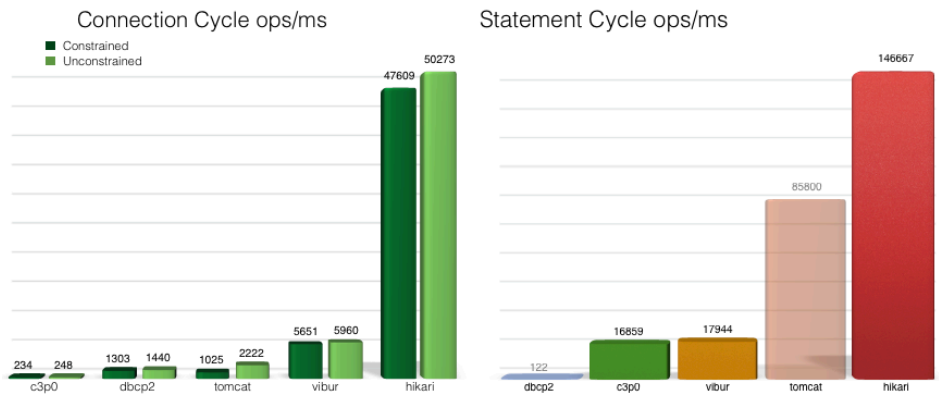
Tomcat+HikariCP

David H Nebinger
2 Years Ago - 31114 Views

In case you aren't aware, Liferay 7 CE and Liferay DXP default to using Hikari CP for the connection pools.

Why? Well here's a pretty good reason:



Hikari just kicks the pants of any other connection pool implementation.

So Liferay is using Hikari CP, and you should too.

I know what you're thinking. It's something along the lines of:

> *But Dave, we're following the best practice of defining our DB connections as Tomcat <Resource /> JNDI definitions so we don't expose our database connection details (URLs, usernames or passwords) to the web applications. So we're stuck with the crappy Tomcat connection pool implementation.*

You might be thinking that, but if you are thankfully you'd be wrong.

Installing the Hikari CP Library for Tomcat

So this is pretty easy, but you have two basic options.

First is to download the .zip or .tar.gz file from http://brettwooldridge.github.io/HikariCP/. This is actually a source release that you'll need to build yourself.

Second option is to download the built jar from a source like Maven Central, http://central.maven.org/maven2/com/zaxxer/HikariCP/2.6.1/HikariCP-2.6.1.jar.

Once you have the jar, copy to the Tomcat lib/ext directory. Note that Hikari CP does have a dependency on SLF4J, so you'll need to put that jar into lib/ext too.

Configuring the Tomcat <Resource /> Definitions

Location of your JNDI datasource <Resource /> definitions depends upon the scope for the connections. You can define them globally by specifying them in Tomcat's conf/server.xml and conf/context.xml, or you can scope them to individual applications by defining them in conf/Catalina/localhost/WebAppContext.xml (where WebAppContext is the web application context for the app, basically the directory name from Tomcat's webapps directory).

For Liferay 7 CE and Liferay DXP, all of your plugins belong to Liferay, so it is usually recommended to put your definitions in conf/Catalina/localhost/ROOT.xml. The only reason to make the connections global is if you have other web applications deployed to the same Tomcat container that will be using the same database connections.

So let's define a JNDI datasource in ROOT.xml for a Postgres database...

Create the file conf/Catalina/localhost/ROOT.xml if it doesn't already exist. If you're using a Liferay bundle, you will already have this file.

Hikari CP supports two different ways to define your actual database connections. The first way is the one that they prefer and it's based upon using a DataSource instance (more standard way of establishing a connection with credentials) or the older way using a DriverManager instance (legacy way that has different ways of passing credentials to the DB driver).

We'll follow their advice and use the DataSource. Use the table from https://github.com/brettwooldridge/HikariCP#popular-datasource-class-names to find your data source class name, we'll need it when we define the <Resource /> element.

Gather up your JDBC url, username and password because we'll need those too.

Okay, so in ROOT.xml inside of the <Context /> tag, we're going to add our Liferay JNDI data source connection resource:

```
<Resource name="jdbc/LiferayPool" auth="Container"
      factory="com.zaxxer.hikari.HikariJNDIFactory"
      type="javax.sql.DataSource"
      minimumIdle="5"
      maximumPoolSize="10"
      connectionTimeout="300000"
      dataSourceClassName="org.postgresql.ds.PGSimpleDataSource"
      dataSource.url="jdbc:postgresql://localhost:5432/lportal"
      dataSource.implicitCachingEnabled="true"
      dataSource.user="user"
      dataSource.password="pwd" />
```

So this is going to define our connection for Liferay and have it use the Hikari CP pool.

Now if you really want to stick with the older driver-based configuration, then you're going to use something like this:

```
<Resource name="jdbc/LiferayPool" auth="Container"
      factory="com.zaxxer.hikari.HikariJNDIFactory"
      type="javax.sql.DataSource"
      minimumIdle="5"
      maximumPoolSize="10"
      connectionTimeout="300000"
      driverClassName="org.postgresql.Driver"
      jdbcUrl="jdbc:postgresql://localhost:5432/lportal"
      dataSource.implicitCachingEnabled="true"
      dataSource.user="user"
      dataSource.password="pwd" />
```

Conclusion

Yep, that's pretty much it. When you restart Tomcat you'll be using your flashy new Hikari CP connection pool.

You'll want to take a look at https://github.com/brettwooldridge/HikariCP#frequently-used to find additional tuning parameters for your connection pool as well as the info for the minimum idle, max pool size and connection timeout details.

And remember, this is going to be your best production configuration. If you're using portal-ext.properties to set up any of your database connection properties, you're not as secure as you can be. Remember, a hacker needs information to infiltrate your system; the more details of your infrastructure you expose, the more info you give a hacker to worm their way in. Using the portal-ext.properties approach, you're exposing your JDBC URL (so hostname and port as well as the DB server type) and the credentials (which will work for DB login but sometimes they might also be system login credentials). This kind of info is worth its weight in gold to a hacker trying to infiltrate you.

So follow the recommended practice of using your JNDI references for the database connections and keep this information out of the hackers hands.

2 Comments

Denis Signoretto
2 Years Ago

Hi David, really useful post. Thanks for sharing!

(You)
2 Years Ago

[...] Jorge Díaz: About HikariCP it is only used in case of configuring Liferay JDBC settings in your portal.properties I have a blog post up showing how you can use Hikari for your tomcat JNDI... [...] Read More