Magnolia CMS with MySQL

Your Rating: ☆☆☆☆☆ Results: ★★★☆☆ 121 rates

Installing MySQL Server

Linux

To install MySQL server on Linux:

- On Debian-like distributions using the apt packager:

```
sudo apt-get install mysql-server
```

- On distributions using the yum packager:

```
sudo yum install mysql
sudo yum install mysql-server
```

Windows

To install MySQL server on Windows:

1. Use the MySQL Server installer.
2. After installation, modify the `path` system variable. Add a semicolon to the end, followed by the path to the `bin` folder of your MySQL Server installation. In the default MySQL 5.1 installation, the path you need to add to the `path`variable looks like this.

   ```
   ;C:\Program Files\MySQL\MySQL Server 5.1\bin
   ```

OSX

To install MySQL server on OSX:

1. Use the package installer provided on the MySQL disk image.
2. Install the MySQLStartupItem.pck package which provides items for starting and stopping the MySQL server.
3. Or, if you use Homebrew:

   ```
   brew install mysql
   ```

Starting and stopping MySQL Server

MySQL starts and stops automatically. If there is a need to do this manually, perform the task in the terminal.

Debian-like Linux distributions

To start MySQL server:

```
sudo service mysql start
```

To stop MySQL server:

```
sudo service mysql stop
```

To restart MySQL server

```
sudo service mysql restart
```

In case `service` doesn't work, you can also try execute these commands directly:

```
sudo /etc/init.d/mysql start
sudo /etc/init.d/mysql stop
sudo /etc/init.d/mysql restart
```

Fedora and similar distributions

```
sudo service mysqld start
sudo service mysqld stop
sudo service mysqld restart
```

or

```
sudo /etc/init.d/mysqld start
sudo /etc/init.d/mysqld stop
sudo /etc/init.d/mysqld restart
```

Windows

To start MySQL server:

```
mysqld
```

To stop MySQL server:

```
mysqladmin -u root shutdown
```

OSX

To start mysql server:

```
sudo /Library/StartupItems/MySQLCOM/MySQLCOM start
# for Homebrew: mysql.server start
```

To stop mysql server:

```
sudo /Library/StartupItems/MySQLCOM/MySQLCOM stop
# for Homebrew: mysql.server stop
```

Configuring MySQL

Changing the root password

To change the root password, issue the following command:

```
mysqladmin -u root password <newPassword>
```

Setting the storage engine and packet size

Setting the storage engine and packet size is done in a configuration file. This file is different depending on the operating system:

- `<mysqlInstallDir>/my.ini` on Windows
- `etc/mysql/my.cnf` on linux and OSX (for MySQL 4.1 and older use `etc/my.cnf`)
- if you are using Homebrew: /usr/local/Cellar/mysql/$MYSQL_VERSION/my.cnf ← in older versions. if you are using a newer version, try this to locate the my.cnf file:

  ```
  ls $(brew --prefix mysql)/support-files/my-*
  /usr/local/opt/mysql/support-files/my-default.cnf
  ```

If the file doesn't exist, create a new file based on the example in `/usr/share/doc/mysql-server-5.1/examples` for Linux or OSX. Add or edit the following lines in the `[mysqld]` section:

Set the storage engine to InnoDB, unless you're at MySQL 5.5+ where InnoDB is used by default. The default MyISAM engine is not supported with Magnolia CMS due to the lack of transaction support.

```
default_storage_engine = InnoDB
```

To prevent the "size exceeds max_allowed_packet" error for large BLOBs, increase the maximum packet size. 32M will be enough for Jackrabbit. This value indicates the maximum permitted value of the packet, not the initial value.

```
max_allowed_packet = 32M
```

After the changes, restart the MySQL server to apply them.

Creating and deleting databases (schema)

Log in to MySQL command line tools by typing in the terminal:

```
mysql -u <userName> -p
```

Create a schema.

```
create schema <schemaName>;
```

To delete a schema:

```
drop schema <schemaName>;
```

Each Magnolia CMS instance needs a a unique schema. Running two or more public or author instances on the same schema will cause unpredictable behavior. In a production environment it is also better to physically separate the databases (not only the schema) to increase security and scalability.

Configuring Magnolia and Jackrabbit

The instructions below are split into two parts: what to do in a freshly installed Magnolia CMS instance and what to do in an already-started instance. Choose one depending on your situation. The examples show how to configure an author instance. The configuration procedure is the same for a public instance, only with different names.

Freshly installed Magnolia CMS instance

Bundle persistence manager

The first option to configure MySQL is using a bundle persistence manager. Follow Setting up a Jackrabbit persistence manager.

The problem with using the bundle persistence manager is that Jackrabbit creates a connection with the database for the entire time it runs and keeps this connection alive even when it is not needed. The issue is that MySQL closes idle connections after a certain timeout. Jackrabbit doesn't know this and thinks that connection is still open. When a request arrives, Jackrabbit spends lot of time trying to use the already-closed connection and then tries to recover from this situation, which causes an error in the log.

To prevent MySQL from closing connections to soon, increase the timeouts. Edit the `my.cnf` or `my.ini` files as described above. There are two timeouts for closing idle connections: `wait_timeout` for server and `interactive_timeout` for client. Increase them to a higher value:

```
wait_timeout                = 86400
interactive_timeout        = 86400
```

These values are in seconds and should reflect the expected idle time (86400 seconds = 24 hours).

Another disadvantage of the bundled persistence manager is the storing of database connections in several files. The first is the repository configuration file. Even this file has database connection in more than one place depending on which components are stored in the database. After start, during the initialisation of workspaces, a file is created for each workspace. These files are created by copying parts from the repository configuration file so they also contain database settings. So changing anything in the database settings is complicated.

Bundle persistence manager with JNDI

Another alternative is to set up Jackrabbit with a JNDI data source. This option uses also the bundle persistence manager so you have to increase `wait_timeout` and `interactive_timeout` as described above. The main advantage of this configuration is the use of data sources where you can define the database connection settings. A data source can be accessed easily by specifying its name. This allows you to keep all database connection settings in a single file so making changes is easier.

You can also decide if you want to use a pooled or an unpooled data source. There is no advantage in using a connection pool because you cannot reuse connections. The bundle persistence manager keeps the connection for the entire time it is running. This is why it is better to use an unpooled data source. For the bundle persistence manager there isn't much difference between the two settings.

To configure a bundle persistence manager with JNDI:

1. Create an empty schema for a Magnolia CMS instance, for example `magnoliaAuthor`.
2. Copy the MySQL driver .jar file into the `<CATALINA HOME>/lib` or `magnoliaAuthor/WEB-INF/lib }}folder and remove from this folder the .jar file that contains the Derby database driver {{derby-x.jar`.
3. In `magnoliaAuthor/WEB-INF/config/default/magnolia.properties`, set the repositories configuration property `magnolia.repositories.jackrabbit.config`. The value should be the location of your configuration file, for example `WEB-INF/config/repo-conf/jackrabbit-bundle-mysql-search.xml`.
4. Edit `<CATALINA HOME>/conf/context.xml` where the data source is defined. Depending on the type of data source you want to use, add an appropriate data source with your database connection settings.
    - For pooled datasource prior to magnolia 4.5:

      ```
      <Resource name="jdbc/MagnoliaAuthor" auth="Container"
      type="com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource"
      factory="com.mysql.jdbc.jdbc2.optional.MysqlDataSourceFactory"
      maxActive="50" validationQuery="SELECT 1" user="root" password="root" explicitUrl="true"
      driverClassName="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost:3306/magnoliaAuthor"/>
      ```

      Parameters:

        - `maxActive` connections needs to be set higher. The default value is 8 but Jackrabbit needs one active connection for each workspace. Without increasing this parameter's value Magnolia CMS will get stuck during initialization of workspaces.
        - `validationQuery` checks if the connection is alive when it is requested from the pool.
        - `explicitUrl` enables adding parameters via a URL.
          Avoid settings which can remove an idle connection from application. An example is the `removeAbandoned` parameter which removes the connection from the application when idle time is larger than `removeAbandonedTimeout`. In the default configuration this parameter is disabled.
    - For unpooled datasource prior to magnolia 4.5:

      ```
      <Resource name="jdbc/MagnoliaAuthor" auth="Container"
      type="com.mysql.jdbc.jdbc2.optional.MysqlDataSource"
      factory="com.mysql.jdbc.jdbc2.optional.MysqlDataSourceFactory"
      user="root" password="root" explicitUrl="true" driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost:3306/magnoliaAuthor"/>
      ```

    - Pooled datasource for magnolia 4.5:
      In magnolia 4.5 was updated jackrabbit to 2.4. With this change previously mentioned MySQL datasources are slow (3-4 times longer startup). That is why generic datasource is recommended instead of MySQL.

      ```
      <Resource name="jdbc/MagnoliaAuthor" auth="Container"
      type="javax.sql.DataSource"
      username="root" password="root" driverClassName="com.mysql.jdbc.Driver"
      validationQuery="SELECT 1" testOnBorrow="true"
      url="jdbc:mysql://localhost:3306/magnoliaAuthor"/>
      ```

      Parameters:

        - `username` is used in this datasource instead of `user` as it is in MySQL datasource.
        - `testOnBorrow` checks if the connection is alive when it is requested from the pool, running `validationQuery` to do so.

5. In `magnoliaAuthor/WEB-INF/config/repo-conf/jackrabbit-bundle-mysql-search.xml`file, point both persistence managers to the data source defined in your database connection.
    - The first is for workspaces in the `Repository/Workspaces/Workspace/`node:

    ```
    <PersistenceManager class="org.apache.jackrabbit.core.persistence.bundle.MySqlPersistenceManager">
      <param name="driver" value="javax.naming.InitialContext"/>
      <param name="url" value="java:comp/env/jdbc/MagnoliaAuthor"/>
      <param name="schemaObjectPrefix" value="${wsp.name}_" />
      <param name="externalBLOBs" value="false" />
    </PersistenceManager>
    ```

    - The second is for versioning in the `Repository/Versioning/`node:

    ```
    <PersistenceManager class="org.apache.jackrabbit.core.persistence.bundle.MySqlPersistenceManager">
      <param name="driver" value="javax.naming.InitialContext"/>
      <param name="url" value="java:comp/env/jdbc/MagnoliaAuthor"/>
      <param name="schemaObjectPrefix" value="version_" />
      <param name="externalBLOBs" value="false" />
    </PersistenceManager>
    ```

6. Start Magnolia CMS and check the logs.

Pooled persistence manager

To avoid the problem of MySQL closing connections, use connection pooling with a pooled bundle persistence manager. In the persistence manager you can define a validation query which checks if the connection is still alive. The validation query should return at least one result. An example of such a query for MySQL is `select 1`.

The pooled persistence manager is available starting with Jackrabbit 2.0. We bundle Jackrabbit 2.4 with Magnolia CMS 4.5. You can use a pooled JNDI datasource for magnolia 4.5 as mentioned above or define a connection in datasource directly in the repository configuration file.

To configure a pooled persistence manager:

1. Create an empty schema for a Magnolia CMS instance, for example `magnoliaAuthor`.
2. Copy the MySQL driver .jar file into the `magnoliaAuthor/WEB-INF/lib` folder and remove from this folder the .jar file that contains the Derby database driver `derby-x.jar`.
3. In `magnoliaAuthor/WEB-INF/config/default/magnolia.properties`, set the repositories configuration property `magnolia.repositories.jackrabbit.config`. The value should be the location of your configuration file, for example `WEB-INF/config/repo-conf/jackrabbit-bundle-mysql-search.xml`.
4. Edit the configuration file `magnoliaAuthor/WEB-INF/config/repo-conf/jackrabbit-bundle-mysql-search.xml`depending where you want to define the data source:
    - If you want define the data source here, add a new (or modify if already exist) `DataSources` node under the `Repository`root node and define your database connection in it:

    ```
    <Repository>
       <DataSources>
          <DataSource name="magnoliaAuthor">
             <param name="driver" value="com.mysql.jdbc.Driver" />
             <param name="url" value="jdbc:mysql://localhost:3306/magnoliaAuthor" />
             <param name="user" value="root" />
             <param name="password" value="root" />
             <param name="databaseType" value="mysql"/>
             <param name="validationQuery" value="select 1"/>
          </DataSource>
       </DataSources>
    ```

    - If you want to use the JNDI datasource, then create `Datasources`which will be pointing to your JNDI:

    ```
    <Repository>
       <DataSources>
          <DataSource name="magnoliaAuthor">
             <param name="driver" value="javax.naming.InitialContext"/>
             <param name="url" value="java:comp/env/jdbc/MagnoliaAuthor"/>
             <param name="databaseType" value="mysql"/>
    ```

```
        </DataSource>
    </DataSources>
```

   and edit `<CATALINA HOME>/conf/content.xml`. Add a pooled data source as described in the section about JNDI above.

5. Point both persistence managers to the data source defined in your database connection and change the handling class to `org.apache.jackrabbit.core.persistence.pool.MySqlPersistenceManager`.
   - First is for workspaces in the `Repository/Workspaces/Workspace/`node:

     ```
     <PersistenceManager class="org.apache.jackrabbit.core.persistence.pool.MySqlPersistenceManager">
        <param name="dataSourceName" value="magnoliaAuthor"/>
        <param name="schemaObjectPrefix" value="${wsp.name}_" />
     </PersistenceManager>
     ```

   - Second is for versioning in the `Repository/Versioning/`node:

     ```
     <PersistenceManager class="org.apache.jackrabbit.core.persistence.pool.MySqlPersistenceManager">
        <param name="dataSourceName" value="magnoliaAuthor"/>
        <param name="schemaObjectPrefix" value="version_" />
     </PersistenceManager>
     ```

6. Start Magnolia CMS and check the logs.

DataStore and FileSystem in the database

Jackrabbit workspaces are composed of several components. Most of them can be placed in the database. In addition to the persistence manager, you can move the DataStore and FileSystem workspaces into the database. The FileSystem workspace contains some additional information about content. DataStore is a place where large files are stored. It is usually faster to store them directly in the file system.

Moving the DataStore workspace to the database means you need to adjust the maximum file size limit in `max_allowed_packet`. It defines the maximum size you can store in the database. Increase the value to accommodate your file size requirements.

If you want DataStore or FileSystem in the database, edit `magnoliaAuthor/WEB-INF/config/repo-conf/jackrabbit-bundle-mysql-search.xml`. Change the classes of the component that should be placed in the database. For each section that will be moved to the database, add a `schemaObjectPrefix` parameter which is a prefix for the tables. Specify the database connection to correspond to your settings which will be same as for the persistence manager.

For example, with data source defined in the repository config with magnolia 4.5:

```
<FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
   <param name="dataSourceName" value="magnoliaAuthor"/>
   <param name="schemaObjectPrefix" value="fs_" />
</FileSystem>

<DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
   <param name="dataSourceName" value="magnoliaAuthor"/>
   <param name="schemaObjectPrefix" value="ds_" />
</DataStore>
```

FileSystem and DataStore are listed three times in the repository configuration file. The first occurrence is global for all repositories. Further, both components are placed in the workspace in the repository and in the versioning node. When moving these components to the database, specify for a different prefix for each component to avoid conflicts.

Already started instance of Magnolia CMS

If you already have a bundle or pooled persistence manager and you don't change location of components, then the change is easy. Edit the configuration files the same way as you would for a freshly installed instance. In addition, modify all workspaces located in `magnoliaAuthor/repositories/workspaces/*/workspace.xml`. Change the persistence manager, the FileSystem and the DataStore to match your settings.

If you want change database type, move some component to different location (for example move DataStore from file system to database) or switch from simple pm then, you have to migrate your data.

Changing the storage engine for already existing tables

Magnolia supports only the InnoDB storage engine for MySQL Server. If you have already created tables with a different engine you must change them to InnoDB. Converting tables between engines is a complicated operation which takes lot of time. Also, InnoDB tables need more space than MyISAM tables but on the other hand InnoDB supports transactions and foreign keys.

⚠ Executing the following commands takes a long time. Check also that you have enough free space before running them.

To change the storage engine to InnoDB, execute the following command on every table in your schema.

```
ALTER TABLE '<schemaName>'.'<tableName>' TYPE = InnoDB;
```

To change the engine for all tables in the schema you can use these two commands in the terminal:

```
mysql -u <userName> -p --execute="USE information_schema; SELECT CONCAT(\"ALTER TABLE \`\", TABLE_SCHEMA,\"\`.
\`\", TABLE_NAME, \"\` TYPE = InnoDB;\") as ''  from TABLES where TABLE_SCHEMA = \""<schemaName>"\";" > convert.
sql;
mysql -u <userName> -p < convert.sql;
```

Replace `<userName>` and `<schemaName>` in the command with actual values. The first command creates a script which converts all tables in the specified schema. The second command runs the script.

Backup

To create a backup, do both steps at the same time so you can be sure they are in sync.

1. Create a database dump.
    - In standard edition MySQL Server you can use `mysqldump`to create the dump in your backup folder.

    ```
    mysqldump -u <userName> -p <schemaName> > dump.sql
    ```

    - In enterprise edition of MySQL Server use the backup module which provides much more functionality such as incremental backup and packing.
2. Copy the repositories folder to your backup location.

Scheduling a backup

Scheduling can be done with a cron job for linux or OSX.

Restoring a backup

In case only the database or repositories are corrupt, restore from a backup:

1. Shut down the Tomcat server.
2. Replace database content with content from a dump file.

```
mysql -u <userName> -p <schemaName> < dump.sql
```

3. Delete the content of the repositories folder and replace it with content from the backed up folders.
4. Start the Tomcat server.

If there are more corrupted things, it is better to restore the data to a clean Magnolia CMS instance. Take the Magnolia CMS WAR file or webapp from the bundle, add all your custom modules, define all your configuration, start the server once and perform the installation and Web update. After that, continue with the restore procedure above.