Best Practices for WebSphere Liberty Connection Pool Configuration and Tuning

Kumaran Ayyakutti Nathan
J2C Support

James Stephen
J2C Development

Sep 22’2021

Agenda

- Introduction and Part 1 unanswered questions
- Liberty Overview
- Liberty Features
- Liberty Config files
- Liberty Connection pool recommendations
- Demo
- QnA – Please ask the questions in the chat

Last Session Unanswered Questions:

Should I restart the server after creating a new datasource and jdbc configuration ?

- Yes and No. **If the application never been invoked**, in other word if the application never been used and if the shared resources were not loaded then we **really don't need to restart the server**. We expect the config to work for the new connections, but in most cases, we expect you to restart the server to avoid any issues.

Should I restart the server after making change to existing datasource or jdbc configuration ?

- Yes. Unfortunately config files can't be loaded dynamically.

What is Liberty ?

IBM WebSphere Liberty is a Java EE application server with a low-overhead Java runtime environment for cloud-native apps and microservices.

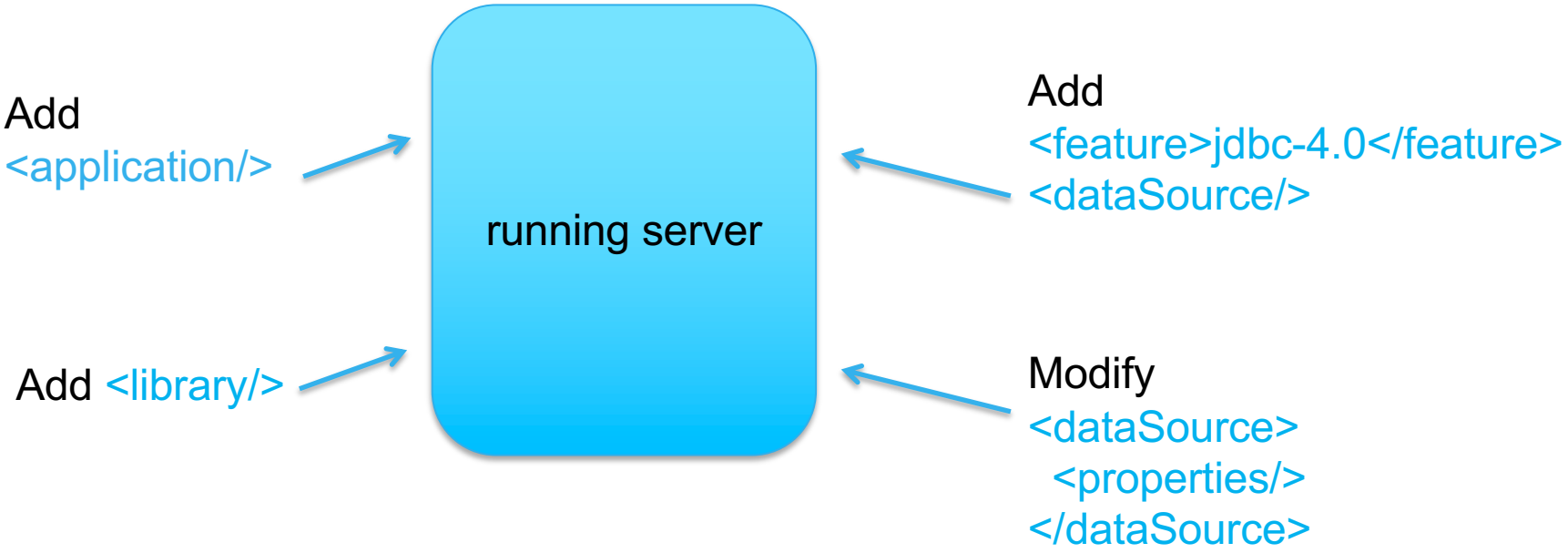
- Lightweight
- Simple
- Composable
- Start Fast
- Flexible
- Dynamic
- Open & Extensible
- Cloud-ready

Key Liberty Features

- **Container and Kubernetes optimized**
  - Liberty is optimized for containerized deployments, continually auto-tuning its performance to the environment.
- **Lightweight, flexible and efficient**
  - The smaller disk footprint of WebSphere Liberty means quicker deploy times, smaller memory footprint and higher throughput.
- **Continuous delivery optimized**
  - A continuous delivery release cycle and zero migration architecture eliminate the need for costly patching and migration cycles.
- **Java EE 8 compatible platform**
  - IBM is committed to Java EE. WebSphere Liberty delivers the latest Java EE technologies and MicroProfile features
- **Spring Boot support**
  - WebSphere Liberty directly supports Spring Boot apps and improves their deployment when used together in containers.
- **DevOps support**
  - WebSphere Liberty works with Gradle, Docker, Kubernetes, Jenkins, IBM UrbanCode® Deploy, and more to automate building, testing and deployment.

Dynamic Configuration

- Dropin application install and update
- Configuration files also monitored for updates
  - **All configuration changes are dynamic**



Configuration Files

The server configuration files are processed in the following order:

- **server.env - ** Optional - Environment variables are specified in this file
- **jvm.options** - Optional - JVM options are set in this file.
- **bootstrap.properties** - Optional - This file influences the startup of the Open Liberty server and can used to configure logging
- **server.xml** - Mandatory - Specifies the server configuration and features.

server.xml file

```
<?xml version="1.0"?>
<server description="new server">
        <!-- Enable features -->
  - <featureManager>
        <feature>jsp-2.3</feature>
        <feature>localConnector-1.0</feature>
        <feature>servlet-4.0</feature>
        <feature>jdbc-4.1</feature>
        <feature>jndi-1.0</feature>
        <feature>monitor-1.0</feature>
    </featureManager>
        <!-- To access this server from a remote client add a host attribute to the following element, e.g. host="*" -->
    <httpEndpoint id="defaultHttpEndpoint" httpsPort="9445" httpPort="9082"/>
        <!-- Automatically expand WAR files and EAR files -->
    <applicationManager autoExpand="true"/>
        <!-- Default SSL configuration enables trust for default certificates from the Java runtime -->
    <ssl id="defaultSSLConfig" trustDefaultCerts="true"/>
    <authData id="test1" user="dbuser1" password="{xor}Oz0vKDtubWwXHw=="/>
    <applicationMonitor updateTrigger="mbean"/>
    <webApplication id="ConnectionExample" name="ConnectionExample" location="ConnectionExample.war"/>
  - <jdbcDriver id="Microsoft_SQL_Server_JDBC_Driver_(XA)">
      - <library>
            <file name="/Users/hunting@us.ibm.com/Documents/eclipse_workspaces/app_dev/ConnectionTesting/src/sqljdbc42.jar"/>
        </library>
    </jdbcDriver>
  - <dataSource id="test" type="javax.sql.XADataSource" jndiName="jdbc/test" jdbcDriverRef="Microsoft_SQL_Server_JDBC_Driver_(XA)">
        <properties.microsoft.sqlserver serverName="localhost" portNumber="1433" databaseName="LIBR0000"/>
        <connectionManager maxPoolSize="20" connectionTimeout="20"/>
    </dataSource>
    <webApplication id="PoolMbeansUtility" name="PoolMbeansUtility" location="PoolMbeansUtility.war"/>
    <logging traceSpecification="*=info:WAS.j2c=all"/>
</server>
```

Liberty Best Practice/Recommendations

Connection Pooling - Connection Manager Properties

Property	Default Value	Comments
connectionTimeout	30s	30S is sufficient in most cases
maxPoolSize	50	50 Per JVM. Adjust based on app requirement
minPoolSize	0	0 is recommended
maxIdleTime	30m	When to remove the connections from free pool – **unusedTimeout** in traditional
reapTime	3m	Thread runs every 180 seconds
agedTimeout	-1	Disabled by default
purgePolicy	EntirePool	EntirePool is recommended, FailingConnectionOnly and ValidateAllConnections are other options

Example code uses the connectionManager element in the server.xml file to define a connection pool for a data source

```
<dataSource id="DefaultDataSource" jndiName="jdbc/example" jdbcDriverRef="DB2" >
        <connectionManager maxPoolSize="10" minPoolSize="2"/>
</dataSource>
```

Connection Pooling – Advanced Connection Manager Properties

- t

Property	Default Value	Comments
enableSharingForDirectLookups	true	his property works only for direct lookup not indirect look up The indirect lookup should be managed by the application resource reference (in web.xml or the ejb-jar.xml)
autoCloseConnections	True	This feature will match the tWas behavior starting Liberty 21.0.0.4
maxConnectionsPerThread	None	Use this for testing and debugging the issue. How many threads in use per connection ?

Connection Pooling – dataSource Properties

- t

Property	Default Value	Comments
queryTimeout	None	▪Application can override the query timeout for a statement at any time by invoking the **java.sql.Statement.setQueryTimeout**
syncQueryTimeoutWithTransactionTimeout	None	Set to **true** to Sync up with your querytimeout

Example code uses the datasource element in the server.xml file to define queryTimeout

<dataSource id="DefaultDataSource" jndiName="jdbc/example" jdbcDriverRef="DB2" **queryTimeout="20S "**
syncQueryTimeoutWithTransactionTimeout =true>
</dataSource>

queryTimeout with SyncQueryTimeoutWithTransactionTimeout

```
statement = connection.createStatement();
statement.executeUpdate(sqlcommand1); // query timeout of 20 seconds is used
statement.executeUpdate(sqlcommand2); // query timeout of 20 seconds is used

transaction.setTransactionTimeout(30);
transaction.begin();
try{
statement.executeUpdate(sqlcommand3); // query timeout of 30 seconds is used
// assume the preceding operation took 5 seconds, remaining time = 30 - 5 seconds
statement.executeUpdate(sqlcommand4); // query timeout of 25 seconds is used
// assume the preceding operation took 10 seconds, , remaining time = 25 - 10 seconds
statement.executeUpdate(sqlcommand5); // query timeout of 15 seconds is used

}
finally
{
transaction.commit();
}
 // query timeout of 20 seconds is used
statement.executeUpdate(sqlcommand6);
```

Demo

Questions and Answers

Summary