Spring Integration: Using Channel Adapters to Integrate with External Systems

INTEGRATING WITH DATABASES

Steven Haines
PRINCIPAL SOFTWARE ARCHITECT

@geekcap www.geekcap.com

Overview

Overview of Database Integrations

MariaDB (MySQL)

MongoDB

Inbound and Outbound Channel Adapters

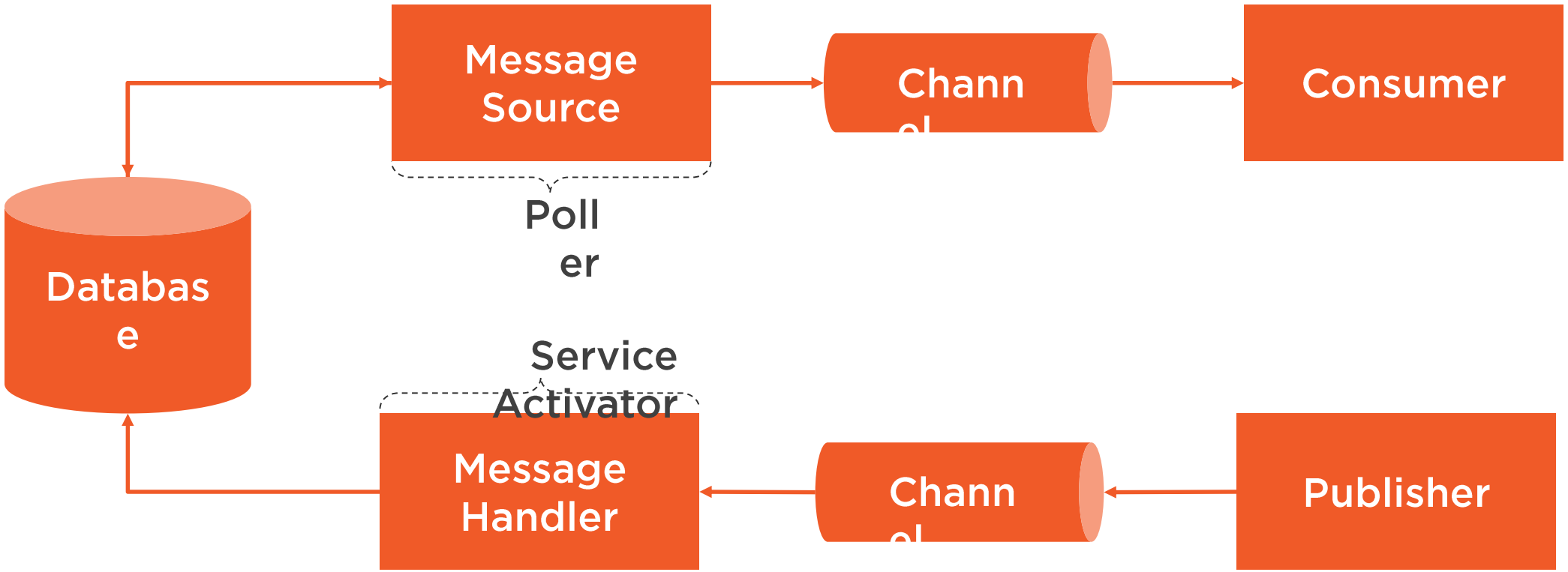
Inbound Channel Adapter

MessageSource

Outbound Channel Adapter

MessageHandler

Database Inbound and Outbound Adapters



MariaDB (JDBC)

MariaDB

MariaDB is a fork of MySQL that is intended to reman free and open-source software under the GNU General Public License. It was forked due to concerns of the acquisition of MySQL by Oracle and is a drop-in replacement for MySQL and its APIs.

Inbound and Outbound Channel Adapters

Inbound Channel Adapter

JdbcPollingChannelAdapter

Outbound Channel Adapter

JdbcMessageHandler

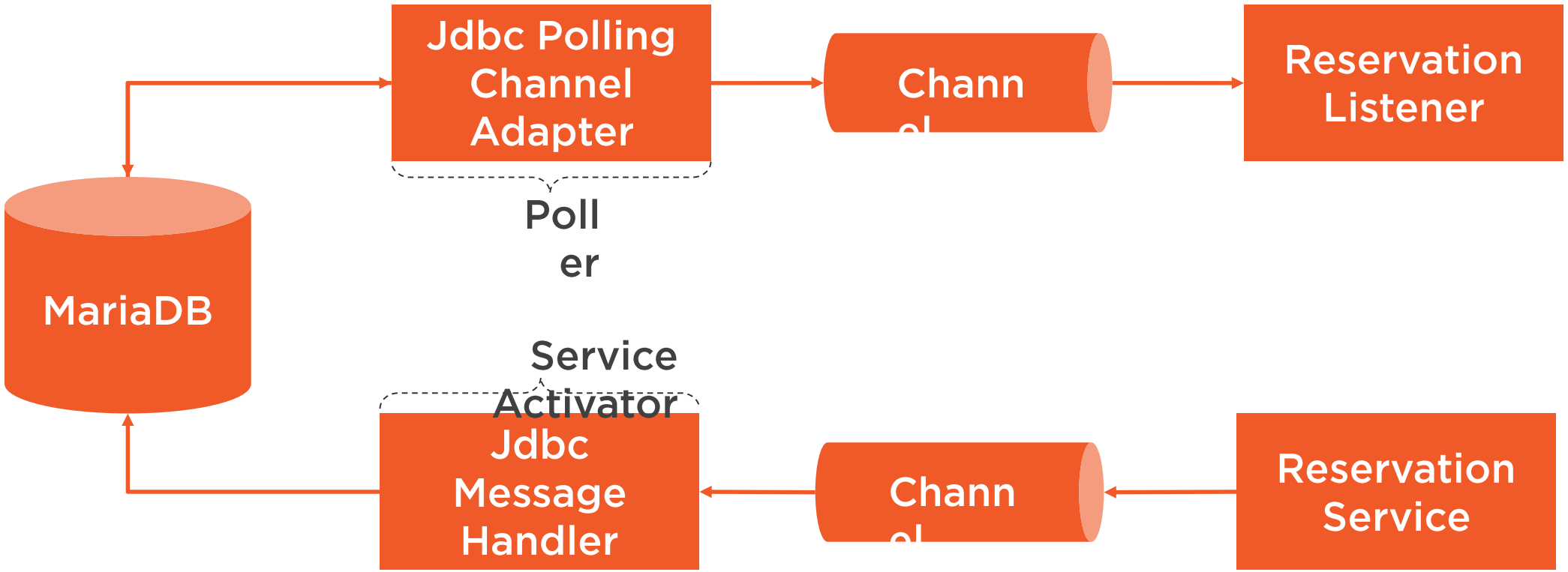
```
@Bean
@InboundChannelAdapter(
                      value =
"newReservationListChannel",
                      poller =
@Poller(fixedDelay="1000"))
public MessageSource<?> inbound(DataSource
dataSource) {
    JdbcPollingChannelAdapter adapter =
                    new
JdbcPollingChannelAdapter(dataSource,
                     "SELECT * FROM reservation
where status = 0");
    adapter.setRowMapper((rs, index) ->
          new Reservation(rs.getLong("id"),
rs.getString("name")));
    adapter.setUpdateSql(
           “UPDATE reservation SET status = 1
where id in (:id)");
    return adapter;
}

@Splitter(inputChannel =
"newReservationListChannel",
                outputChannel =
"newReservationChannel")
```

- Create an InboundChannelAdapter with a 1-second poller
- Create a JdbcPollingChannelAdapter
- Add an update SQL to update the records that we processed
- Create a Splitter that splits the list of message records into individual reservation messages

```
@Bean
@ServiceActivator(inputChannel =
“inputChannel")
public MessageHandler handler(DataSource
dataSource) {

    JdbcMessageHandler jdbcMessageHandler =
        new JdbcMessageHandler(dataSource,
            "INSERT INTO reservation (id,
name, status)
             VALUES (?, ?, 0)");

jdbcMessageHandler.setPreparedStatementSetter(
            (ps, message) -> {
                Reservation reservation =

(Reservation) message.getPayload();
                ps.setLong(1,
reservation.getId());
                ps.setString(2,
reservation.getName());
            });
    return jdbcMessageHandler;

}
```

- Create a ServiceActivator
- Create a JdbcMessageHandler
- INSERT SQL statement
- Define the PreparedStatement values from the reservation

JDBC Inbound and Outbound Adapters



```
docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=password mariadb

docker exec <container-id> mysql -u root -ppassword -e "create database
reservationdb"
```

Running MariaDB in Docker

Official MariaDB Docker Image: mariadb

MariaDB DockerHub Link: https://hub.docker.com/_/mariadb

Note: we have to create the database from the command line

Demo

Build our application

- Configuration
- Inbound Channel Adapter and Reservation Listener
- Outbound Channel Adapter and Reservation Service

Write Reservations to MariaDB

Read Reservation from MariaDB

Summary

JdbcPollingChannelAdapter

JdbcMessageHandler

Next up: MongoDB

MongoDB (NoSQL)

MongoDB

MongoDB is an open-source, documented-based, distributed database built for modern developers and for the cloud era

Channel Adapters and Gateways

Inbound Channel Adapter

MongoDbMessageSource

Outbound Channel Adapter

MongoDbStoringMessageHandler

Outbound Gateway

MongoDbOutboundGateway

```
@Bean
@InboundChannelAdapter(
         value =
"reservationListFromMongoChannel",
         poller = @Poller(fixedDelay="3000"))
public MessageSource
              mongoMessageSource(MongoTemplate
template) {
    MongoDbMessageSource messageSource =
                    new
MongoDbMessageSource(template,
                                new
LiteralExpression("{'status' : 'None'}"));

    messageSource.setCollectionNameExpression(
                                    new
LiteralExpression("reservations"));


messageSource.setEntityClass(Reservation.class
);
    return messageSource;
}

@Splitter(
         inputChannel =
```

- Create an InboundChannelAdapter with a 1-second poller
- Create a MongoDbMessageSource with a match String
- Define the MongoDB Collection Name
- Define the class to which to map the results
- Split the list of messages into individual messages

```
@Bean
@ServiceActivator(inputChannel =
“toMongoChannel")
public MessageHandler mongoMessageHandler(

MongoTemplate template) {

    MongoDbStoringMessageHandler handler =
                    new
MongoDbStoringMessageHandler(template);

    handler.setCollectionNameExpression(
                                    new
LiteralExpression("reservations"));

    return handler;
}
```

- Create a MessageHandler with a ServiceActivator
- Create a MongoDbStoringMessageHandler
- Set the collection name

```
@Bean
@ServiceActivator(inputChannel =
"getReservationChannel")
public MessageHandler mongoDbOutboundGateway(

MongoTemplate template) {

    MongoDbOutboundGateway gateway =
                        new
MongoDbOutboundGateway(template);

gateway.setCollectionNameExpressionString("'rese
rvations'");
    gateway.setQueryExpressionString("payload");
    gateway.setExpectSingleResult(true);
    gateway.setEntityClass(Reservation.class);
    gateway.setOutputChannelName(

"getReservationReplyChannel");
    return gateway;
}

 @MessagingGateway(
           defaultRequestChannel =
"getReservationChannel",
           defaultReplyChannel =
```

- Create a MessageHandler with a ServiceActivator

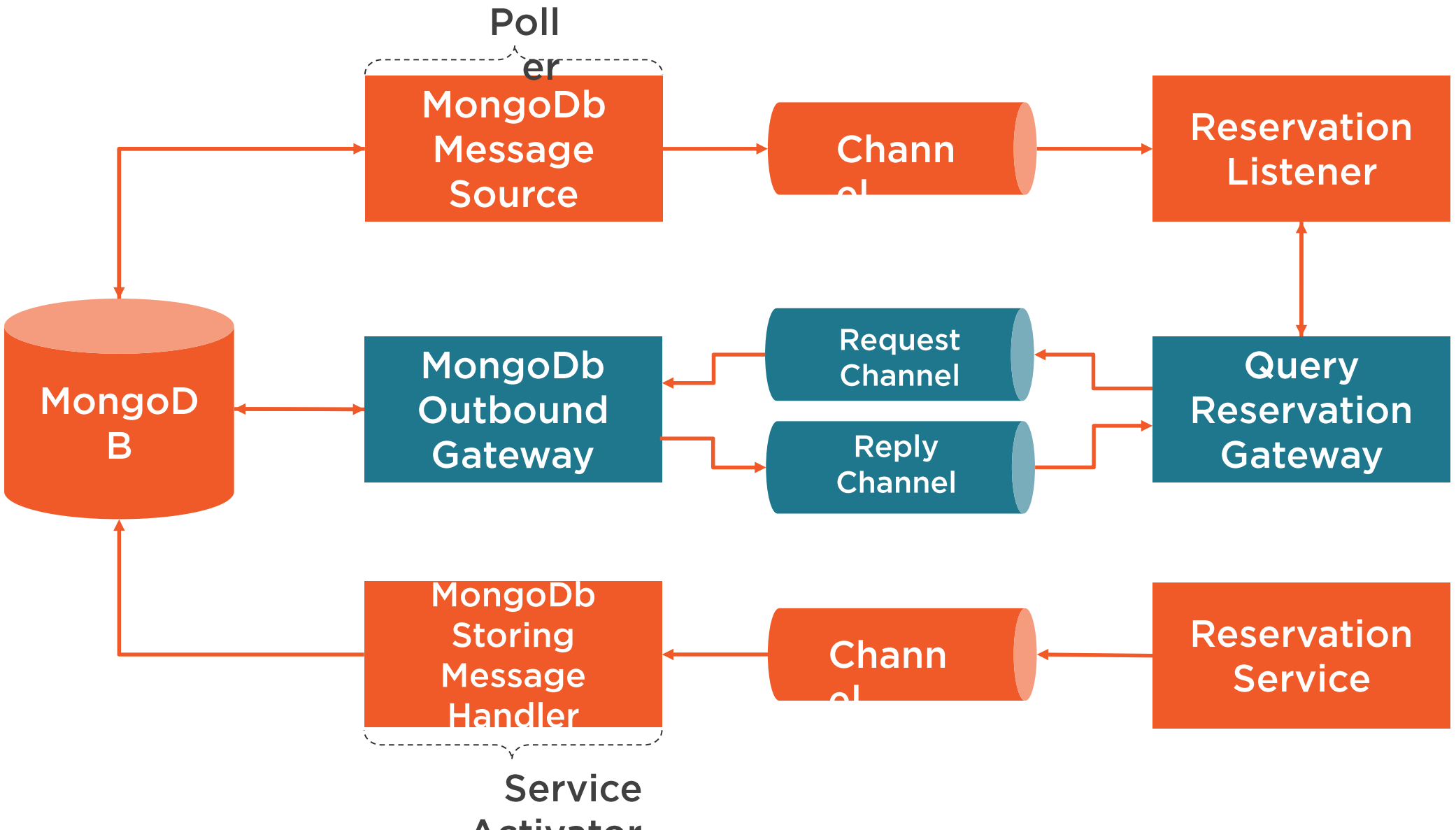
- Create a MongoDbOutboundGateway

- Set the collection name
- Use the message payload as our query expression

- Deserialize the record into a Reservation
- Publish the results to the getReservationReplyChannel

- Create a MessagingGateway that publishes messages to the request channel and receives the response on the reply channel

MongoDB Adapters and Gateway



```
docker run -d -p 27017:27017 mongo
```

Running MongoDB in Docker

Official MongoDB Docker Image: mongo

MongoDB DockerHub Link: https://hub.docker.com/_/mongo

Demo

Build our application

- Inbound Channel Adapter
- Outbound Channel Adapter
- Outbound Gateway

Run the application

Validate the results

Summary

MongoDbMessageSource

MongoDbStoringMessageHandler

MongoDbOutboundGateway

Next up: Module Wrap-up

Conclusion

Database Integrations

MariaDB

JDBC

MongoDB

NoSQL

Inbound and Outbound Channel Adapters

Inbound Channel Adapter

MessageSource

Outbound Channel Adapter

MessageHandler

Summary

You should understand how to integrate with relational databases using JDBC

You should understand how to integrate with MongoDB

You should understand the Spring Integration model for integrating with databases

Next Module: Integrating with RESTful Web Services