Configuring Grails to use an Embedded H2 Database

Mark Woodford, 1/18/2015

When developing a Grails application, there are a number of database options available. You may choose to use H2 (default), HSQLDB, MySQL, etc. In addition, you can choose how the database will be stored:

1. **hosted on a server:** a service such as mysqld, running separately from the web application and possibly on a different machine, handles requests from the application to create, read, update, or delete data within the database
2. **embedded:** the database is located within the application itself, and all operations to the database are done directly by the application

For rapid prototyping, and for use cases in which accesses to the database are relatively infrequent and there are only a few connections at once, an embedded database will work just fine. For small projects, embedded databases have the advantage of being easy to initialize and maintain.

Another design choice is whether to have the database stored on disk or in memory. If the database is expected to grow very large, then it would be better to store the database files on disk to prevent out-of-memory errors. In addition, while keeping the database in memory allows faster access times, if something goes wrong, all data in memory will be lost.

I now explain how to configure your Grails application to connect to an embedded H2 database stored on disk.

1. Open the DataSource.groovy configuration file.
2. Direct your attention to the dataSource settings at the beginning of the file. The property driverClassName should be “org.h2.Driver”. Also, note that the username and password for the database are defined here.
3. Now locate the environment-specific settings. Grails allows you to configure specific settings for development, testing, and production. I suggest leaving the development and testing settings how they are. In the production settings, just above the url, insert the following line:

```
dbdir = "${System.properties['catalina.base']}/project-name/database-name"
```

This defines a filepath for the destination of the on-disk database. The system variable ‘catalina.base’ will translate to the home directory on the tomcat server. Replace project-name with the name of your project, to distinguish it from other projects on the server, and replace database-name with a name for your database. Lastly, modify the connection url to be:

```
url = "jdbc:h2:file:${dbdir};MVCC=TRUE;LOCK_TIMEOUT=10000"
```

4. Save the changes to DataSource.groovy. Now your Grails application is configured to connect to an on-disk H2 database at the filepath specified on the tomcat server. Now when you deploy your application (as a WAR file) to the tomcat server, a new H2 database with the given name will be created (if it doesn't already exist).

Now you can make the WAR file and deploy your web app to the Tomcat server. WAR stands for Web ARrchive and is very similar to a JAR file. Deployment is primarily uploading the WAR file to the server's appBase. The appBase is the directory that Tomcat looks for new WAR files and expands them, called exploding. Our Tomcat server is at

```
cshci-dev.mtu.edu
```

and the appBase is at

```
/var/lib/tomcat/webapps
```

Below are detailed instructions for making the WAR files and deploying your web app:

1. In the BuildConfig.groovy, uncomment the war.file specification. It is near the top of the file, and edit it so that it looks like

   ```
   grails.project.war.file = "target/${appName}.war"
   ```

   Note the we do not want the version number. The WAR file name must match the project name or the web app will not deploy properly. You can edit the WAR file name if you want without hurting the contents.
2. On your development machine, your home machine, make the war file in production environment mode. Production environment is the default for the war command but not for the run-war command. The command executed in your project directory is simply

   ```
   grails war
   ```

   The build and compression will take several minutes. You will find the WAR file in your project's target/ directory.
3. Using a SFTP client (WinSCP is a very good choice for Windows) log in into the Tomcat machine. (See domain name above.) I highly recommend that you use RSA keys so that the client can auto log in during network distributions. If you are logging into the server from a machine then you need to use a virtual network (VPN) with mtu. Point a browser to

   ```
   vpn.mtu.edu
   ```

Log in and chose get mtu. You’ll probably have to install a plugin into the browser the first time. The plugin does not work for all browsers. I have been successful installing the plugin to FireFox. Choose “get an MTU address” and a second window window will pop up. Leave this window open or you’ll lose the connection. You can now use the SFTP client. If you do not have a VPN connection, you’ll get an “network error” and not be able to connect.

4. In the SFTP client, navigate to the appBase. (See above.)
5. If you have previously deployed a WAR file for the web app that you are currently deploying then it should still exist in the appBase alongside the exploded web app. The exploded wep app is a directory with the same name as the WAR file. Delete the old WAR in the appBase before uploading the new WAR.
6. Wait. Wait until Tomcat deletes the old exploded web app. If you do not wait then deployment will fail, and Tomcat will never explode the new WAR file.
7. When the old web app directory disappears, use your SFTP client to upload the new WAR file.
8. Wait. Sometimes Tomcat takes several minute to see the new WAR file and explode it.
9. Point your browser to the web app and inspect it.