pentaho®
A Hitachi Group Company

Best Practices - Securing Connection Passwords for the Pentaho BA Suite

Contents

Overview

This document provides an approach to use encrypted or obfuscated passwords with the BA and DI servers deployed within a supported Tomcat server. This guide uses features and APIs within Tomcat, assumes familiarity with the archive installation methods for the Pentaho servers, and general knowledge of Oracle's Java platform.

Pentaho provides and documents different ways to install the Pentaho Business Analytic (BA) server and Data Integration (DI) server to accommodate different requirements. Many options are explained in the Select Installation Method article available in the online documentation. The options allow solutions to use a new Apache Tomcat instance for the installation to deploy the Pentaho servers into existing configured application servers.

Software	Version
Pentaho	5.4, 6.x, 7.x

Pentaho and Data Source Connections

Data source connections are central concepts for a Pentaho-based solution. Pentaho uses a database to manage its solution repository, minimally. The solution repository is the collection of all reports, transformations, and dashboards that are visible within the Pentaho User Console (PUC). The solution repository is composed of several components, such as content managed by Apache Jackrabbit, auxiliary information stored in a database named Hibernate, and schedules managed by Quartz. All data sources used within Pentaho will directly or indirectly use a data source connection.

Pentaho supports three methods for defining data source connections to databases:

- Connections can be defined within the application server and exposed with the Java Naming and Directory Interface (JNDI)
- Defined within PUC as Pentaho managed connections
- Specified directly within the data source, report, transformation, or job

We recommend using connections defined within the application server, and those available as JNDI for production environments.

JNDI connections are more portable, providing better performance due to more scalable connection pooling provided by the application servers, and benefit from the security features of the chosen application server. Visit the Specify Data Connections articles in Pentaho documentation for more details.

This recommendation for JNDI-defined connections holds true for the database hosting the solution repository. Since Pentaho-managed connections are also stored within the solution repository, using those connections is not an option for these solution repository related connections.

Tomcat Data Source Connections

Tomcat's JNDI connections are defined in XML configuration files within either Tomcat's global configuration or the deployed web application. The features available for these connections are determined by Tomcat, and can be found in Tomcat's JNDI Resources HOW-TO and Pentaho's Define JNDI Connections documentation.

Defining new connections within Tomcat requires a restart of the application server, because of how Tomcat handles connections. Use Pentaho managed connections or a different application server if the solution will involve frequently adding new data source connections, and frequent restarts of the server is not suitable. However, the connections for the solution repository should be defined within Tomcat and made available with JNDI.

Tomcat's use of XML for connection definition potentially exposes the password for these connections as plain text. While these files should be on a server where access to these configuration files is secured and monitored, passwords in plain text may be a challenge for some environments. For tips on managing the configuration files, see Tomcat's guidance documentation. Tomcat uses Java's javax.naming.spi.ObjectFactory API to allow administrators to change how passwords are specified for data source connections. Example implementations are available at JDev, Experts Exchange, and WebSphere Journal; however, others can also be found using any internet search engine. Evaluate existing implementations or implement a custom `ObjectFactory` to employ a desired technique for encrypting, masking, or obfuscating the passwords for data source connections within Tomcat.

Pentaho supports the use of JNDI, but does not specifically support any custom `ObjectFactory` used within Tomcat. Therefore, issues with the custom factory are beyond Pentaho Support.

Using JNDI for the Solution Repository

The database connection for the solution repository is defined within a few configuration files found in the `pentaho-solutions/system` directory of the installed Pentaho server. These files are text files; however, if a JNDI connection is used as described, the database connection would only be referenced by name and would not include any information about the actual connection. Therefore, if the connection is defined within Tomcat, no Pentaho configuration files would include the credentials for the connection. No passwords related to connections will be in clear text if the connections are securely defined within Tomcat.

The following steps document the process for installing the Pentaho servers with a new solution repository using JNDI connections:

- Create the Solution Repository
- Configure the Solution Repository
- Specify the Connections
- Update the Solution Repository to Use JNDI
- Start and Restart the Server

They are based upon the install with Own Repository installation method for the servers, also known as the archive-based install method. The steps are available in Pentaho's guide for Installing Pentaho Servers with your own appropriate repositories.

Step 1: Create the Solution Repository

Follow the instructions in the Initialize Repository guide to create your solution repository.

For Pentaho 6.x, the BA and DI servers cannot share the same instance of the solution repository; however, their individual repositories can coexist within the same relational database server.

For Pentaho 7.x, the BA and DI solutions can coexist in the same repository; however, it is a best practice to keep the two environments separated.

Be sure to follow the instructions for the targeted relational database technology.

Step 2: Configure the Solution Repository

Pentaho provides documentation on configuring the server to point to the solution repository databases created. For ease of evaluation, these instructions use plain text passwords in the configuration files. Follow the Configure Repository guide and use a generic password. The future steps will update these files to use JNDI connections.

Step 3: Specify the Connections

Follow the documented steps for your repository database to specify connections for Pentaho. This step defines the connections within Tomcat so that they can be referenced by Pentaho. If using a custom `ObjectFactory`, update the definitions in the `context.xml` appropriately and copy the necessary files for the custom `ObjectFactory` to the `tomcat/lib` directory.

Below is an excerpt from a `context.xml` that uses a custom `ObjectFactory` named `com.mycompany.CustomFactory` for the solution repository hosted in `PostgreSQL`. This custom factory retrieves the password from the environment and not from the configuration file. The factory would be used for each of the connections defined within the Specify Connections documentation.

For Pentaho 7.x, the parameter for `maxWait` is called `maxWaitMillis`. The parameter for `maxActive` is called `maxTotal`.

```
<Resource
 validationQuery="select 1"
 url="jdbc:postgresql://localhost:5432/hibernate"
driverClassName="org.postgresql.Driver"
 username="hibuser"
 maxWait="10000"
 maxIdle="5"
 maxActive="20"
 factory="com.mycompany.CustomFactory"
 type="javax.sql.DataSource" auth="Container"
name="jdbc/Hibernate"/>
```

Additionally, define connections for the `jackrabbit` database created in the Step 2: Configure the Solution Repository section above. Below is a connection for the `jackrabbit` database using the same custom factory. Note the name of the connection in this example is `jackrabbit`.

```
<Resource
 validationQuery="select 1"
 url="jdbc:postgresql://localhost:5432/jackrabbit"
 driverClassName="org.postgresql.Driver"
 username="jcr_user"
 maxWait="10000"
 maxIdle="5"
 maxActive="20"
 factory="com.mycompany.CustomFactory"
 type="javax.sql.DataSource" auth="Container"
name="jdbc/jackrabbit"/>
```

Step 4: Update the Solution Repository to Use JNDI

When you configured the repository, the configuration files were updated to point to the solution repository databases. This step updates them to use JNDI instead of the embedded credentials.

The two files that need to be modified are the Hibernate configuration file for the targeted database and the `pentaho-solutions/system/jackrabbit/repository.xml`. The specific Hibernate configuration file to modify for your environment is the `pentaho-solutions/system/hibernate/hibernate-settings.xml` file.

We recommend backing up the specific Hibernate configuration file and the `repository.xml` before making modifications.

For the specific Hibernate file, the modification is to remove the credentials, URL, and driver specified and replace it with the JNDI name. The example below is a Hibernate configuration file for Postgres, `postgresql.hibernate.cfg.xml`, updated to use JNDI.

```
<hibernate-configuration>
 <session-factory>
  <property

name="cache.provider_class">net.sf.ehcache.hibernate.SingletonEhCach
eProvider</property>

 <property name="hibernate.generate_statistics">true</property>
 <property name="hibernate.cache.use_query_cache">true</property>

 <!-- Postgres 8 Configuration -->
 <property
name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
 <property

name="hibernate.connection.datasource">java:comp/env/jdbc/Hibernate<
/property>
 <property name="connection.pool_size">10</property>
 <property name="show_sql">false</property>
 <property
name="hibernate.jdbc.use_streams_for_binary">true</property>
 <!-- replaces DefinitionVersionManager -->
 <property name="hibernate.hbm2ddl.auto">update</property>
 <!-- load resource from classpath -->
 <mapping resource="hibernate/postgresql.hbm.xml" />
 <!-- mapping resource above is from CE; below is from EE -->
 <mapping resource="hibernate/postgresql.EE.hbm.xml" />
 </session-factory>
</hibernate-configuration>
```

The remaining configuration file to update defines the connections for the Jackrabbit portion of the repository: `pentaho-solutions/system/jackrabbit/repository.xml`. Replace the driver and URL parameter for each connection, and remove the user name and password as documented in the Jackrabbit documentation, in order to update this file.

Below are excerpts from a `repository.xml` for Postgres that has been updated to use the JNDI connection named `jackrabbit` as noted earlier. The same update will be required for each of the connections within the file.

```
<FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
 <param name="driver" value="javax.naming.InitialContext"/>
 <param name="url" value="java:comp/env/jdbc/jackrabbit"/>
 <param name="schema" value="postgresql"/>
 <param name="schemaObjectPrefix" value="fs_repos_"/>
</FileSystem>
<DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
 <param name="driver" value="javax.naming.InitialContext"/>
 <param name="url" value="java:comp/env/jdbc/jackrabbit"/>
 <param name="databaseType" value="postgresql"/>
 <param name="minRecordLength" value="1024"/>
 <param name="maxConnections" value="3"/>
 <param name="copyWhenReading" value="true"/>
 <param name="tablePrefix" value=""/>
 <param name="schemaObjectPrefix" value="ds_repos_"/>
</DataStore>
<PersistenceManager

class="org.apache.jackrabbit.core.persistence.bundle.PostgreSQLPersi
stenceManager">
 <param name="driver" value="javax.naming.InitialContext"/>
 <param name="url" value="java:comp/env/jdbc/jackrabbit"/>
 <param name="schema" value="postgresql"/
 <param name="schemaObjectPrefix" value="${wsp.name}_pm_ws_"/>
</PersistenceManager>
```

Step 5: Start and Restart the Server

Now that everything is configured, start the Pentaho servers and continue the configuration of the servers as documented. Refer to the Start and Stop the Pentaho Server documentation for the necessary remaining steps to prepare the servers.

Using JNDI for Other Data Sources

JNDI should be used for relational data source connections, whenever possible. This includes any connections that use a Java Database Connectivity (JDBC) driver. The process for defining those within Tomcat as described in the Define JNDI Connections article in the online documentation. The same process should be followed for the Pentaho DI Server. Refer to the Move Pentaho Managed Data Sources to JNDI and Define JNDI Connections for Report Designer and Metadata Editor articles on how to use the JNDI data sources for the design tools.

Related Information

Please visit the following links for more information about topics discussed in this document.

Pentaho Documentation:

- Select Business Analytics (BA) Installation Options
- Select Data Integration (DI) Installation Options
- Database Access Protocol Decision Table
- Define JNDI Connections for the BA Server
- Installing the BA and DI Server
- Initialize and Configure the Repository
- Specify Connections for the BA Server
- Specify Connections for the DI Server
- Start DI Server
- Start BA Server
- Define JNDI Connections for Report Designer and Metadata Editor
- Move Pentaho Managed Data Sources to JNDI

Pentaho Best Practice:

- Configure the Solution Repository

Apache Tomcat:

- Tomcat's JNDI Resources HOW-TO
- XML configuration files

Apache Wiki:

- Tomcat's guidance
- Jackrabbit documentation

General Resources:

- Apache Jackrabbit
- Quartz
- JDev
- Experts Exchange
- WebSphere Journal

Best Practice Check List

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed.

Name of the Project: ____

Date of the Review: ____

Name of the Reviewer: ____

Item	Response	Comments
Did you create the solution repository?	YES ___ NO ___	
Did you configure the solution repository?	YES ___ NO ___	
Did you specify the connections?	YES ___ NO ___	
If needed, did you update the solution repository to use JNDI?	YES ___ NO ___	
Did you start and restart the server?	YES ___ NO ___	