Reactive Messaging RedisStream

Table of Contents

This project provides a **Redis Stream**-based **SmallRye Reactive Messaging** connector that enables the use of the **MicroProfile Reactive Messaging API** over Redis Stream. The implementation allows seamless integration of asynchronous data streams into WildFly and Quarkus-based applications.

For more details on **MicroProfile Reactive Messaging**, check:

MicroProfile Reactive Messaging Specification

For information on **SmallRye Reactive Messaging**, visit:

SmallRye Reactive Messaging

1. Modules

The project consists of the following modules:

- `quarkus-reactive-messaging-redisstream-extension-sample`: A sample application demonstrating the usage of the Redis Streams extension.
- `quarkus-reactive-messaging-redisstream-extension-parent`: A custom Quarkus extension that integrates Redis Streams as a reactive messaging connector, utilizing the core Quarkus Redis extension for managing Redis connections.
- `reactive-messaging-redisstream-connector`: Provides the core connector logic to integrate Redis Streams API with Reactive Messaging Specification. It does not directly build on any specific Redis library, but rather provides an interface that conforms to the Redis Streams commands.

2. Connector overview

The connector core module aims to integrate the Redis Streams API with the Reactive Messaging Specification without depending on a specific Redis SDK.

2.1. Consuming Messages

The inbound connector reads messages from a Redis Stream as a consumer group using the `XREADGROUP` command.

2.1.1. Consumer Group Setup

- If the specified consumer group does not exist, it is automatically created.
- Each connector instance uses a unique consumer ID (UUID) to track message processing.

2.1.2. Incoming Message

Incoming stream entries are converted to `org.eclipse.microprofile.reactive.messaging.Message` instances with a String payload and `IncomingRedisStreamMetadata` metadata.

- The payload is extracted from the stream entries' `message` field by default.
  - The field can be configured via the `payload-field` configuration property (Optional).
- `IncomingRedisStreamMetadata` consists of:
  - The stream key from which the message was read.

    - Stream entry ID (as generated on Redis).
    - All additional fields (excluding the payload-field) in a map.

2.1.3. Message Processing

- Stream entries are acknowledged on Redis using (`XACK`) if the `org.eclipse.microprofile.reactive.messaging.Message` has been acked as per the Microprofile Reactive Messaging specification. You can use the `@Acknowledgment` annotation to control the acknowledgment behavior.
- Expired messages (based on the `ttl` field) are automatically acknowledged and skipped.
- `XREADGROUP` commands are retried with exponential backoff (1s to 30s) on processing failures.

Example Configuration

```
mp.messaging.incoming.my-channel.connector=reactive-messaging-redis-streams ①
mp.messaging.incoming.my-channel.stream-key=mystream ②
mp.messaging.incoming.my-channel.group=mygroup ③
mp.messaging.incoming.my-channel.payload-field=message ④
mp.messaging.incoming.my-channel.xread-count=10 ⑤
mp.messaging.incoming.my-channel.xread-block-ms=10000 ⑥
mp.messaging.incoming.in-reactive.xread-noack=false ⑦
```

① Activate the redis stream connector for the incoming channel called `my-channel`.

② The stream key on redis to read messages from.

③ The consumer group name.

④ The field name to extract the payload from the stream entry.

⑤ The number of messages to read in a single `XREADGROUP` call. The `COUNT` parameter of the `XREADGROUP` command.

⑥ The milliseconds to block during `XREADGROUP` calls. The `BLOCK` parameter of the `XREADGROUP` command.

⑦ Include `NOACK` option in the `XREADGROUP` calls.

2.2. Producing Messages

The outbound connector publishes messages to a Redis Stream using `XADD`.

2.2.1. Outgoing Message

Outgoing `org.eclipse.microprofile.reactive.messaging.Message` instances with a String payload are converted to stream entries.

- The payload is sent as the stream entries' `message` field by default.
    - The field can be configured via the `payload-field` configuration property (Optional).
- Additional fields can be added via `RedisStreamMetadata`.

```
@Outgoing("out-channel")
public Message<String> produce() {
    return Message.of("Hello World!")
        .addMetadata(new RedisStreamMetadata()
                            .withAdditionalField("additionalKey", "additional value")
        );
//    The produced stream entry will be like:
//      1) 1) "out-stream"
//         2) 1) 1) "1739262584638-0"
//            2) 1) "message"
//               2) "Hello World!"
//               3) "additionalKey"
//               4) "Test-additional value"
}
```

2.2.2. Stream Trimming

- Use `xadd-maxlen` to trim by entry count (`MAXLEN` parameter of the `XADD` command).
  - Exact trimming can be enabled by setting `xadd-exact-maxlen` to `true` (defaults to false since almost exact is more efficient).
- Use `xadd-ttl-ms` to compute an expiration timestamp
  - Adds a `ttl` field with a value of (current epoch time + TTL), in order to consumers be able to skip expired messages.
  - It also sets a minimum ID for stream trimming (`MINID` parameter of the `XADD` command).
- If both are set, `xadd-maxlen` takes precedence.

Example Configuration

```
mp.messaging.outgoing.out-channel.connector=reactive-messaging-redis-streams ①
mp.messaging.outgoing.out-channel.stream-key=mystream ②
mp.messaging.incoming.out-channel.payload-field=message ③
mp.messaging.outgoing.out-channel.xadd-maxlen=1000 ④
mp.messaging.outgoing.out-channel.xadd-exact-maxlen=true ⑤
#mp.messaging.outgoing.out-channel.xadd-ttl-ms=1000 ⑥
```

① Activate the redis stream connector for the incoming channel called `my-channel`.

② The stream key on redis to read messages from.

③ The field name to use as the payload in the stream entry.

④ The maximum number of entries to keep in the stream.

⑤ Enable exact trimming by entry count.

⑥ Possible trimming based on milliseconds to set the TTL for the stream entry.

2.3. Graceful Shutdown

On shutdown:

- New message consumption stops immediately.
- In-flight messages are given up to `graceful-timeout-ms` (default: 60000ms) to complete.
- Redis connections are closed after timeout or all messages are processed.

Configure the timeout via:

```
mp.messaging.connector.reactive-messaging-redis-streams.graceful-timeout-ms=30000
```

3. Configuration

The connector is identified by `reactive-messaging-redis-streams`. Below are the configuration attributes available for both inbound and outbound channels.

Attribute	Description	Default	Mandatory	Direction
`connection-key`	The Redis connection key to use. Can be implementation specific in case of the quarkus extension it is the key used to define the redis connection according to the quarkus redis client. For example if you use `quarkus.redis.my-redis.hosts` configuration then you can use `my-redis` as connection key.	`default`	No	INCOMING_AND_OUTGOING
`stream-key`	The Redis key holding the stream items.	-	Yes	INCOMING_AND_OUTGOING
`payload-field`	The stream entry field name containing the message payload.	`message`	No	INCOMING_AND_OUTGOING
`group`	The consumer group of the Redis stream to read from.	-	Yes	INCOMING
`xread-count`	The maximum number of entries to receive per `XREADGROUP` call.	`1`	No	INCOMING
`xread-block-ms`	The milliseconds to block during `XREADGROUP` calls.	`5000`	No	INCOMING
`xread-noack`	Include the `NOACK` parameter in the `XREADGROUP` call.	`true`	No	INCOMING

Attribute	Description	Default	Mandatory	Direction
broadcast	Allow the received entries to be consumed by multiple channels.	false	No	INCOMING
xadd-maxlen	The maximum number of entries to keep in the stream (trims old entries).	-	No	OUTGOING
xadd-exact-maxlen	Use exact trimming for MAXLEN (requires Redis 6.2+).	false	No	OUTGOING
xadd-ttl-ms	Milliseconds to keep an entry in the stream (uses minid trimming).	-	No	OUTGOING
wait-for-write-completion	Whether the Redis client waits for the XADD to respond before acknowledging the message.	true	No	OUTGOING
max-inflight-messages	The maximum number of messages to be written to the RedisStream concurrently. You can set this attribute to 0 remove the limit.	1024	No	OUTGOING
retries	The maximum number of retries for sending messages to the Redis stream. If the value is set to 0, no retries will be performed. If set to negative number, it will retry indefinitely.	3	No	OUTGOING
retry-initial-delay-ms	The initial delay for the retry.	1000	No	OUTGOING
retry-max-delay-ms	The maximum delay for the retry.	10000	No	OUTGOING
graceful-timeout-ms	Milliseconds to wait for the consumed messages to finish processing before closing the consumer group.	60000	No	CONNECTOR

4. Examples

4.1. Consuming Messages

Example usage of the Redis Streams connector for consuming messages from a Redis Stream.

4.1.1. Pre-requisites:

- Redis server running on localhost:6379 or dev-container from Redis extension.

4.1.2. Set up quarkus

Add the following dependency to your project:

pom xml

```
<dependency>
    <groupId>hu.icellmobilsoft.quarkus.extension</groupId>
    <artifactId>quarkus-redisstream-extension</artifactId>
</dependency>
```

4.1.3. Configure the connector

MicroProfile configuration

```
mp:
  messaging:
    incoming:
      in-channel: ①
        connector: reactive-messaging-redis-streams ②
        stream-key: mystream ③
        group: mygroup ④
        connection-key: my-redis-connection ⑤
        payload-field: message #optional defaults to 'message'
        xread-count: 10 #optional defaults to '1'
        xread-block-ms: 10000 #optional defaults to '5000'
        xread-noack: false #optional defaults to 'true'
    connector:
      reactive-messaging-redis-streams:
        graceful-timeout-ms: 10000 #optional defaults to '60000' ⑥
quarkus:
  redis:
    my-redis-connection: ⑤
      hosts: redis://localhost:6379
```

① The incoming MP channel name.

② Specify the connector to use.

③ The Redis stream key to read messages from.

④ The consumer group name.

⑤ The Redis connection reference to use.

⑥ Connector specific config to set the graceful timeout.

4.1.4. Implement the consumer

Blocking implementation

Blocking consumer method

```
@Incoming("in-channel") ①
//@Blocking(ordered = false, value = "incoming-pool") ②
//@Retry(maxRetries = 2) ③
```

```
public void process(String message) {
// Process the message
}
```

① The incoming MP channel name.

② Optional: Use the `@Blocking` annotation to specify parallel processing with a custom thread pool. Pool size can be set via the `smallrye.messaging.worker.incoming-pool.max-concurrency` configuration key.

③ Optional: Use the `@Retry` annotation from MP Fault Tolerance to control method retry behavior.

Reactive implementation

Reactive consumer method

```
@Incoming("in-channel") ①
    public Uni<Void> consumeReactive(Message<String> message) {
        return Uni.createFrom()
                .item(message)
                .invoke(this::processMessage) ②
                .replaceWithVoid();
    }

    private void processMessage(Message<String> message) {
        // process the message
        message.getMetadata()
                .get(IncomingRedisStreamMetadata.class)
                .ifPresent(this::processMetadata); ③
        message.ack(); ④
    }

    private void processMetadata(IncomingRedisStreamMetadata metadata) { ③
            // process metadata
    }
```

① The incoming MP channel name.

② Process the message reactively.

③ Example on how to process the metadata.

④ Acknowledge the message manually.

4.2. Producing Messages

Example usage of the Redis Streams connector for producing messages to a Redis Stream.

4.2.1. Pre-requisites:

- Redis server running on localhost:6379 or dev-container from Redis extension.

4.2.2. Set up quarkus

Add the following dependency to your project:

pom xml

```
<dependency>
    <groupId>hu.icellmobilsoft.quarkus.extension</groupId>
    <artifactId>quarkus-redisstream-extension</artifactId>
</dependency>
```

4.2.3. Configure the connector

MicroProfile configuration

```
mp:
  messaging:
    outgoing:
      out-channel: ①
        connector: reactive-messaging-redis-streams ②
        stream-key: mystream ③
        connection-key: my-redis-connection ④
        payload-field: message #optional defaults to 'message'
#        xadd-maxlen: 10 #optional
#        xadd-exact-maxlen: true #optional defaults to 'false'
        xadd-ttl-ms: 10000 #optional
quarkus:
  redis:
    my-redis-connection: ④
      hosts: redis://localhost:6379
```

① The outgoing MP channel name.

② Specify the connector to use.

③ The Redis stream key to read messages from.

④ The Redis connection reference to use.

4.2.4. Implement the Producer

Simple producer

Emitter based producer with metadata

```
    @Inject
    @Channel("out-channel") ①
    Emitter<String> emitter;


    public void produceWithMetadata() {
```

```
        emitter.send("Hello"); ②
    }
```

① The incoming MP channel name.

② The message payload

The resulting message will be like:

```
 1) 1) "mystream"
    2) 1) 1) "1739262584638-0"
          2) 1) "message"
             2) "Hello"
             3) "ttl"
             4) "1739262594638"
```

With custom metadata

Emitter based producer with metadata

```
    @Inject
    @Channel("out-channel") ①
    Emitter<String> emitter;


    public void produceWithMetadata() {
        emitter.send(
                ContextAwareMessage.of("Hello") ②
                        .addMetadata(new RedisStreamMetadata()
                            .withAdditionalField("otherKey", "Other value") ③
        ));
    }
```

① The incoming MP channel name.

② The message payload

③ Additional fields

The resulting message will be like:

```
 1) 1) "mystream"
    2) 1) 1) "1739262584638-0"
          2) 1) "message"
             2) "Hello"
             3) "otherKey"
             4) "Other value"
             5) "ttl"
             6) "1739262594638"
```

5. Migration Description

Next section describes the changes between releases.

5.1. 1.0.0 → 1.1.0

5.1.1. quarkus-reactive-messaging-redisstream-additional-fields module

- Introduced the new `quarkus-reactive-messaging-redisstream-additional-fields` module for MDC support (i.e. `extSessionId`) in the redis stream additional fields with overridable methods.

Migration

Changes are backwards compatible doesn't need any migration.

5.1.2. Changes

- OSS Sonatype → Maven Central repository migration

Migration

Changes are backwards compatible doesn't need any migration.

5.2. 1.1.0 → 1.2.0

5.2.1. reactive-messaging-redisstream-connector

- RedisStreamsConnector#prudentRun handling fix, separate handling of the flag per config.
- Consume messages on vert.x context to retain context data in all cases.
- Added `broadcast` incoming configuration to allow multiple consumers on the same channel (ie. multiple methods with `@Incoming` for the same channel).

Migration

Changes are backwards compatible doesn't need any migration.

5.2.2. quarkus-reactive-messaging-redisstream-extension-sample

- Added both imperative and reactive example for `@Incoming` usage.

Migration

Changes are backwards compatible doesn't need any migration.

5.3. 1.2.0 → 1.3.0

5.3.1. reactive-messaging-redisstream-connector

- Introduced additional config options for outgoing producers in order to fine-tune backpressure and handle fire-and-forget scenarios.

Migration

- The connector now defaults to 3 retries for outgoing messages. If you want to keep the previous behavior of no retries, you need to set `retries` to `0` in your configuration.