code>
Пароль:	
<input type="button" value="Соединиться"/> <input type="button" value="Тестовое соединение"/>	

2. Create DB Tables and Add Some Data

In H2 Web Console Execute:

- CREATE TABLE IF NOT EXISTS **QUOTES** (id int, date_time timestamp, price double, PRIMARY KEY (id));
- CREATE TABLE IF NOT EXISTS **TRADES** (id int, symbol varchar, PRIMARY KEY (id));
- INSERT INTO TRADES (id, symbol) VALUES (1, 'IBM');
- INSERT INTO QUOTES (id, date_time, price) VALUES (1, CURRENT_TIMESTAMP(), 1.0);

The screenshot shows the H2 Web Console interface. On the left, a sidebar displays the database structure with the following items:

- jdbc:h2:/tmp/gridgain-h2-connect (selected)
- QUOTES
- TRADES
- INFORMATION_SCHEMA
- Пользователи
- H2 1.4.197 (2018-03-18)

At the top right, there are buttons for "Выполнить" (Execute), "Выполнить выделенное" (Execute selected), and "Архив" (Archive). Below these buttons, two SQL queries are listed:

```
SELECT * FROM TRADES;  
SELECT * FROM QUOTES;
```

SELECT * FROM TRADES;

ID	SYMBOL
1	IBM

(1 строка, 0 ms)

SELECT * FROM QUOTES;

ID	DATE_TIME	PRICE
1	2018-10-10 14:47:37.096	1.0

(1 строка, 1 ms)

3. Start GridGain Cluster (Single-node)

```
<bean id="ignite.cfg" class="org.apache.ignite.configuration.IgniteConfiguration">
    <property name="discoverySpi">
        <bean class="org.apache.ignite.spi.discovery.tcp.TcpDiscoverySpi">
            <property name="ipFinder">
                <bean class="org.apache.ignite.spi.discovery.tcp.ipfinder.vm.TcpDiscoveryVmIpFinder">
                    <property name="addresses">
                        <list>
                            <value>127.0.0.1:47500</value>
                        </list>
                    </property>
                </bean>
            </property>
        </bean>
    </property>
</bean>
</bean>
```

```
$GRIDGAIN_HOME/bin/ignite.sh /tmp/gridgain-h2-connect/ignite-server.xml
[15:41:15] Ignite node started OK (id=b9963f9a)
[15:41:15] Topology snapshot [ver=1, servers=1, clients=0, CPUs=8, offheap=3.2GB, heap=1.0GB]
[15:41:15]     ^-- Node [id=B9963F9A-8F1E-4177-9743-F129414EB133, clusterState=ACTIVE]
```

4. Deploy Source and Sink Connectors

- Download Confluent JDBC Connector package from
<https://www.confluent.io/connector/kafka-connect-jdbc/>
- Unzip Confluent JDBC Connector package into /tmp/gridgain-h2-connect/confluentinc-kafka-connect-jdbc
- Copy GridGain Connector package from \$GRIDGAIN_HOME/integration/gridgain-kafka-connect into /tmp/gridgain-h2-connect/gridgain-kafka-connect
- Copy kafka-connect-standalone.properties Kafka worker configuration file from \$KAFKA_HOME/config into /tmp/gridgain-h2-connect and set the plugin path property:
plugin.path=/tmp/gridgain-h2-connect/confluentinc-kafka-connect-jdbc-5.1.0-SNAPSHOT,/tmp/gridgain-h2-connect/gridgain-gridgain-kafka-connect-8.7.0-SNAPSHOT

5. Start Kafka Cluster (Single-broker)

- Configure Zookeeper with /tmp/gridgain-h2-connect/zookeeper.properties:

```
dataDir=/tmp/gridgain-h2-connect/zookeeper  
clientPort=2181
```

- Start Zookeeper:

```
$KAFKA_HOME/bin/zookeeper-server-start.sh /tmp/gridgain-h2-  
connect/zookeeper.properties
```

- Configure Kafka broker: copy default \$KAFKA_HOME/config/server.properties to
/tmp/gridgain-h2-connect/kafka-server.properties customize it:

```
broker.id=0  
listeners=PLAINTEXT://:9092  
log.dirs=/tmp/gridgain-h2-connect/kafka-logs  
zookeeper.connect=localhost:2181
```

- Start Kafka broker:

```
$KAFKA_HOME/bin/kafka-server-start.sh /tmp/gridgain-h2-  
connect/kafka-server.properties
```

```
[2018-10-10 16:11:21,573] INFO Kafka version : 2.0.0  
(org.apache.kafka.common.utils.AppInfoParser)
```

```
[2018-10-10 16:11:21,573] INFO Kafka commitId : 3402a8361b734732  
(org.apache.kafka.common.utils.AppInfoParser)
```

```
[2018-10-10 16:11:21,574] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
```

6. Configure Source JDBC Connector

/tmp/gridgain-h2-connect/**kafka-connect-h2-source.properties**:

```
name=h2-marketdata-source
connector.class=io.confluent.connect.jdbc.JdbcSourceConnector
tasks.max=10

connection.url=jdbc:h2:tcp://localhost:19092//tmp/gridgain-h2-connect/marketdata
table.whitelist=quotes,trades

mode=timestamp+incrementing
timestamp.column.name=date_time
incrementing.column.name=id

topic.prefix=h2-
```

7. Configure Sink GridGain Connector

/tmp/gridgain-h2-connect/**kafka-connect-gridgain-sink.properties**:

```
name=gridgain-marketdata-sink
topics=h2-QUOTES,h2-TRADES
tasks.max=10
connector.class=org.gridgain.kafka.sink.IgniteSinkConnector
```

```
igniteCfg=/tmp/gridgain-h2-connect/ignite-client-sink.xml
topicPrefix=h2-
```

8. Start Kafka-Connect Cluster (Single-worker)

```
$KAFKA_HOME/bin/connect-standalone.sh \
/tmp/gridgain-h2-connect/kafka-connect-standalone.properties \
/tmp/gridgain-h2-connect/kafka-connect-h2-source.properties \
/tmp/gridgain-h2-connect/kafka-connect-gridgain-sink.properties
```

```
[2018-10-10 16:52:21,618] INFO Created connector h2-marketdata-source
(org.apache.kafka.connect.cli.ConnectStandalone:104)
```

```
[2018-10-10 16:52:22,254] INFO Created connector gridgain-marketdata-sink
(org.apache.kafka.connect.cli.ConnectStandalone:104)
```

9. See Caches Created in GridGain

Open GridGain Web Console Monitoring Dashboard at

<https://console.gridgain.com/monitoring/dashboard> and see GridGain Sink Connector created QUOTES and TRADES caches:

Caches: 2 Selected: 0 all											
Name	ID	M	On-heap entries					Off-heap entries			
			Total	Primary	Backup	Near	Total	Primary	Backup		
QUOTES	-1895070345	P	0	0	0	0	1	1	0		
TRADES	-1812385905	P	0	0	0	0	1	1	0		

10. See Initial H2 Data in GridGain

Open GridGain Web Console Queries page and run Scan queries for QUOTES and TRADES:

☰ Page: 1 Results so far: 1 Duration: 0ms

Key Class	Key	Value Class	
java.lang.Integer	1	o.a.i.i.binary.BinaryObjectImpl	TRADES [hash=-88677777, symbol=IBM, id=1]

Showing results for scan of TRADES

☰ Page: 1 Results so far: 1 Duration: 12ms

Key Class	Key	Value Class	Value
java.lang.Integer	1	o.a.i.i.binary.BinaryObjectImpl	QUOTES [hash=1628784114, date_time=2018-10-10 14:47:37.096, price=1.0, id=1]

Showing results for scan of QUOTES

11. Update H2 Tables

In H2 Web Console Execute:

- INSERT INTO TRADES (id, symbol) VALUES (2, 'INTL');
- INSERT INTO QUOTES (id, date_time, price) VALUES (2, CURRENT_TIMESTAMP(), 2.0);

The screenshot shows the H2 Web Console interface. On the left, a sidebar displays the database schema with tables QUOTES, TRADES, INFORMATION_SCHEMA, and Пользователи, along with the version information H2 1.4.197 (2018-03-18). The main area contains two execution panes.

Execution Pane 1:

```
Выполнить Выполнить выделенное Ае
SELECT * FROM TRADES;
SELECT * FROM QUOTES;
```

Execution Pane 2:

```
SELECT * FROM TRADES;
```

ID	SYMBOL
1	IBM
2	INTL

(2 строки, 0 ms)

```
SELECT * FROM QUOTES;
```

ID	DATE_TIME	PRICE
1	2018-10-10 14:47:37.096	1.0
2	2018-10-10 17:22:31.374	2.0

(2 строки, 0 ms)

12. See Realtime H2 Data in GridGain

Open GridGain Web Console Queries page and run Scan queries for QUOTES and TRADES:

≡ Page: 1 Results so far: 2 Duration: 0ms			
Key Class	Key	Value Class	
java.lang.Integer	1	o.a.i.i.binary.BinaryObjectImpl	TRADES [hash=-88677777, symbol=IBM, id=1]
java.lang.Integer	2	o.a.i.i.binary.BinaryObjectImpl	TRADES [hash=-1299591934, symbol=INTL, id=2]
Showing results for scan of TRADES			

≡ Page: 1 Results so far: 2 Duration: 0ms			
Key Class	Key	Value Class	Value
java.lang.Integer	1	o.a.i.i.binary.BinaryObjectImpl	QUOTES [hash=1628784114, date_time=2018-10-10 14:47:37.096, price=1.0, id=1]
java.lang.Integer	2	o.a.i.i.binary.BinaryObjectImpl	QUOTES [hash=-626738300, date_time=2018-10-10 17:22:31.374, price=2.0, id=2]
Showing results for scan of QUOTES			

Performance and scalability tuning

Disable Processing of Updates

For performance reasons, Sink Connector does not support existing cache entry update by default.

Set `shallProcessUpdates` configuration setting to `true` to make the Sink Connector update existing entries.

Disable Dynamic Schema

Source connector caches key and values schemas.

- The schemas are created as the first cache entry is pulled and re-used for all subsequent entries.

This works only if the schemas never change.

- Set `isSchemaDynamic` to `true` to support schema changes.

Consider Disabling Schema

Source Connector does not generate schemas
if `isSchemaless` configuration setting is `true`.

Disabling schemas significantly improves performance.

Carefully Choose Failover Policy

- Can allow losing data? Use **None**.
- Caches are small (e.g. reference data caches)? Use **Full Snapshot**.
- Otherwise use **Backlog**.

Plan Kafka Connect Backlog Capacity

Only **Backlog** failover policy supports both “at least once” and “exactly once” delivery guarantee.

GridGain Source Connector creates Backlog in the “kafka-connect” memory region, which requires capacity planning to avoid losing data by eviction (unless persistence is enabled).

Consider the worst case scenario:

- Maximum Kafka Connector worker downtime allowed in your system
- Peak traffic

Multiple peak traffic by max downtime to estimate “kafka-connect” data region size.

Q & A

Thank you!