**Using Spring Support for distributed JTA transactions across multiple XA resources without Java EE application server**

Spring Boot supports distributed JTA transactions across multiple XA resources by using either an Atomikos or Bitronix embedded transaction manager.

When a JTA environment is detected, Spring’s JtaTransactionManager is used to manage transactions. Auto-configured JMS, DataSource, and JPA beans are upgraded to support XA transactions. We can use standard Spring idioms, such as @Transactional, to participate in a distributed transaction.

Atomikos is a popular open source transaction manager that is designed to easily automate exception handling by canceling all inappropriately performed steps of a failing or failed transaction, even across different databases or systems.

In this post, we will use Atomikos light weight embedded transaction manager to support transactions across multiple databases in a self-contained cloud native application.

Here are the steps: -

1. Define project dependencies.
2. Define data model.
3. Add datasource configurations to the properties file.
4. Create data repositories.
5. Provide necessary configurations for JTA transaction manager.
6. Integrate with Atomikos.
7. Configure datasources and data repositories.
8. Create a Service that saves information into multiple databases in a transaction.
9. Write a Rest Controller to test the distributed transactions.
10. **Project Dependencies**

<dependency>

<groupId>com.atomikos</groupId>

<artifactId>transactions</artifactId>

<version>3.9.3</version>

</dependency>

<dependency>

<groupId>com.atomikos</groupId>

<artifactId>transactions-jta</artifactId>

<version>3.9.3</version>

</dependency>

<dependency>

<groupId>com.atomikos</groupId>

<artifactId>transactions-hibernate3</artifactId>

<version>3.9.3</version>

<exclusions>

<exclusion>

<artifactId>hibernate</artifactId>

<groupId>org.hibernate</groupId>

</exclusion>

</exclusions>

</dependency>

1. **Define data model**

Our data model comprises of an EventMessage, and an EventFuture as follows: -

@Entity

@Table(name = "eventmessage")

**public** **class** EventMessage {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

**private** Integer id;

@Column(name = "description", nullable = **false**, length=255)

**private** String description;

@Entity

@Table(name = "eventfuture")

**public** **class** EventFuture {

@Id

**private** Integer id;

@Column(name = "action", nullable = **false**, length=10)

**private** String action;

We would save both the entities into two separate databases.

1. **Add datasource configurations to the properties file**

eventmessage.datasource.url=jdbc:h2:mem:eventmessage

eventmessage.datasource.username=sa

eventmessage.datasource.password=

eventfuture.datasource.url=jdbc:h2:mem:eventfuture

eventfuture.datasource.username=sa

eventfuture.datasource.password=

1. **Create data repositories**

**public** **interface** EventMessageRepository **extends** JpaRepository<EventMessage, Integer> {

}

**public** **interface** EventFutureRepository **extends** JpaRepository<EventFuture, Integer> {

}

1. **Provide necessary configurations for JTA transaction manager**

@Configuration

@ComponentScan

@EnableTransactionManagement

**public** **class** ApplicationConfig {

@Bean

**public** PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {

**return** **new** PropertySourcesPlaceholderConfigurer();

}

@Bean

**public** JpaVendorAdapter jpaVendorAdapter() {

HibernateJpaVendorAdapter hibernateJpaVendorAdapter = **new** HibernateJpaVendorAdapter();

hibernateJpaVendorAdapter.setShowSql(**true**);

hibernateJpaVendorAdapter.setGenerateDdl(**true**);

hibernateJpaVendorAdapter.setDatabase(Database.***H2***);

**return** hibernateJpaVendorAdapter;

}

@Bean(name = "userTransaction")

**public** UserTransaction userTransaction() **throws** Throwable {

UserTransactionImp userTransactionImp = **new** UserTransactionImp();

userTransactionImp.setTransactionTimeout(10000);

**return** userTransactionImp;

}

@Bean(name = "atomikosTransactionManager", initMethod = "init", destroyMethod = "close")

**public** TransactionManager atomikosTransactionManager() **throws** Throwable {

UserTransactionManager userTransactionManager = **new** UserTransactionManager();

userTransactionManager.setForceShutdown(**false**);

AtomikosJtaPlatform.*transactionManager* = userTransactionManager;

**return** userTransactionManager;

}

@Bean(name = "transactionManager")

@DependsOn({ "userTransaction", "atomikosTransactionManager" })

**public** PlatformTransactionManager transactionManager() **throws** Throwable {

UserTransaction userTransaction = userTransaction();

AtomikosJtaPlatform.*transaction* = userTransaction;

TransactionManager atomikosTransactionManager = atomikosTransactionManager();

**return** **new** JtaTransactionManager(userTransaction, atomikosTransactionManager);

}

}

1. **Integrate with Atomikos**

**public** **class** AtomikosJtaPlatform **extends** AbstractJtaPlatform {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**static** TransactionManager *transactionManager*;

**static** UserTransaction *transaction*;

@Override

**protected** TransactionManager locateTransactionManager() {

**return** *transactionManager*;

}

@Override

**protected** UserTransaction locateUserTransaction() {

**return** *transaction*;

}

}

1. **Configure repository and data sources**

@Configuration

@DependsOn("transactionManager")

@EnableJpaRepositories(basePackages = "com.example.repository.eventMessage", entityManagerFactoryRef = "eventMessageEntityManager", transactionManagerRef = "transactionManager")

@EnableConfigurationProperties(EventMessageDatasourceProperties.**class**)

**public** **class** EventMessageConfig {

@Autowired

**private** JpaVendorAdapter jpaVendorAdapter;

@Autowired

**private** EventMessageDatasourceProperties eventMessageDatasourceProperties;

@Primary

@Bean(name = "eventMessageDataSource", initMethod = "init", destroyMethod = "close")

**public** DataSource eventMessageDataSource() {

AtomikosDataSourceBean dataSource = **new** AtomikosDataSourceBean();

dataSource.setUniqueResourceName("xadseventmessage");

dataSource.setXaDataSourceClassName("org.h2.jdbcx.JdbcDataSource");

Properties xaProperties = **new** Properties();

xaProperties.put("url", eventMessageDatasourceProperties.getUrl());

xaProperties.put("user", eventMessageDatasourceProperties.getUsername());

xaProperties.put("password", eventMessageDatasourceProperties.getPassword());

dataSource.setXaProperties(xaProperties);

dataSource.setPoolSize(10);

**return** dataSource;

}

@Primary

@Bean(name = "eventMessageEntityManager")

@DependsOn("transactionManager")

**public** LocalContainerEntityManagerFactoryBean eventMessageEntityManager() **throws** Throwable {

HashMap<String, Object> properties = **new** HashMap<String, Object>();

properties.put("hibernate.transaction.jta.platform", AtomikosJtaPlatform.**class**.getName());

properties.put("javax.persistence.transactionType", "JTA");

LocalContainerEntityManagerFactoryBean entityManager = **new** LocalContainerEntityManagerFactoryBean();

entityManager.setJtaDataSource(eventMessageDataSource());

entityManager.setJpaVendorAdapter(jpaVendorAdapter);

entityManager.setPackagesToScan("com.example.domain.eventMessage");

entityManager.setPersistenceUnitName("eventMessagePersistenceUnit");

entityManager.setJpaPropertyMap(properties);

**return** entityManager;

}

}

package com.example;

import java.util.HashMap;

import java.util.Properties;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.context.properties.EnableConfigurationProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.DependsOn;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.orm.jpa.JpaVendorAdapter;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

import com.atomikos.jdbc.AtomikosDataSourceBean;

import com.example.repository.eventFuture.EventFutureDatasourceProperties;

@Configuration

@DependsOn("transactionManager")

@EnableJpaRepositories(basePackages = "com.example.repository.eventFuture", entityManagerFactoryRef = "eventFutureEntityManager", transactionManagerRef = "transactionManager")

@EnableConfigurationProperties(EventFutureDatasourceProperties.class)

public class EventFutureConfig {

@Autowired

private JpaVendorAdapter jpaVendorAdapter;

@Autowired

private EventFutureDatasourceProperties eventFutureDatasourceProperties;

@Bean(name = "eventFutureDataSource", initMethod = "init", destroyMethod = "close")

public DataSource eventFutureDataSource() {

AtomikosDataSourceBean dataSource = new AtomikosDataSourceBean();

dataSource.setUniqueResourceName("xadseventfuture");

dataSource.setXaDataSourceClassName("org.h2.jdbcx.JdbcDataSource");

Properties xaProperties = new Properties();

xaProperties.put("url", eventFutureDatasourceProperties.getUrl());

xaProperties.put("user", eventFutureDatasourceProperties.getUsername());

xaProperties.put("password", eventFutureDatasourceProperties.getPassword());

dataSource.setXaProperties(xaProperties);

dataSource.setPoolSize(10);

return dataSource;

}

@Bean(name = "eventFutureEntityManager")

public LocalContainerEntityManagerFactoryBean eventFutureEntityManager() throws Throwable {

HashMap<String, Object> properties = new HashMap<String, Object>();

properties.put("hibernate.transaction.jta.platform", AtomikosJtaPlatform.class.getName());

properties.put("javax.persistence.transactionType", "JTA");

LocalContainerEntityManagerFactoryBean entityManager = new LocalContainerEntityManagerFactoryBean();

entityManager.setJtaDataSource(eventFutureDataSource());

entityManager.setJpaVendorAdapter(jpaVendorAdapter);

entityManager.setPackagesToScan("com.example.domain.eventFuture");

entityManager.setPersistenceUnitName("eventFuturePersistenceUnit");

entityManager.setJpaPropertyMap(properties);

return entityManager;

}

**}**

1. **Create a Service that saves information into multiple databases in a transaction**

@Transactional(rollbackFor = Exception.**class**)

@Override

**public** **void** insert(EventMessage eventMessage, EventFuture eventFuture) **throws** Exception {

eventMessageRepository.save(eventMessage);

eventFuture.setId(eventMessage.getId());

eventFutureRepository.save(eventFuture);

}

1. **Write a Rest Controller to test the distributed transactions**

@Autowired EventService eventService;

@ResponseBody

@RequestMapping(value = "/event/{msg}", method = RequestMethod.***GET***)

**public** Map<String, Object> event(@PathVariable("msg") String msg) {

Map<String, Object> result = **new** HashMap<String, Object>();

**try** {

EventMessage m = **new** EventMessage();

m.setDescription(msg);

EventFuture f = **new** EventFuture();

f.setAction(m.getDescription().toUpperCase().startsWith("ERROR")?"TO-BE-DETERMINED":"SCHEDULED");

eventService.insert(m, f);

Assert.*notNull*(m.getId());

Assert.*notNull*(f.getId());

result.put("success", "true");

} **catch** (Exception e) {

e.printStackTrace();

result.put("success", "false");

result.put("exception", e.getMessage());

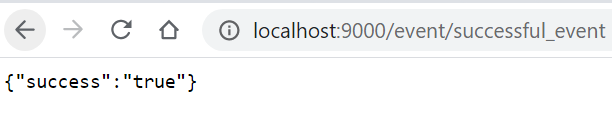
}

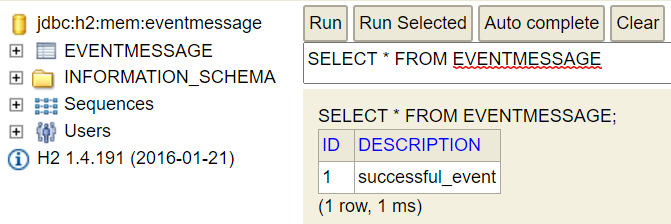
**return** result;

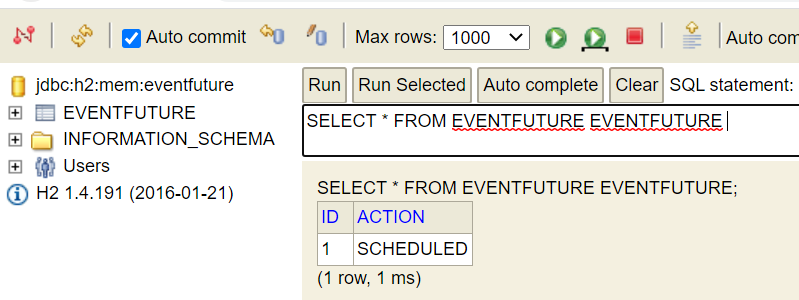
}

}

**Here is a sample output with successfully committed transaction.**







**If we send a message that starts with “ERROR”, we will see a database exception and transaction will rollback.**

