

# Pool connections-Wildfly-stress

Pool de conexiones como técnica eficiente en el manejo de recursos y tiempos de respuesta

## Pool conexiones Wildfly - Pruebas stress

“

**Los pools de conexiones** son una estrategia utilizada para aplicaciones web para maximizar el rendimiento de la aplicación optimizando los recursos, cuando nuestra aplicación se conecta a una base de datos lo llamamos como una conexión y consiste en un hilo de comunicación entre el servidor y la base de datos y recae en tener un usuario y una contraseña con la cual manipular dicha BD ya sea en lectura, escritura o cualquier otra operación.

### ¿Estrategias de conexión en una aplicación?

#### Conexión dedicada

“

Puede funcionar Para una aplicación de escritorio que como está aislada es necesario crear una conexión a la base de datos en el momento en que arranca la aplicación y cerrarla al finalizar la misma, entonces cada usuario tiene una conexión para si mismo y como el equipo no es compartido pues la conexión tampoco puede ser compartida.

Esto funciona en algunas aplicaciones de escritorio donde los usuarios no son tantos y se puede controlar fácilmente la apertura y cierre de las conexiones.

#### No puede funcionar

“

En el esquema de aplicaciones web pues es mucho más difícil saber cuándo el usuario abandona el portal web y saber exactamente cuándo cerrar la conexión ( no es solo cerrar la página, sino cuando ya se desvincula del proceso y no es requerida la conexión, es complejo saber cuando ya no la va a necesitar) y como el potencial de usuarios en una aplicación web es mucho mas amplio se vuelve mucho mas costoso el tener tantas conexiones dedicadas y esto solo hará que se almacenen conexiones activas y conexiones abiertas sin usar y probablemente se caería la base de datos dejando sin servicio a la aplicación, además se facilitaría a atacantes denegar nuestro servicio.

Una opción podría ser crear una conexión por cada request y cerrarla al finalizar, pero como esto tiene un costo en tiempo pues haría muy lenta nuestra aplicación y no es lo que se quiere.

## Pool de conexiones

“

Como solución para aplicaciones web se utiliza el **pool de conexiones**, donde básicamente es un conjunto de conexiones variables donde se definen unos mínimos y unos máximos para el manejo de las conexiones totales de nuestra aplicación además de otros parámetros para establecer el comportamiento de estas.

El pool debe garantizar no crear tantas conexiones como usuarios tenga la aplicación y además encargarse de administrarlas, cerrándolas automáticamente, compartiéndolas y demás.

Entonces un pool tiene como parámetros básicos un mínimo de conexiones y un máximo, el mínimo se puede entender como las conexiones que crea el sistema y mantiene activas para abordar las peticiones al servidor, si por ejemplo definimos un mínimo de 10 conexiones, nuestra aplicación crea 10 conexiones al iniciar y las repartirá a las diferentes peticiones que lleguen e interactúen con nuestra BD, al llegar una solicitud de conexión el pool le asigna una conexión a la solicitud y esta conexión queda como ocupada y tan pronto es procesada la solicitud esta conexión no se cierra como tal sino que queda libre para ser usada por alguna otra solicitud y en caso de requerir más y tener un máximo no alcanzado pues se crean las necesarias siempre y cuando no supere el máximo.

- Existen otros parametros como, por ejemplo:

**Flush Strategy:** en caso de error le define al pool como cerrar las conexiones a un estado base.

**Timeouts:** define tiempos de espera antes de activar alguna acción como reducir conexiones en espera.

**Validations:** permite validar la conexión y garantiza que una conexión que el Datasource interpreta como activa realmente esté activa y no haya sido cerrada por Oracle sin darse cuenta.

Estos parámetros se definen sobre el **Datasource** asociado a la aplicación y en nuestro caso manejado por el servidor de aplicaciones wildfly, entonces ¿qué es un datasource?

Básicamente es una conexión configurada desde un servidor, ya sea desde el servidor de backend o desde el servidor de aplicaciones wildfly como es nuestro caso donde el backend llama dicha implementación con la propiedad jndi que tenemos en el properties.

El Datasource nos permite establecer los parámetros como de driver de conexión, el usuario la contraseña, la base de datos, podemos decir que controla la configuración necesaria para establecer la conexión a la BD desde nuestra aplicación y como mencionamos antes permite mediante muchos otros parámetros como establecer tiempos, seguridad, validaciones y por supuesto el concepto de pool de conexiones.

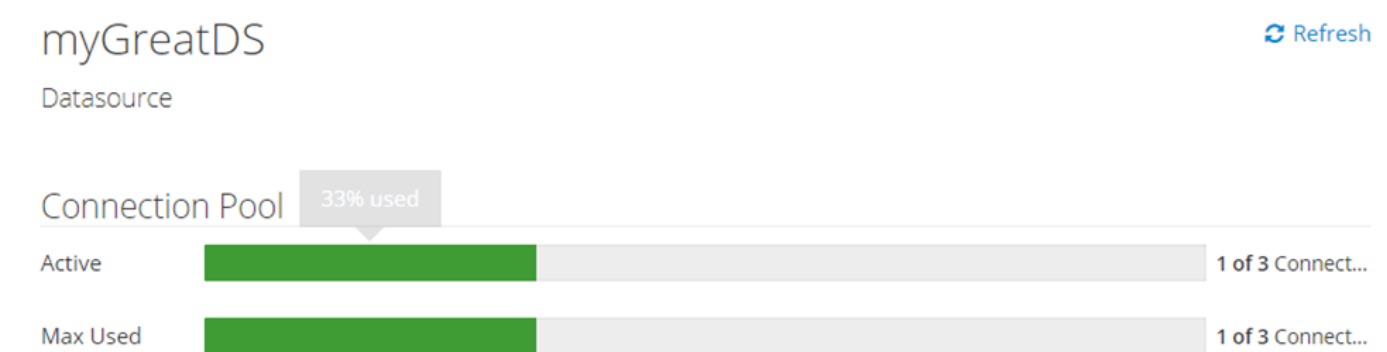
## Transacciones y excepciones

- Cuando la aplicación entra en una transacción la conexión se mantiene ocupada para dicha transacción y es usada por los métodos internos de la misma, y únicamente cuando sucede un error y su respectivo rollback o cuando se hace commit y el método transaccional finaliza es cuando la conexión es liberada.

```

214         } else {
215             return result;
216         }
217     } else {
218         TransactionAspectSupport.TransactionInfo txInfo = this.createTransactionIfNecessary(ptm, txAttr, joinpointIdentification); txInfo:
219
220         Object retVal;
221         try {
222             retVal = invocation.proceedWithInvocation(); invocation: TransactionInterceptor$lambda@24843
223         } catch (Throwable var18) {
224             this.completeTransactionAfterThrowing(txInfo, var18); txInfo: "PROPAGATION_REQUIRED, ISOLATION_DEFAULT"
225             throw var18;
226         } finally {
227             this.cleanupTransactionInfo(txInfo);
228         }
229     }

```



“

Si no es transaccional se crea una transacción que es liberada tan pronto la solicitud es procesada y en caso de errores cuando la excepción es capturada antes de realizar la consulta no se crea ninguna conexión pero si no es detectada y el error es devuelto desde Oracle entonces la conexión se abre encuentra el error lo devuelve y libera la conexión al procesarse la excepción.

## Ejemplo de configuración del DATASOURCE en wildfly

```

1 <datasource jndi-name="java:jboss/datasources/myGreatDS" pool-name="myGreatDS"
2   <connection-url>jdbc:oracle:thin:@10.0.30.10:1521:uisdb01</connection-url>
3   <driver>oracle</driver>
4   <pool>
5     <min-pool-size>1</min-pool-size>
6     <max-pool-size>2</max-pool-size>
7     <prefill>false</prefill>
8     <flush-strategy>AllIdleConnections</flush-strategy>
9   </pool>
10  <security>
11    <user-name>develop</user-name>
12    <password>d3v_3lop$</password>

```

```
13     </security>
14     <validation>
15         <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker" />
16         <check-valid-connection-sql>SELECT 1 FROM DUAL</check-valid-connection-sql>
17         <validate-on-match>true</validate-on-match>
18     </validation>
19     <timeout>
20         <idle-timeout-minutes>1</idle-timeout-minutes>
21     </timeout>
22     <statement>
23         <track-statements>TRUE</track-statements>
24     </statement>
25 </datasource>
```

1. La configuración esta compuesta por una conexión url que especifica como conectarse a la base de datos, la dirección el puerto y el sid o nombre de la instancia de base de datos.
2. El driver que en este caso es el de Oracle y luego tenemos el pool de conexiones.
  - ▶ En este caso tiene un mínimo de 1 y un máximo de 2 conexiones.
3. Tiene las credenciales de conexión en el apartado de seguridad.
4. En el apartado de validación se establece el sql para comprobar la conectividad.
5. En la parte del timeout se establece el tiempo en minutos que debe transcurrir para regresar al pool las conexiones idle o conexiones en espera o libres.
6. En el apartado statement un track-statements para monitorear el cierre de conexiones y obtener información en caso de no ser cerradas por algún error.

## Pruebas realizadas

- ▶ Testeo con entity manager (creación).
- ▶ Test auditorio.
- ▶ Testeo con entity manager close – no afecta nada las conexiones.
- ▶ Testeo con múltiples peticiones (1000 - 4000) en modo lectura – diferentes mínimos y maximos.
- ▶ Testeo con múltiples peticiones en modo escritura (1000 - 4000) - diferentes mínimos y máximos.
- ▶ Testeo en debug multi threats para comprobar los estados simultáneos y el consumo de conexiones.
- ▶ Testeo de propiedades del datasource para manejar las conexiones idle y poder establecer un tiempo de espera para recogerlas.
- ▶ Testeo dejar conexiones abiertas ocupadas para generar errores y ver el comportamiento de cola deseado.

## Configuración recomendada

```
1  <datasource jndi-name="java:jboss/datasources/myGreatDS" pool-name="myGreatDS"
2      <connection-url>jdbc:oracle:thin:@10.0.30.10:1521:uisdb01</connection-u
3      <driver>oracle</driver>
4      <pool>
5          <min-pool-size>8</min-pool-size>
6          <max-pool-size>24</max-pool-size>
7          <prefill>false</prefill>
8          <flush-strategy>AllIdleConnections</flush-strategy>
9      </pool>
10     <security>
11         <user-name>develop</user-name>
12         <password>d3v_3lop$</password>
13     </security>
14     <validation>
15         <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.ext
16         <check-valid-connection-sql>SELECT 1 FROM DUAL</check-valid-connectio
17         <validate-on-match>true</validate-on-match>
18     </validation>
19     <timeout>
20         <idle-timeout-minutes>1</idle-timeout-minutes>
21     </timeout>
22     <statement>
23         <track-statements>TRUE</track-statements>
24     </statement>
25 </datasource>
```

- ▶ Un pool de conexiones de mínimo en 8 esto nos garantiza tener 8 conexiones activas para procesar solicitudes y un máximo de 24 para solicitudes en paralelo en casos de alto tráfico.
- ▶ Un flush strategy de AllIdleConnections que en caso de error limpie todas aquellas conexiones libres.
- ▶ Una validación con select 1 from dual.
- ▶ un timeout con idle-timeout-minutes de 1 minuto, recordando que no es 1 minuto realmente sino 1 minuto más un tiempo de retardo causado por el tiempo de escaneo del idleremover y causa un tiempo final mayor.
- ▶ Un statement con track-statements para cuando las conexiones deben cerrarse y no lo logran poder identificarlo.

## Resultados test aplicados

Con 0-50 conexiones podemos llegar a tener en primer intento esto.

Test run on 8/03/2022 9:09:45 a. m.

**\*\* Project and Scenario Comments, Operator \*\***

**Results of period #1 (from 2 sec to 13 sec ):**

\*\*\*\*\*

Completed Clicks: 500 with 0 Errors (=0,00%)

Average Click Time for 500 Users: 4.403 ms

Successful clicks per Second: 44,72 (equals 161.002, 18 Clicks per Hour)

**Results of complete test**

\*\*\*\*\*

**\*\* Results per URL for complete test \*\***

URL #1 (tes1): Average Click Time 4.403 ms, 500 Clicks, 0 Errors

Total Number of Clicks: 500 (0 Errors)

Average Click Time of all URLs: 4.403 ms

Datasource

Connection Pool

Active  50 of 50 Conn...

Max Used  50 of 50 Conn...

“

Inmediatamente se satura el pool de conexiones y quedan activas y esperando todas las 50 conexiones entonces la segunda prueba genera mejores resultados como se ve a continuación.

Test run on 8/03/2022 9:11:42 a. m.

**\*\* Project and Scenario Comments, Operator \*\***

**Results of period #1 (from 1 sec to 10 sec ):**

\*\*\*\*\*

Completed Clicks: 500 with 0 Errors (=0,00%)

Average Click Time for 500 Users: 416 ms

Successful clicks per Second: 56,32 (equals 202.764,90 Clicks per Hour)

**Results of complete test**

\*\*\*\*\*

**\*\* Results per URL for complete test \*\***

URL #1 (tes1): Average Click Time 416 ms, 500 Clicks, 0 Errors

Total Number of Clicks: 500 (0 Errors)

Average Click Time of all URLs: 416 ms

“

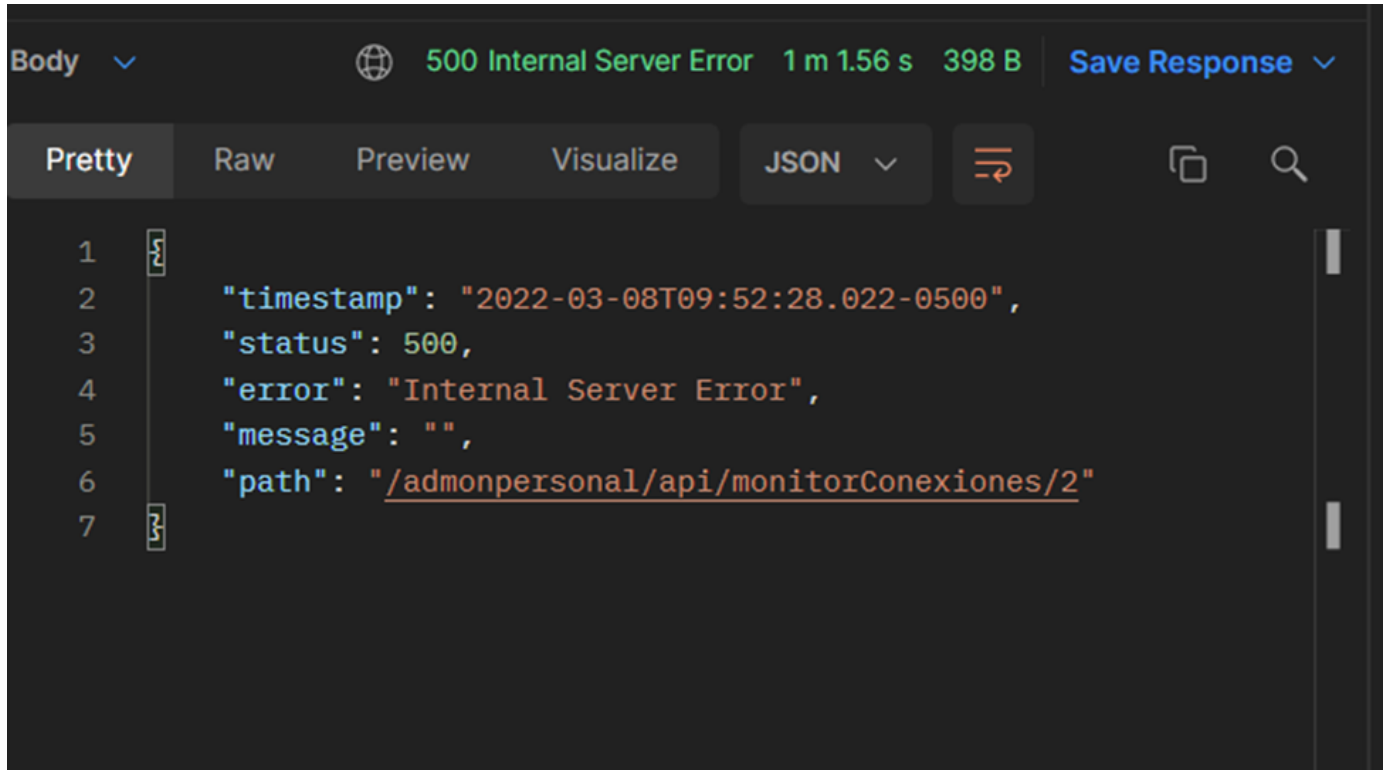
Esto se entiende de modo que cuando el flujo del programa aumenta el servidor inicia a crear conexiones para satisfacer la demanda de solicitudes y en caso de que el flujo siga alto por periodos largos de actividad como por ejemplo de matrícula o algo de esto responde de manera muy eficiente a costo de claro al inicio tener un rendimiento no tan optimo por el costo de crear las conexiones.

## Descripción funcionamiento detallado del pool en WILDFLY

**Cuando se realiza una configuración** para que el servidor de aplicaciones administre el datasource este se encarga de proveer todos los servicios necesarios para ser consumidos, mejora la eficiencia del servidor, la eficiencia de procesos y el control y monitoreo de nuestra aplicación, al definir un datasource y en este caso un pool de conexiones igualmente spring boot se encarga de crear y de cerrar las conexiones dependiendo de las solicitudes, tanto con métodos transaccionales como métodos no transaccionales, en casos de excepciones o en caso de éxito, incluso en caso de retardos por tema de conectividad y el esquema es el siguiente:

1. Registramos un pool de tamaño  $M^{**} \text{ ☹️ } * N$  donde M son las conexiones en espera y N son las conexiones disponibles que pueden llegar a crearse.
2. Tan pronto una conexión es abierta N se reduce en 1 y queda  $M^{**} \text{ ☹️ } * N-1$  pero tan pronto tan pronto se procesa la solicitud la conexión es liberada o regresada al pool y pasa a estar disponible pero sin cerrarse de modo que queda  $M+1^{**} \text{ ☹️ } * N$ .
3. M puede llegar a ser tan grande como el parámetro max-pool-size y tan pequeño como 0.
4. N puede ser tan grande como el parámetro max-pool-size y tan pequeño como 0 cuando no hay mas disponibles porque no se han cerrado correctamente.

5. Cuando una conexión no es cerrada como se mostraba queda activa, pero agota las conexiones disponibles y no regresa al pool, es decir ninguna otra solicitud puede ser tramitada porque no se han liberado las conexiones para ser usadas por otro proceso es por esto que es muy necesario CERRAR SIEMPRE NUESTRAS CONEXIONES, pero claro el backend está preparado para esto y la configuración lo realiza por nosotros así que no debemos preocuparnos por esa parte.
6. Ejemplo de solicitud no resuelta por mal manejo de conexiones:





The screenshot shows a web browser interface with a dark theme. At the top, it displays '500 Internal Server Error' in green text, along with '1 m 1.56 s' and '398 B'. A 'Save Response' button is visible on the right. Below the status bar, there are tabs for 'Pretty', 'Raw', 'Preview', and 'Visualize', with 'Pretty' selected. A 'JSON' dropdown menu is also present. The main content area shows a JSON object with the following fields: 'timestamp', 'status', 'error', 'message', and 'path'. The 'path' field contains the URL '/admonpersonal/api/monitorConexiones/2'.

```
1 {
2   "timestamp": "2022-03-08T09:52:28.022-0500",
3   "status": 500,
4   "error": "Internal Server Error",
5   "message": "",
6   "path": "/admonpersonal/api/monitorConexiones/2"
7 }
```

“

Por último recalcamos que el **backend** está preparado y bien configurado para no sufrir de estos problemas y no tener que preocuparnos puesto que la administración de las conexiones las hace la aplicación en conjunto con el datasource configurado con el pool de conexiones y no debemos preocuparnos por esto al desarrollar.

## Bibliografía

1. <https://docs.wildfly.org/20/wildscribe/subsystem/datasources/data-source/ExampleDS/index.html> 
2. [http://www.cursorhibernate.es/doku.php?id=patrones:pool\\_conexiones](http://www.cursorhibernate.es/doku.php?id=patrones:pool_conexiones) 
3. <https://jorgesanchez.net/manuales/abd/arquitectura-oracle.html#:~:text=Es%20posible%20incluso%20que%20a,se%20almacenan%20en%20el%20servidor> 