Kafka Connectors

Apache Kafka

Machine Learning + Big Data in Real Time + Cloud Technologies

=> The Future of Intelligent Systems

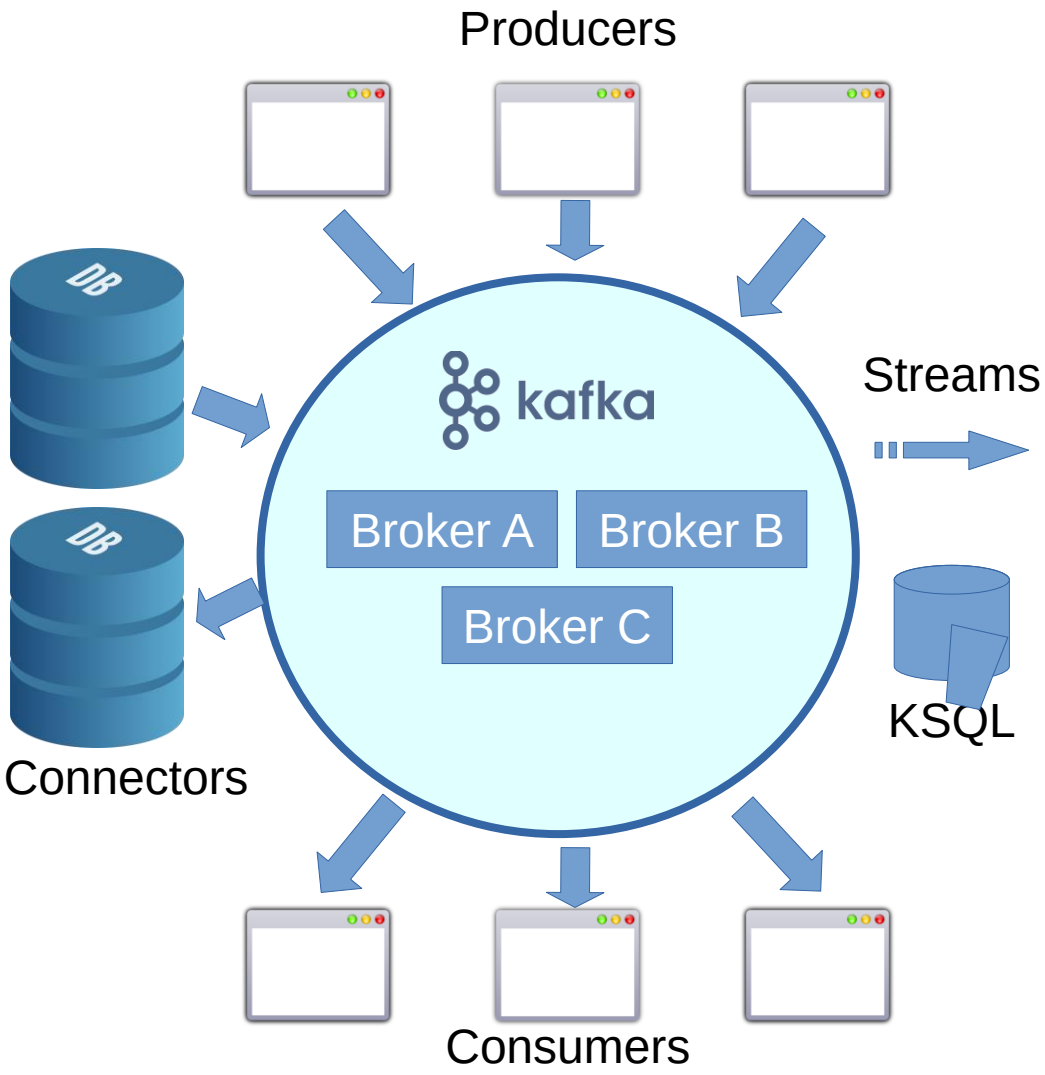
Where to Find The Code and Materials?

https://github.com/iproduct/course-apache-kafka

Kafka Connectors

Kafka Core APIs

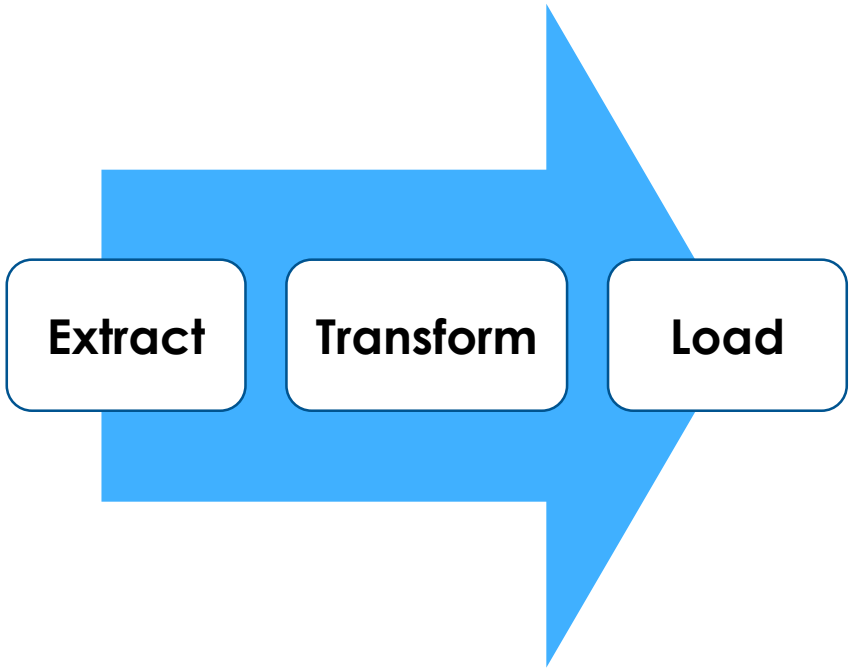
- The Producer API - publish a stream of records to one or more Kafka topics.
- The Consumer API - subscribe to one or more topics and process the stream of records produced to them.
- The Streams API - a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems – e.g. connector to a DB might capture every change in a table



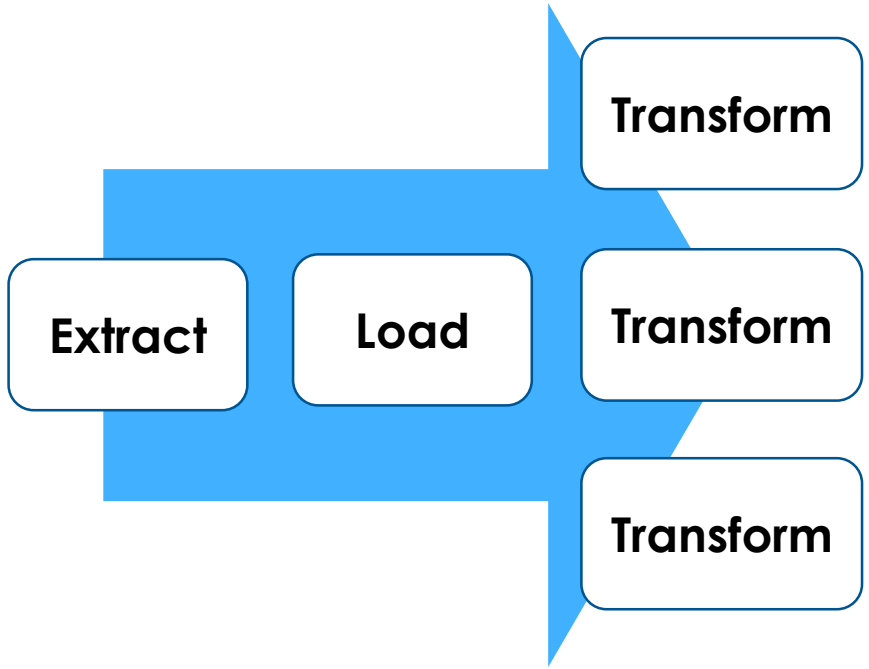
Kafka Connect

- Kafka Connect is a tool for scalably and reliably streaming data between Apache Kafka and other systems.
- It makes it simple to quickly define connectors that move large collections of data into and out of Kafka.
- Kafka Connect can ingest entire databases or collect metrics from all your application servers into Kafka topics, making the data available for stream processing with low latency.
- An export job can deliver data from Kafka topics into secondary storage and query systems or into batch systems for offline analysis.

Data Processing: ETL vs. ELT

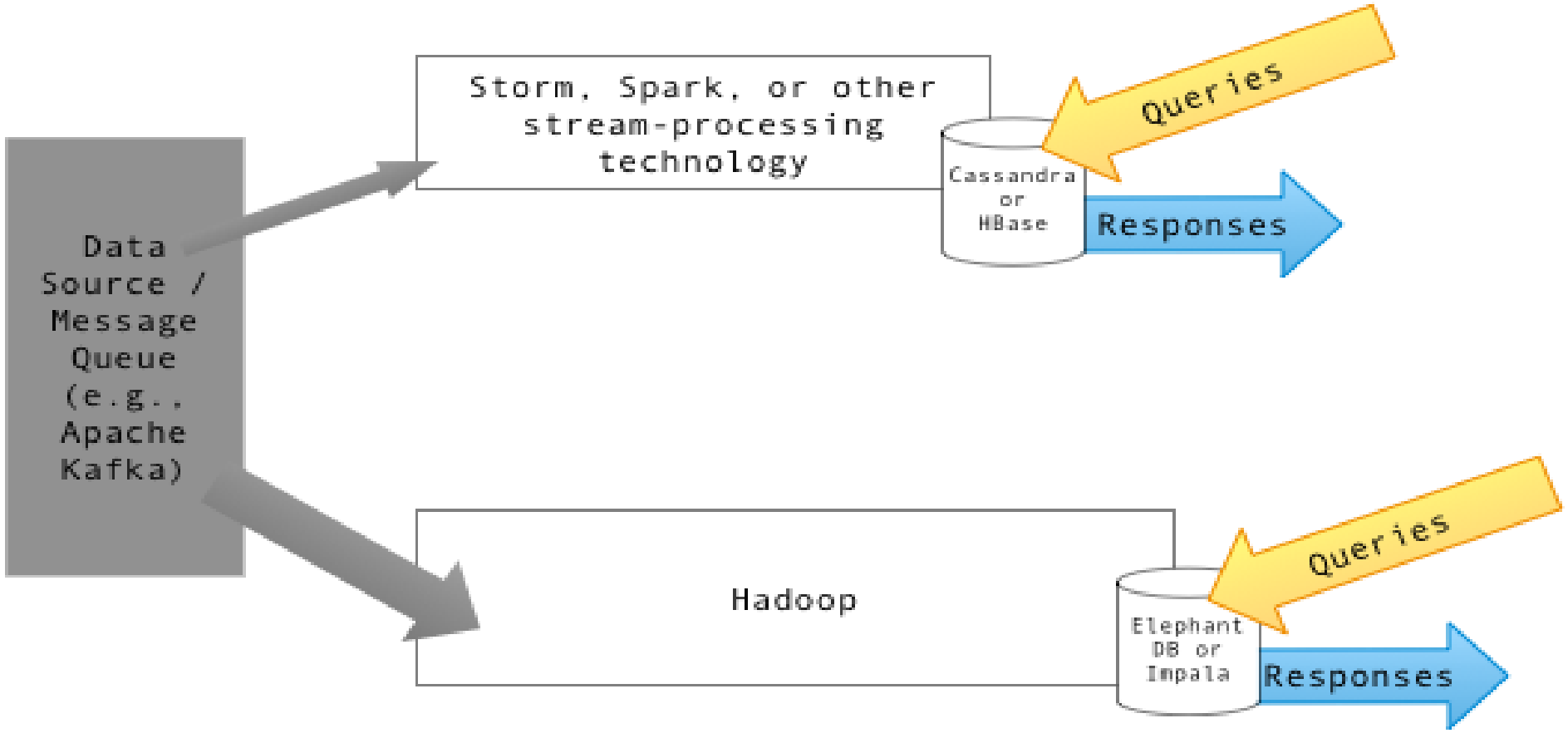


kafka



Lambda Architecture - I

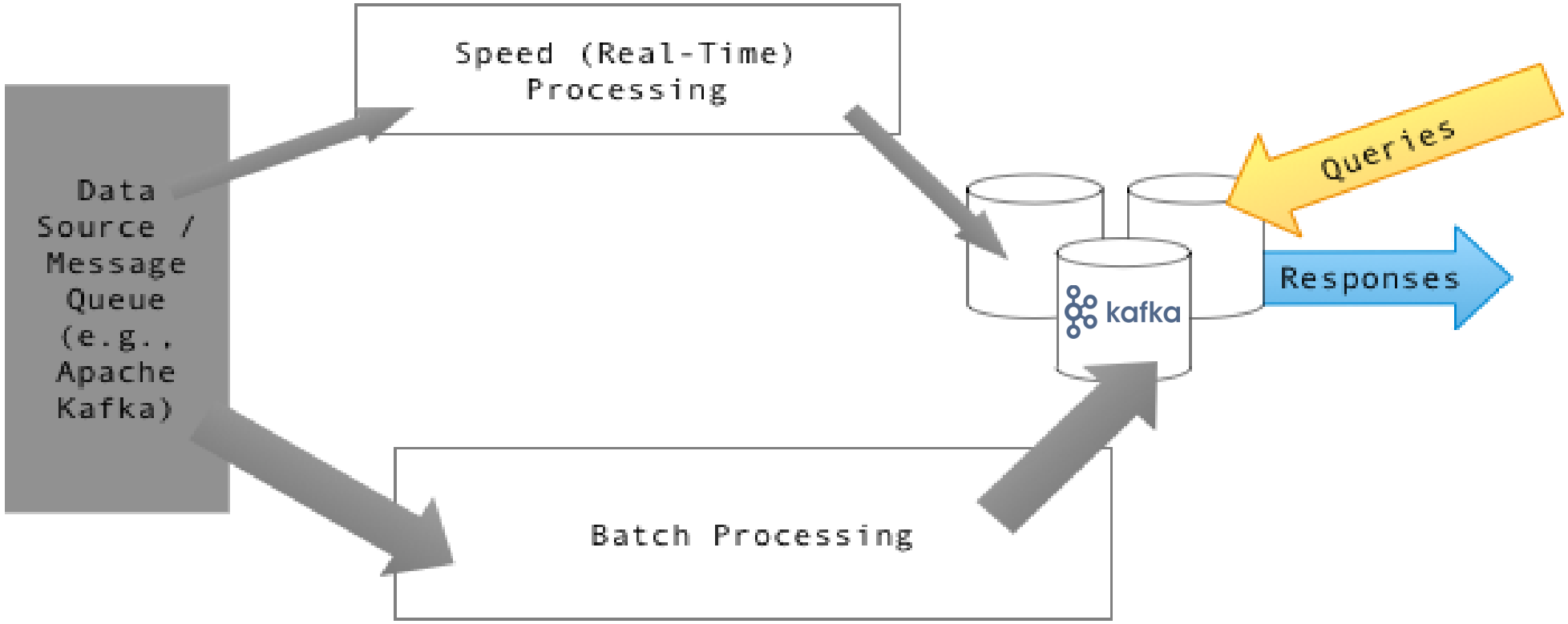
Query = λ (Complete data) = λ (live streaming data) * λ (Stored data)



https://commons.wikimedia.org/w/index.php?curid=34963987, By Textractor - Own work, CC BY-SA 4

Lambda Architecture - II

Query = λ (Complete data) = λ (live streaming data) * λ (Stored data)



https://commons.wikimedia.org/w/index.php?curid=34963986, By Textractor - Own work, CC BY-SA 4

Kafka Connect Features - I

- Kafka Connect standardizes integration of other data systems with Kafka, simplifying connector development, deployment, and management.
- Distributed and standalone modes - scale up to a large, centrally managed service supporting an entire organization or scale down to development, testing, and small production deployments.
- REST interface - submit and manage connectors to your Kafka Connect cluster via an easy to use REST API
- Automatic offset management - with just a little information from connectors, Kafka Connect can manage the offset commit process automatically so connector developers do not need to worry about this error prone part of connector development

https://kafka.apache.org/documentation.html#connect

Kafka Connect Features - II

- Distributed and scalable by default - Kafka Connect builds on the existing group management protocol. More workers can be added to scale up a Kafka Connect cluster.
- Streaming/batch integration - leveraging Kafka's existing capabilities, Kafka Connect is an ideal solution for bridging streaming and batch data systems.

https://kafka.apache.org/documentation.html#connect

Kafka Connect Resources and Downloads

- Kafka Connect official documentation – https://kafka.apache.org/documentation.html#connect
- Kafka Connect javadocs – https://kafka.apache.org/32/javadoc/index.html?org/apache/kafka/connect

- Download Debezium Change Data Capture (CDC) Connector for MySQL – https://debezium.io/documentation/reference/stable/connectors/index.html

Kafka Connect Common Properties – I

- bootstrap.servers - List of Kafka servers used to bootstrap connections to Kafka
- key.converter - Converter class used to convert between Kafka Connect format and the serialized form that is written to Kafka. This controls the format of the keys in messages written to or read from Kafka, and since this is independent of connectors it allows any connector to work with any serialization format. Examples of common formats include JSON and Avro.
- value.converter - Converter class used to convert between Kafka Connect format and the serialized form that is written to Kafka. This controls the format of the values in messages written to or read from Kafka, and since this is independent of connectors it allows any connector to work with any serialization format. Examples of common formats include JSON and Avro.

https://kafka.apache.org/documentation.html#connect

Kafka Connect Common Properties – II

- plugin.path (default empty) - a list of paths that contain Connect plugins (connectors, converters, transformations). Before running quick starts, users must add the absolute path that contains the example FileStreamSourceConnector and FileStreamSinkConnector packaged in **connect-file-"version".jar**, because these connectors are not included by default to the CLASSPATH or the plugin.path

The important configuration options specific to standalone mode are:

- offset.storage.file.filename - file to store offset data in

https://kafka.apache.org/documentation.html#connect

Configuring Individual Connectors Properties

- name - Unique name for the connector. Attempting to register again with the same name will fail.
- connector.class - The Java class for the connector
- tasks.max - The maximum number of tasks that should be created for this connector. The connector may create fewer tasks if it cannot achieve this level of parallelism.
- key.converter - (optional) Override the default key converter set by the worker.
- value.converter - (optional) Override the default value converter set by the worker.

https://kafka.apache.org/documentation.html#connect

Additional Sink Connectors Properties

- Sink connectors also have a few additional options to control their input. Each sink connector must set one of the following:

- topics - a comma-separated list of topics to use as input for this connector
- topics.regex - a Java regular expression of topics to use as input for this connector

https://kafka.apache.org/documentation.html#connect

Kafka Connect Simple Example using File Connector

connect-standalone config\connect-standalone.properties config\connect-file-source.properties config\connect-file-sink.properties

connect-standalone.properties

```
bootstrap.servers=localhost:9093

# The converters specify the format of data in Kafka and how to translate it into Connect data. Every Connect user will need to configure these based on the format they want their data in when loaded from or stored into Kafka
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# Converter-specific settings can be passed in by prefixing the Converter's setting with the converter we want to apply it to
key.converter.schemas.enable=false
value.converter.schemas.enable=false

offset.storage.file.filename=/tmp/connect.offsets
# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=1000

# Set to a list of filesystem paths separated by commas (,) to enable class loading isolation for plugins
# (connectors, converters, transformations).
plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors,
```

connect-file-source.properties

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=test.txt
topic=connect-test
```

connect-file-sink.properties

```
name=local-file-sink
connector.class=FileStreamSink
tasks.max=1
file=test.sink.txt
topics=connect-test
```

Example: FileStreamSinkConnector - I

```
public class FileStreamSinkConnector extends SinkConnector {

    public static final String FILE_CONFIG = "file";
    private static final ConfigDef CONFIG_DEF = new ConfigDef()
        .define(FILE_CONFIG, Type.STRING, null, Importance.HIGH, "Destination filename. If not specified, the
standard output will be used");

    private String filename;

    @Override
    public String version() {
        return AppInfoParser.getVersion();
    }

    @Override
    public void start(Map<String, String> props) {
        AbstractConfig parsedConfig = new AbstractConfig(CONFIG_DEF, props);
        filename = parsedConfig.getString(FILE_CONFIG);
    }
```

Example: FileStreamSinkConnector - II

```
@Override
public Class<? extends Task> taskClass() {
    return FileStreamSinkTask.class;
}

@Override
public List<Map<String, String>> taskConfigs(int maxTasks) {
    ArrayList<Map<String, String>> configs = new ArrayList<>();
    for (int i = 0; i < maxTasks; i++) {
        Map<String, String> config = new HashMap<>();
        if (filename != null)
            config.put(FILE_CONFIG, filename);
        configs.add(config);
    }
    return configs;
}
```

Example: FileStreamSinkConnector - III

```
    @Override
    public void stop() {
        // Nothing to do since FileStreamSinkConnector has no background monitoring.
    }

    @Override
    public ConfigDef config() {
        return CONFIG_DEF;
    }
}
```

Example: FileStreamSinkTask - I

```
public class FileStreamSinkTask extends SinkTask {
    private static final Logger log = LoggerFactory.getLogger(FileStreamSinkTask.class);

    private String filename;
    private PrintStream outputStream;

    public FileStreamSinkTask() {
    }

    // for testing
    public FileStreamSinkTask(PrintStream outputStream) {
        filename = null;
        this.outputStream = outputStream;
    }

    @Override
    public String version() {
        return new FileStreamSinkConnector().version();
    }
```

Example: FileStreamSinkTask - II

```
    @Override
    public void start(Map<String, String> props) {
        filename = props.get(FileStreamSinkConnector.FILE_CONFIG);
        if (filename == null) {
            outputStream = System.out;
        } else {
            try {
                outputStream = new PrintStream(
                    Files.newOutputStream(Paths.get(filename), StandardOpenOption.CREATE,
StandardOpenOption.APPEND),
                    false,
                    StandardCharsets.UTF_8.name());
            } catch (IOException e) {
                throw new ConnectException("Couldn't find or create file '" + filename + "' for FileStreamSinkTask", e);
            }
        }
    }
```

Example: FileStreamSinkTask - III

```
@Override
    public void put(Collection<SinkRecord> sinkRecords) {
        for (SinkRecord record : sinkRecords) {
            log.trace("Writing line to {}: {}", logFilename(), record.value());
            outputStream.println(record.value());
        }
    }
    @Override
    public void flush(Map<TopicPartition, OffsetAndMetadata> offsets) {
        log.trace("Flushing output stream for {}", logFilename());
        outputStream.flush();
    }
    @Override
    public void stop() {
        if (outputStream != null && outputStream != System.out)
            outputStream.close();
    }

    private String logFilename() {
        return filename == null ? "stdout" : filename;
    }
```

Transformations

- Connectors can be configured with transformations to make lightweight message-at-a-time modifications. They can be convenient for data messaging and event routing.
- A transformation chain can be specified in the connector configuration.
  - transforms - list of aliases for the transformation, specifying the order in which the transformations will be applied.
  - transforms.$alias.type - fully qualified class name for the transformation.
  - transforms.$alias.$transformationSpecificConfig - configuration properties for the transformation

https://kafka.apache.org/documentation.html#connect

Transformations Example

For example, lets take the built-in file source connector and use a transformation to add a static field.

The file source connector reads each line as a String. We will wrap each line in a Map and then add a second field to identify the origin of the event. To do this, we use two transformations:

- HoistField to place the input line inside a Map
- InsertField to add the static field. In this example we'll indicate that the record came from a file connector

https://kafka.apache.org/documentation.html#connect

Transformations Example: connect-file-source.properties

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=test.txt
topic=connect-test
transforms=MakeMap, InsertSource
transforms.MakeMap.type=org.apache.kafka.connect.transforms.HoistField$Value
transforms.MakeMap.field=line
transforms.InsertSource.type=org.apache.kafka.connect.transforms.InsertField$Value
transforms.InsertSource.static.field=data_source
transforms.InsertSource.static.value=test-file-source
```

https://kafka.apache.org/documentation.html#connect

Included Transformations

- InsertField - Add a field using either static data or record metadata
- ReplaceField - Filter or rename fields
- MaskField - Replace field with valid null value for the type (0, empty string, etc) or custom replacement
- ValueToKey - Replace the record key with a new key formed from a subset of fields in the record value
- HoistField - Wrap the entire event as a single field inside a Struct or a Map
- ExtractField - Extract a specific field from Struct and Map and include only this field in results
- SetSchemaMetadata - modify the schema name or version
- TimestampRouter - Modify the topic of a record based on original topic and timestamp. Useful when using a sink that needs to write to different tables or indexes based on timestamps
- RegexRouter - modify the topic of a record based on original topic, replacement string and a regular expression
- Filter - Removes messages from all further processing. This is used with a predicate to selectively filter messages.
- InsertHeader - Add a header using static data
- HeadersFrom - Copy or move fields in the key or value to the record headers
- DropHeaders - Remove headers by name

Example: CDC with Debezium connector for MySQL

- MySQL has a binary log (binlog) that records all operations in the order in which they are committed to the database. This includes changes to table schemas as well as changes to the data in tables. MySQL uses the binlog for replication and recovery.
- The Debezium MySQL connector reads the binlog, produces change events for row-level INSERT, UPDATE, and DELETE operations, and emits the change events to Kafka topics. Client applications read those Kafka topics.
- As MySQL is typically set up to purge binlogs after a specified period of time, the MySQL connector performs an initial consistent snapshot of each of your databases. The MySQL connector reads the binlog from the point at which the snapshot was made.

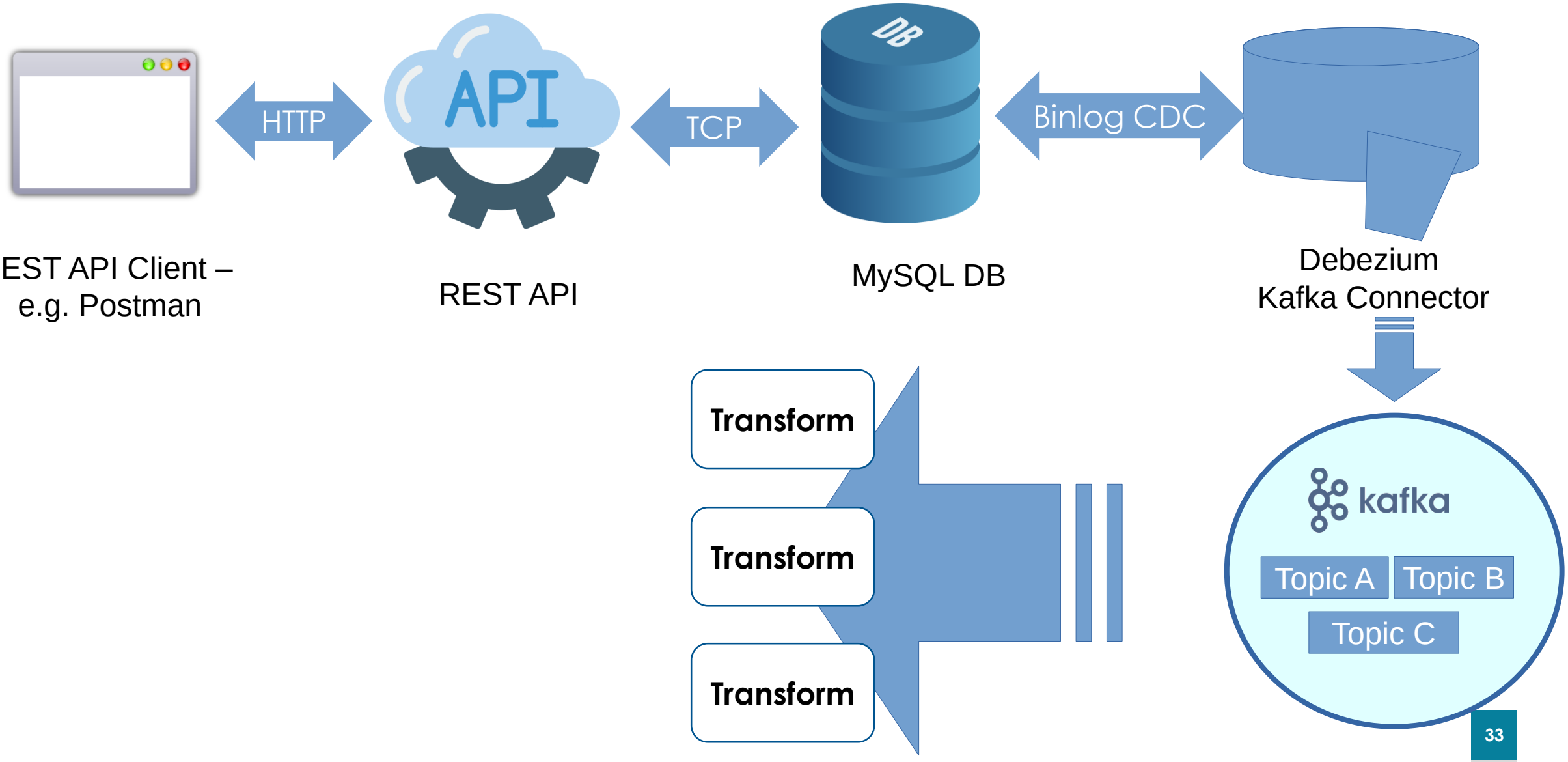
Kafka Connect Simple Example using Debezium Connector

bin\windows\connect-standalone config\connect-standalone.properties D:\CourseKafka\kafka_2.12-3.1.0\connect\debezium-connector-mysql\debezium-mysql-connector.properties

debezium-mysql-connector.properties

```
name=debezium-connector-mysql
bootstrap.servers=localhost:9093
connector.class=io.debezium.connector.mysql.MySqlConnector
tasks.max=1
database.hostname=localhost
database.port=3306
database.user=root
database.password=root
allowPublicKeyRetrieval=True
database.server.id=184054
database.server.name=mysql
database.include.list=spring_blogs_2022
database.history.kafka.bootstrap.servers=localhost:9093
database.history.kafka.topic=dbhistory.mysql
include.schema.changes=true
```

App Demo: REST API + MySQL + Debezium CDC + Kafka



Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD