Configuración avanzada de R2DBC con PostgreSQL

1. Introducción

Esta configuración manual de R2DBC está diseñada para aplicaciones reactivas con PostgreSQL que requieren control avanzado sobre el pool de conexiones, la configuración SSL o múltiples instancias. Aunque Spring Boot configura R2DBC automáticamente, este enfoque proporciona mayor granularidad para arquitecturas empresariales.

2. Configuración de la clase principal (R2dbcConfiguration)

Definimos la clase principal con la anotación @Configuration, habilitamos los repositorios reactivos y condicionamos su activación al valor de la propiedad 'spring.r2dbc.manual-config=true'.

```
@Configuration @EnableR2dbcRepositories(basePackages = "com.example.repository") @ConditionalOnProperty(name = "spring.r2dbc.manual-config", havingValue = "true") public class R2dbcConfiguration extends AbstractR2dbcConfiguration {
```

3. Definición de propiedades externas

Las propiedades como host, puerto, usuario y contraseña se inyectan desde el archivo application.properties o application.yml usando @Value.

```
@Value("${spring.r2dbc.host:localhost}")
private String host;



@Value("${spring.r2dbc.port:5432}")
private int port;

@Value("${spring.r2dbc.username}")
private String username;

@Value("${spring.r2dbc.password}")
private String password;

@Value("${spring.r2dbc.database}")
private String database;
```

4. Configuración de ConnectionFactory y ConnectionPool

Creamos una instancia de PostgresqlConnectionFactory y la envolvemos en un ConnectionPool personalizado con validaciones, retry, tiempos máximos y lifecycle hooks.

```
@Bean
@Override
public ConnectionFactory connectionFactory() {


    PostgresqlConnectionConfiguration pgConfig = PostgresqlConnectionConfiguration.builder()
            .host(host)                    .port(port)
            .username(username)    .password(password)
            .database(database)      .schema("public")
            .applicationName("customers-service")
            .connectTimeout(Duration.ofSeconds(10))
            .lockWaitTimeout(Duration.ofSeconds(10))
            .build();
```

```
                    ConnectionPoolConfiguration     poolConfig     =     ConnectionPoolConfiguration.builder(new
PostgresqlConnectionFactory(pgConfig))
            .name("customers-pool")     .initialSize(10)     .maxSize(30)     .maxIdleTime(Duration.ofMinutes(10))
            .maxLifeTime(Duration.ofMinutes(30))                    .validationQuery("SELECT                    1")
            .validationDepth(ConnectionPoolConfiguration.ValidationDepth.LOCAL)                  .acquireRetry(3)
            .background(Duration.ofMinutes(1))   .postAllocate(conn   ->   conn.createStatement("SET   TIME   ZONE
            'UTC'").execute().then(Mono.just(conn)))      .preRelease(conn      ->      conn.createStatement("RESET
            ALL").execute().then(Mono.just(conn))) .build();









    return new ConnectionPool(poolConfig);
}
```

5. TransactionManager reactivo

Este bean permite usar la anotación @Transactional en métodos que devuelven Mono o Flux.

```
@Bean
public ReactiveTransactionManager transactionManager(ConnectionFactory connectionFactory) {
    return new R2dbcTransactionManager(connectionFactory);
}
```

6. Métricas del pool de conexiones (opcional)

Si se activa la propiedad 'spring.r2dbc.pool.monitoring.enabled', se crea un bean para exponer las métricas del pool como tamaño actual, conexiones libres, etc.

```
@Bean
@ConditionalOnProperty(name = "spring.r2dbc.pool.monitoring.enabled", havingValue = "true")
public ConnectionPoolMetrics connectionPoolMetrics(ConnectionFactory connectionFactory) {
    if (connectionFactory instanceof ConnectionPool pool) {
        return new ConnectionPoolMetrics(pool);
    }
    t h r o w   n e w   I l l e g a l S t a t e E x c e p t i o n ( " C o n n e c t i o n F a c t o r
}
```

7. Configuración de segunda base de datos (opcional)

Define un segundo ConnectionFactory si la propiedad 'spring.r2dbc.secondary.enabled' está activada. Útil en sistemas multibase de datos.

```
@Bean("secondaryConnectionFactory")
@ConditionalOnProperty(name = "spring.r2dbc.secondary.enabled", havingValue = "true")
public ConnectionFactory secondaryConnectionFactory(
        @Value("${spring.r2dbc.secondary.url}") String secondaryUrl,
        @Value("${spring.r2dbc.secondary.username}") String secondaryUsername,
        @Value("${spring.r2dbc.secondary.password}") String secondaryPassword) {

    PostgresqlConnectionConfiguration secondaryConfig = PostgresqlConnectionConfiguration.builder()
            .host("secondary-host")
```

```
            .port(5432) .username(secondaryUsername) .pass
            word(secondaryPassword) .database("secondary_d
            b") .applicationName("customers-service-
            secondary") .build();



    return new ConnectionPool(
        ConnectionPoolConfiguration.builder(new PostgresqlConnectionFactory(secondaryConfig))
            .name("secondary-pool")
            .initialSize(5)
            .maxSize(15)
            .maxIdleTime(Duration.ofMinutes(5))
            .validationQuery("SELECT 1")
            .build()
    );
}
```

8. Clase auxiliar ConnectionPoolMetrics

Clase para exponer datos del pool de conexiones, útil para Prometheus o dashboards personalizados.

```
class ConnectionPoolMetrics {
    private final ConnectionPool pool;

    public ConnectionPoolMetrics(ConnectionPool pool) {
        this.pool = pool;
    }

    public ConnectionPool.PoolMetrics getMetrics() {
        return pool.getMetrics().orElse(null);
    }

    public int getAcquiredSize() {
        return getMetrics() != null ? getMetrics().acquiredSize() : 0;
    }

    public int getAllocatedSize() {
        return getMetrics() != null ? getMetrics().allocatedSize() : 0;
    }

    public int getIdleSize() {
        return getMetrics() != null ? getMetrics().idleSize() : 0;
    }

    public int getPendingAcquireSize() {
        return getMetrics() != null ? getMetrics().pendingAcquireSize() : 0;
    }
}
```