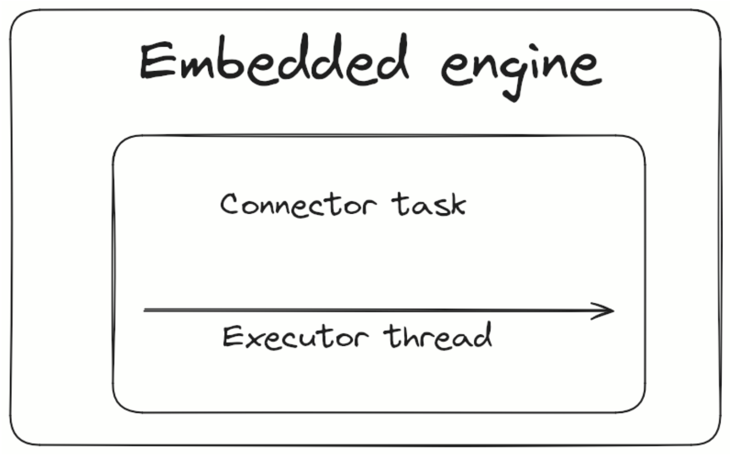
Debezium Asynchronous Engine

Vojtěch Juránek

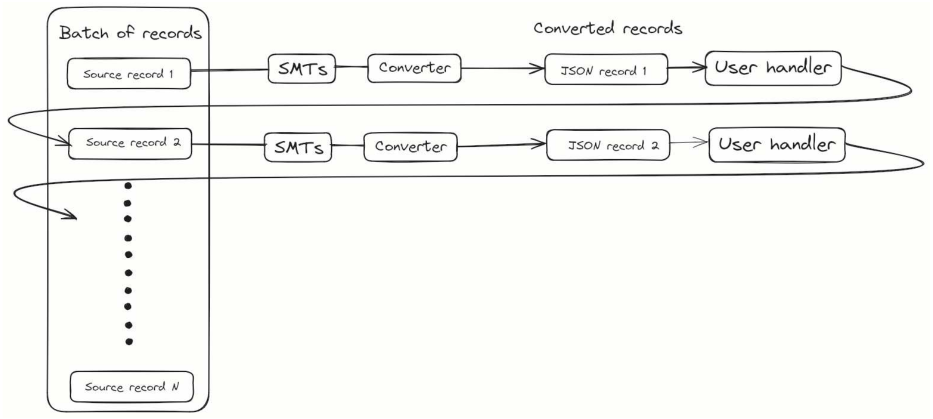
Red Hat

May 21st 2024, Debezium F2F meeting, Brno

Debezium embedded engine



Record processing: embedded engine



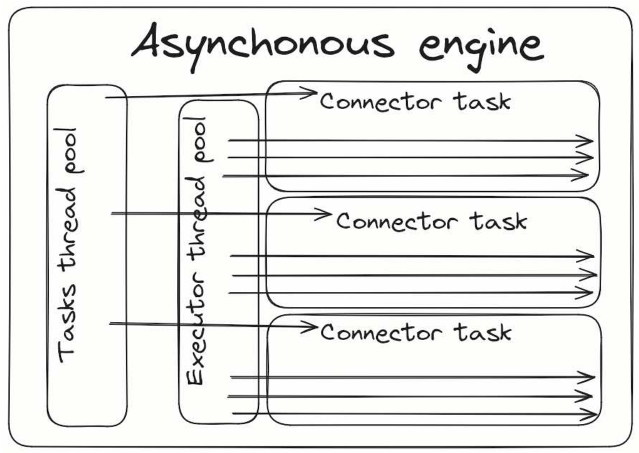
Asynchronous engine goals

- Allow to run multiple source tasks for given connector if the connector provides multiple tasks.
- Run potentially time-consuming code (e.g. event transformation or serialization) in the dedicated threads.
- Allow possible further speedup by optionally disabling total ordering of the messages.
- Be well-prepared for future changes and new features:
- Adjust Debezium testsuite to use `DebeziumEngine` interface instead hardcoded `EmbeddedEngine`.

Asynchronous engine non-goals

- Change `DebeziumEngine` interface.
- Implement any parallelization inside connectors.
- Remove dependency on Kafka Connect API.
- Add support for multiple source connectors or sink connector.

Debezium asynchronous engine



Asynchronous engine

- Just another implementation of `DebeziumEngine` interface
- Creation and APIs are same as for `EmbeddedEngine`, only use different builder factory

```
DebeziumEngine engine = DebeziumEngine.create(
        keyFormat, valueFormat, headerFormat,
       ConvertingAsyncEngineBuilderFactory.class.getName
           ())
       .using(props)
       .notifying(consumer)
       .build();
```

Async engine specific configuration options

- `record.processing.threads` - number of threads to be used for processing CDC records.
- `record.processing.shutdown.timeout.ms` - maximum time in milliseconds to wait for processing submitted records when task shutdown is called.
- `record.processing.order` - determines how the records should be produced (`ORDERED`, `UNORDERED`).
- `record.processing.with.serial.consumer` - specifies whether the default ChangeConsumer should be created from provided Consumer, resulting in serial Consumer processing.
- `task.management.timeout.ms` - time to wait for task's lifecycle management operations (starting and stopping).

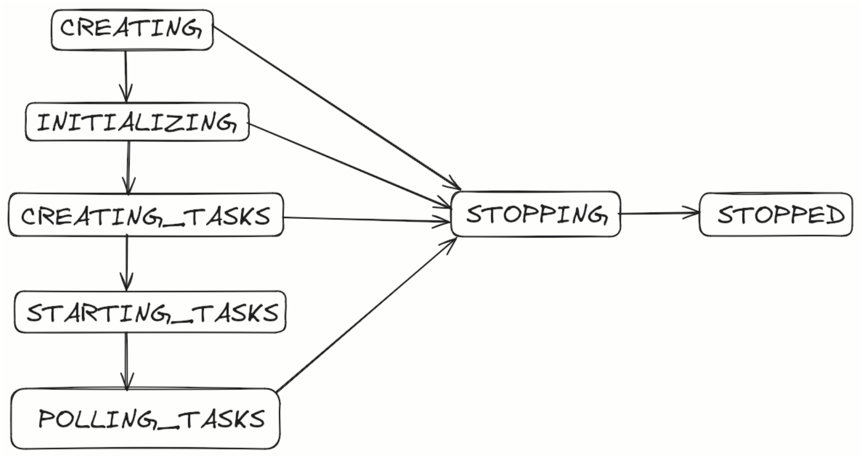
`AsyncEngine` internals

Engine states

`AsyncEngine` lifecycle pahses and states:

- `CREATING` - engine object is being created or was already created, but `run()` method wasn't called yet
- `INITIALIZING` - initializing the connector
- `CREATING_TASKS` - creating connector tasks
- `STARTING_TASKS` - starting connector tasks
- `POLLING_TASKS` - running tasks polling, this is the main phase when the data are produced
- `STOPPING` - the engine is being stopped
- `STOPPED` - engine has been stopped, any call on engine object in this state should fail

Debezium asynchronous engine

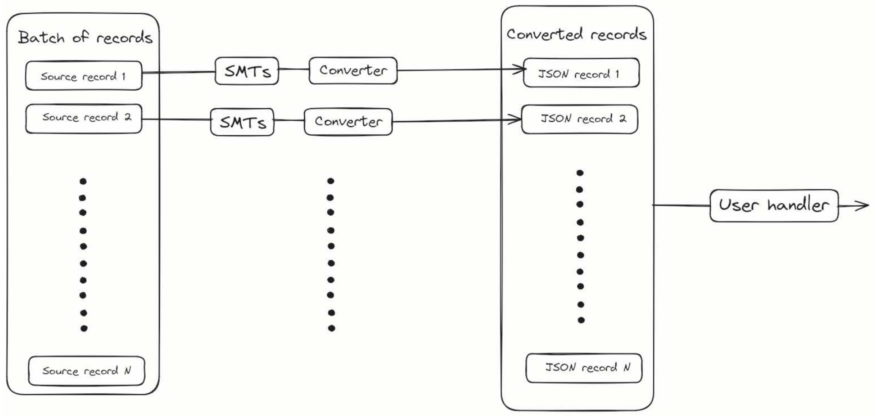


Record processor

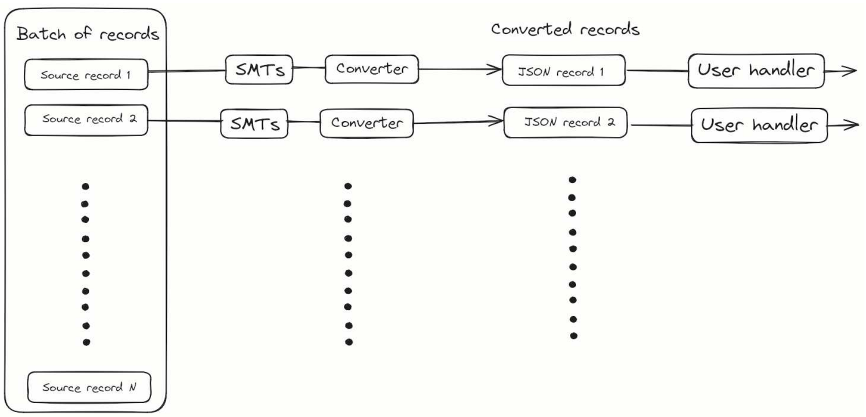
```java
@Incubating
public interface RecordProcessor<R> {
    void initialize(
        final ExecutorService recordService,
        final Transformations transformations,
        final Function<SourceRecord, R> serializer,
        final RecordCommitter committer);

    void processRecords(final List<SourceRecord> records)
         throws InterruptedException;
}
```

Record processing: async. engine ordered



Record processing: async. engine unordered



Auxiliary interfaces and classes

```java
@Incubating
public interface DebeziumSourceConnector {
    DebeziumSourceConnectorContext context();
    void initialize(DebeziumSourceConnectorContext
        context);
}
```

```java
@Incubating
public interface DebeziumSourceConnectorContext {
    OffsetBackingStore offsetStore();
    OffsetStorageReader offsetStorageReader();
    OffsetStorageWriter offsetStorageWriter();
}
```

and more. See
`debezium-embedded/src/main/java/io/debezium/engine/source/`

Code walk through

Future

- gRPC
- virtual threads - switching just few lines of code
- Quarkus integration

Resources

- DDD-7: Asynchronous Debezium Embedded Engine
- Discussion under DDD-7 PR
- DBZ-7024 - main async. engine tracking Jira
- Other possible interesting Jiras:DBZ-7764, DBZ-7777

Questions?

Thank you!

Questions?