Configuration in Enhydra Enterprise Server

Tanja Jovanovic

Table of Contents

List of Examples

Chapter 1. JNDI

Introduction

The Java Naming and Directory Interface (JNDI) is part of the Java platform, providing applications based on Java technology with a unified interface to multiple naming and directory services.

A fundamental facility in any computing system is the naming service - the means by which names are associated with objects and objects are found based on their names.

JNDI Overview

The Java Naming and Directory Interface (JNDI) is an application programming interface (API) that provides naming and directory functionality to applications written using the java programming language. It is defined to be independent of any specific directory service implementation.

Architecture

The JNDI architecture consists of an API and a service provider interface (SPI). Java applications use the JNDI API to access a variety of naming and directory services. The SPI enables a variety of naming and directory services to be plugged in transparently, by allowing the Java application using the JNDI API to access their services.

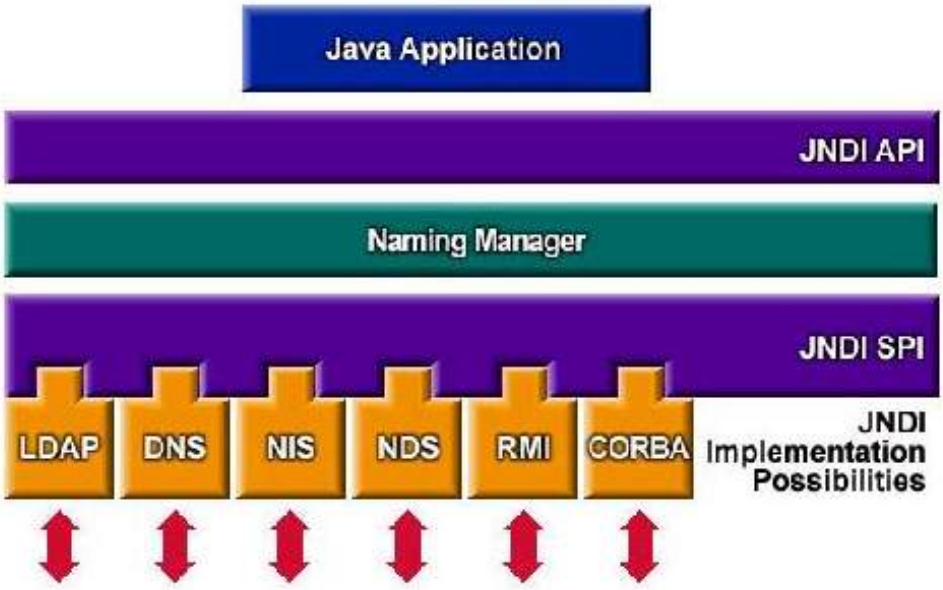

Figure 1: JNDI Architecture

Packaging

The JNDI is included in the Java 2 SDK, v1.3 [http://java.sun.com/j2se/1.3/] and later releases. It is also available as a Java Standard Extension for use with the JDK 1.1 and the Java 2 SDK, v1.2. It extends the v1.1 and v1.2 platforms to provide naming and directory functionality.

To use the JNDI, you must have the JNDI classes and one or more service providers. The Java 2 SDK, v1.3 includes three service providers for the following naming/directory services:

- Lightweight Directory Access Protocol (LDAP)
- Common Object Request Broker Architecture (CORBA) Common Object Services (COS) name service
- Java Remote Method Invocation (RMI) Registry

Other service providers can be downloaded from the JNDI Web site [http://java.sun.com/products/jndi/serviceproviders.html] or obtained from other vendors. When using the JNDI as a Standard Extension on the JDK 1.1 and Java 2 SDK, v1.2, you must first download [http://java.sun.com/products/jndi/] the JNDI classes and one or more service providers.

The JNDI is divided into five packages:

- javax.naming
- javax.naming.directory

- javax.naming.event
- javax.naming.ldap
- javax.naming.spi

You can find out more about JNDI at http://java.sun.com/products/jndi/tutorial/trailmap.html.

Chapter 2. Selection of application configuration file

Introduction

Application configuration can be read from application's <appName>.conf file. Since Enhydra 6.1, the configuration can also be read from application's web.xml file. Two parameters that define which file will be used are *ConfFile* and *ConfFileClass*. They are defined in web.xml in

```
<init-param>
```

tag.

NOTE: <appName> is the name of the application.

The first parameter is obligatory. It contains the name of the configuration file (with the path relative to directory where web.xml file is) that will be used for reading configuration.

The second parameter contains the name of the class (with full package) that will be used for creation of configuration file and for reading application configuration. It can have the following values:

- `com.lutris.util.ConfigFile`

  - for <appName>.conf file

- `org.enhydra.util.XMLConfigFile`

  - for web.xml file

The default value is

```
com.lutris.util.ConfigFile.
```

Example 2.1. Reading configuration from <appName>.conf file

```
<init-param>
  <param-name>ConfFile</param-name>
  <param-value>../conf/discRack.conf</param-value>
</init-param>
<init-param>
  <param-name>ConfFileClass</param-name>
  <param-value>com.lutris.util.ConfigFile</param-value>
</init-param>
```

Example 2.2. Reading configuration from web.xml file

```
<init-param>
  <param-name>ConfFile</param-name>
  <param-value>web.xml</param-value>
</init-param>
<init-param>
  <param-name>ConfFileClass</param-name>
  <param-value>org.enhydra.util.XMLConfigFile</param-value>
</init-param>
```

Tag <init-param> in web.xml file

As described in XML DTD Servlet 2.3 specification for web-app, the tag <init-param> has the following form:

```
<!ELEMENT init-param (param-name, param-value, description?)>
```

The init-param element contains a name/value pair as an initialization param of the servlet.

The param-name element contains the name of a parameter. Each parameter name must be unique in the web application.

Form:

```
<!ELEMENT param-name (#PCDATA)>
```

The param-value element contains the value of a parameter.

Form:

```
<!ELEMENT param-value (#PCDATA)>
```

The description element is used to provide text describing the parent element. The description element should include any information that the web application war file producer wants to provide to the consumer of the web application war file (i.e., to the Deployer). Typically, the tools used by the web application war file consumer will display the description when processing the parent element that contains the description.

Form:

```
<!ELEMENT description (#PCDATA)>
```

Chapter 3. Application parameters

Introduction

Since Enhydra 6.1, application parameters are read via JNDI. Parameters can be read from application's web.xml file, or from its <appName>.conf file.

Parameters in <appName>.conf file

As in the pervious Enhydra versions, application parameters can be defined in application's conf file (<appName>.conf file).

Example 3.1. Application parameters defined in <appName>.conf file:

```
Server.XMLC.AutoRecompilation = false
SessionManager.Lifetime = 60
Application.DefaultUrl = "personMgmt/Login.po"
```

The section delimiter is "." sign.

Parameters in web.xml file

From Enhydra 6.1 version, application parameters can be defined in application's web.xml file. Tag <env-entry> of web.xml is used for this purpose.

NOTE: Application parameters are read from web.xml which is in application war file (for example, *web.xml* from *discRack.war*).

Example 3.2. Application parameters defined in web.xml file:

```
<env-entry>
  <env-entry-name>SessionManager/Lifetime</env-entry-name>
  <env-entry-value>60</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

The section delimiter is "/" sign.

Tag <env-entry> in web.xml file

As described in XML DTD Servlet 2.3 specification for web-app, the tag <env-entry> has the following form:

```
<!ELEMENT env-entry (description?, env-entry-name,
                     env-entry-value?, env-entry-type)>
```

The env-entry element contains the declaration of a web application's environment entry. The declaration consists of an optional description, the name of the environment entry, and an optional value. if a value is not specified, one must be supplied during deployment.

The env-entry-name element contains the name of a web applications's environment entry. The name is a JNDI name relative to the java:comp/env context. The name must be unique within a web application.

Form:

```
<!ELEMENT env-entry-name (#PCDATA)>
```

Example:

```
<env-entry-name>minAmount</env-entry-name>
```

The env-entry-type element contains the fully-qualified Java type of the environment entry value that is expected by the web application's code.

The following are the legal values of env-entry-type:

- java.lang.Boolean
- java.lang.Byte
- java.lang.Character
- java.lang.String
- java.lang.Short
- java.lang.Integer
- java.lang.Long
- java.lang.Float
- java.lang.Double

Form:

```
<!ELEMENT env-entry-type (#PCDATA)>
```

All parameters in Enhydra applications are type java.lang.String.

The env-entry-value element contains the value of a web application's environment entry. The value must be a String that is valid for the constructor of the specified type that takes a single String parameter, or for java.lang.Character, a single character.

Form:

```
<!ELEMENT env-entry-value (#PCDATA)>
```

Example:

```
<env-entry-value>100.00</env-entry-value>
```

The description element is used to provide text describing the parent element. The description element should include any information that the web application war file producer wants to provide to the consumer of the web application war file (i.e., to the Deployer). Typically, the tools used by the web application war file consumer will display the description when processing the parent element that contains the description.

Form:

```
<!ELEMENT description (#PCDATA)>
```

Chapter 4. Order of reading configuration parameters

When an Enhydra application is being started, JOnAS reads (via JNDI) applications configuration parameters from web.xml located in application's war file. So, no matter which file is used for application configuration file (<appName>.conf or web.xml), during the application's start, first are read parameters from web.xml file in war file (if defined there).

After that, depending on which configuration file is used, the parameters are read from that configuration file. If any of application parameters is defined both in web.xml in war and in unpacked configuration file (<appName>.conf or web.xml) the value defined in unpacked file overrides the the value defined in web.xml in application's war file.

Chapter 5. Adjustments for reading configuration from <appName>.conf file

In order to read application parameters from <appName>.conf file, check if the following things are done:

In *carol.properties* file (in <ENHYDRA_HOME>/conf directory), in the list of active service providers (after default service provider - jrmp, iiop, jeremie, or cmi) should be added JNDI service provider for reading .conf parameters.

NOTE: <ENHYDRA_HOME> is directory where Enhydra is installed.

Example 5.1. Setting service provider for reading conf file

For example, we'll call this new service provider conffile

```
# jonas rmi acativation (jrmp, iiop, jeremie, cmi)
carol.protocols=jrmp, conffile
```

Also, configuration about this service provider should be added in this file (carol.properties).

Example 5.2. Configuration for conf service provider

```
###############################################################################
#                   Configuration for conf file                               #
###############################################################################
# portable remote object delegate class for this protocol (class name with package)
carol.conffile.PortableRemoteObjectClass=org.enhydra.util.spi.ConfFilePRODelegate
# Name service class for this protocol
carol.conffile.NameServiceClass=org.enhydra.util.spi.ConfFileRegistry
# java.naming.factory.initial property
carol.conffile.context.factory=org.enhydra.util.spi.ConfFileInitialContextFactory
# java.naming.provider.url property (only for carol, no importance)
carol.conffile.url=conffile://nohost:0
```

Service provider for configuration file is in jndiConfSpi.jar. This jar file should be placed in ENHY-<DRA_HOME>/lib/ext/enhydra/tools/jndi directory.

Chapter 6. Adjustments for reading configuration from unpacked web.xml file

In order to read application parameters from web.xml file, check if the following things are done:

In *carol.properties* file (in <ENHYDRA_HOME>/conf directory), in the list of active service providers (after default service provider - jrmp, iiop, jeremie, or cmi) should be added JNDI service provider for reading parameters from unpacked web.xml file.

NOTE: <ENHYDRA_HOME> is directory where Enhydra is installed.

Example 6.1. Setting service provider for reading unpacked web.xml file

For example, we'll call this new service provider webxmlfile

```
# jonas rmi acativation (jrmp, iiop, jeremie, cmi)
carol.protocols=jrmp, webxmlfile
```

Also, configuration about this service provider should be added in this file (carol.properties).

Example 6.2. Configuration for unpacked web.xml service provider

```
#########################################################################
#                   Configuration for web.xml (unpacked)                #
#########################################################################
# portable remote object delegate class for this protocol (class name with package)
carol.webxmlfile.PortableRemoteObjectClass=org.enhydra.util.spi.webxml.WebXmlPRODelegate
# Name service class for this protocol
carol.webxmlfile.NameServiceClass=org.enhydra.util.spi.webxml.WebXmlRegistry
# java.naming.factory.initial property
carol.webxmlfile.context.factory=org.enhydra.util.spi.webxml.WebXmlInitialContextFactory
# java.naming.provider.url property (only for carol, no importance)
carol.webxmlfile.url=webxmlfile://nohost:0
```

Service provider for configuration file is in jndiWebXmlSpi.jar. This jar file should be placed in EN-<HYDRA_HOME>/lib/ext/enhydra/tools/jndi directory.

In the situation when the configuration is read from web.xml and the service provider for reading unpacked web.xml (jndiWebXmlSpi.jar) doesn't exist, the default service provider will be used and the configuration parameters will be read only from the web.xml file located in application's war file.

Chapter 7. Using Jonas Admin

Database details

Since Enhydra 6.0, details about database that an application may use, can be entered by using Jonas Admin. This section explains what needs to be done in order to use this enhydra'a feature.

In jonas.properties file (in <ENHYDRA_HOME>/conf directory), services that need to be launched in the JOnAS Server are set. The propery that contains these values is jonas.services, and the services important for this enhydra's feature are: *jtm* and *dbm*.

Example 7.1. Setting jonas.properties for working with database resource

```
jonas.services jtm,dbm,security,web
```

In application configuration file should be added parameter */DatabaseManager/DB/<db_name>Connection/DataSourceName* for web.xml configuration file or parameter *DatabaseManager.DB.<db_name>.Connection.DataSourceName* for <appName>.conf configuration file. The value of this parameter has the following form:

```
jndi:<database_jndi_name>
```

where <database_jndi_name> is database's JNDI name under which the data (about the database the application uses) will be entered in Jonas Admin.

Example 7.2. Setting database JNDI name in <appName>.conf file

```
DatabaseManager.DB.sid1.Connection.DataSourceName = jndi:discRackBase3
```

Example 7.3. Setting database JNDI name in web.xml file

```
<env-entry>
  <env-entry-name>DatabaseManager/DB/sid1/Connection/DataSourceName</env-entry-name>
  <env-entry-value>jndi:discRackBase3</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Now, we go to Jonas Administration application (http://localhost:<communication_port>/jonasAdmin) to enter database data. The dafault <communication_port> is *9000*, the dafault user name is *admin* and the dafault password is *enhydra*.

When started, we go to part *Resources* and choose *Database (JDBC)*.

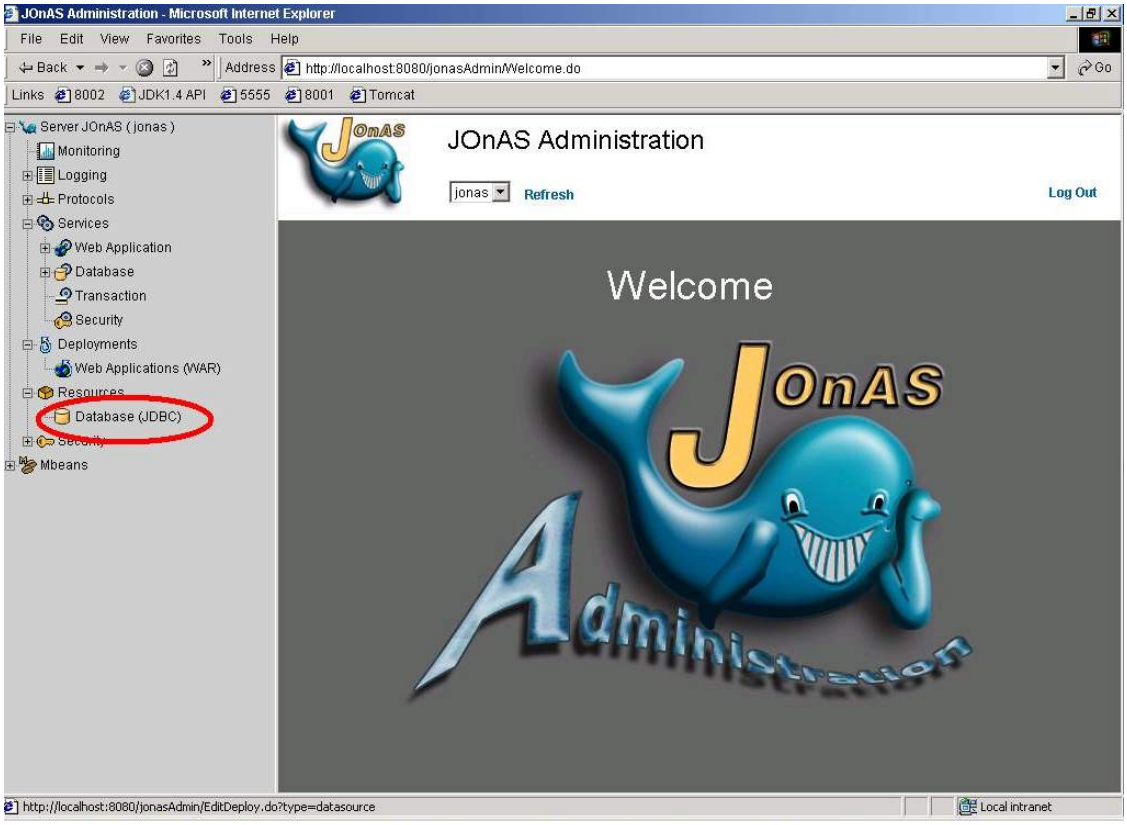

Figure 2: Database resource in Jonas Admin

We go to *Datasources* tab and click on *New datasource* button to add a new database.

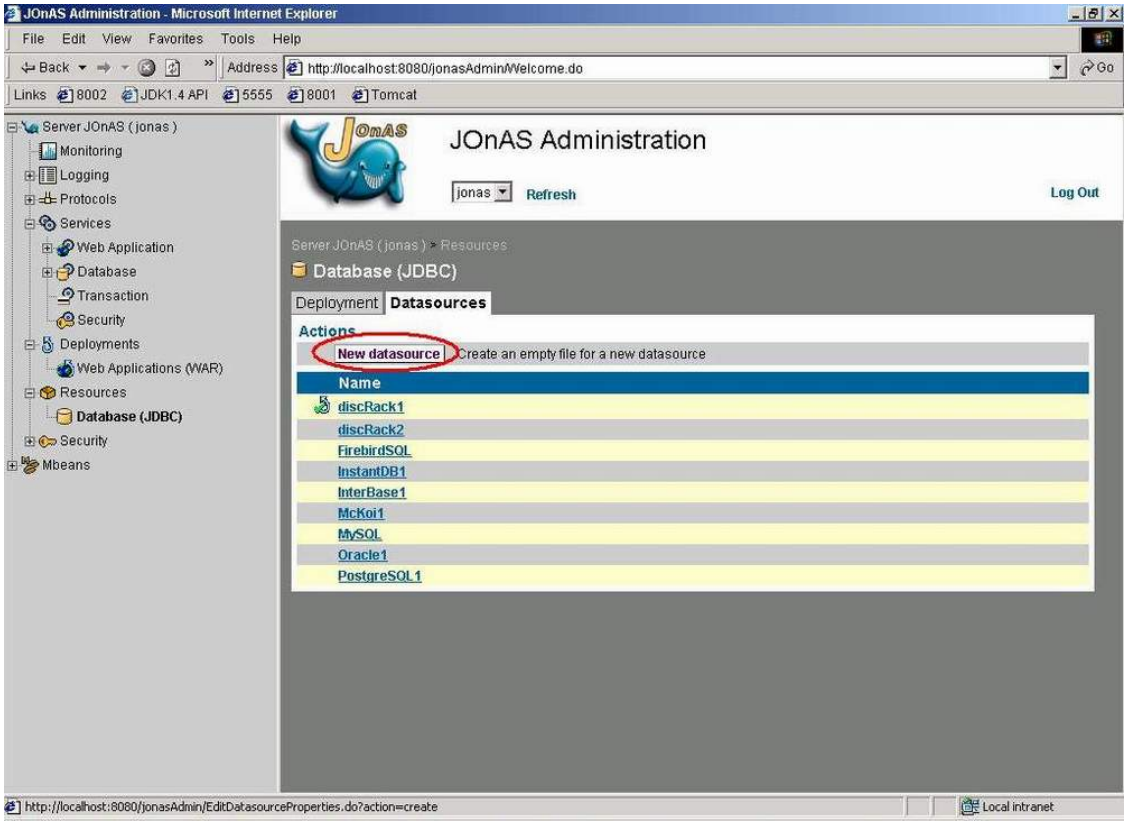

Figure 3: New datasource button

The window for entering database data will appear. The following information should be added:

- Name - File name of the datasource. Data entered here will be saved in file under this name (<Name>.properties) in <ENHYDRA_HOME>/conf directory.
- JNDI name - JNDI name of the database. In our example, that name is *discRackBase3*.
- Description - Description of the datasource.
- URL - URL to access to the database. This is the *DatabaseManager.DB.<db_name>.Connection.Url* parameter of <appName>.conf file or *DatabaseManager/DB/<db_name>/Connection/Url* parameter of web.xml file.
- JDBC Driver - JDBC driver class name to access to the database. This is the *DatabaseManager.DB.<db_name>.JdbcDriver* parameter of <appName>.conf file or *DatabaseManager/DB/<db_name>/JdbcDriver* parameter of web.xml file.
- User name - User name or login to log in the database. This is the *DatabaseManager.DB.<db_name>.Connection.User* parameter of <appName>.conf file or *DatabaseManager/DB/<db_name>/Connection/User* parameter of web.xml file.
- User password - User password to log in the database. This is the *DatabaseManager.DB.<db_name>.Connection.Password* parameter of <appName>.conf file or *DatabaseManager/DB/<db_name>/Connection/Password* parameter of web.xml file.

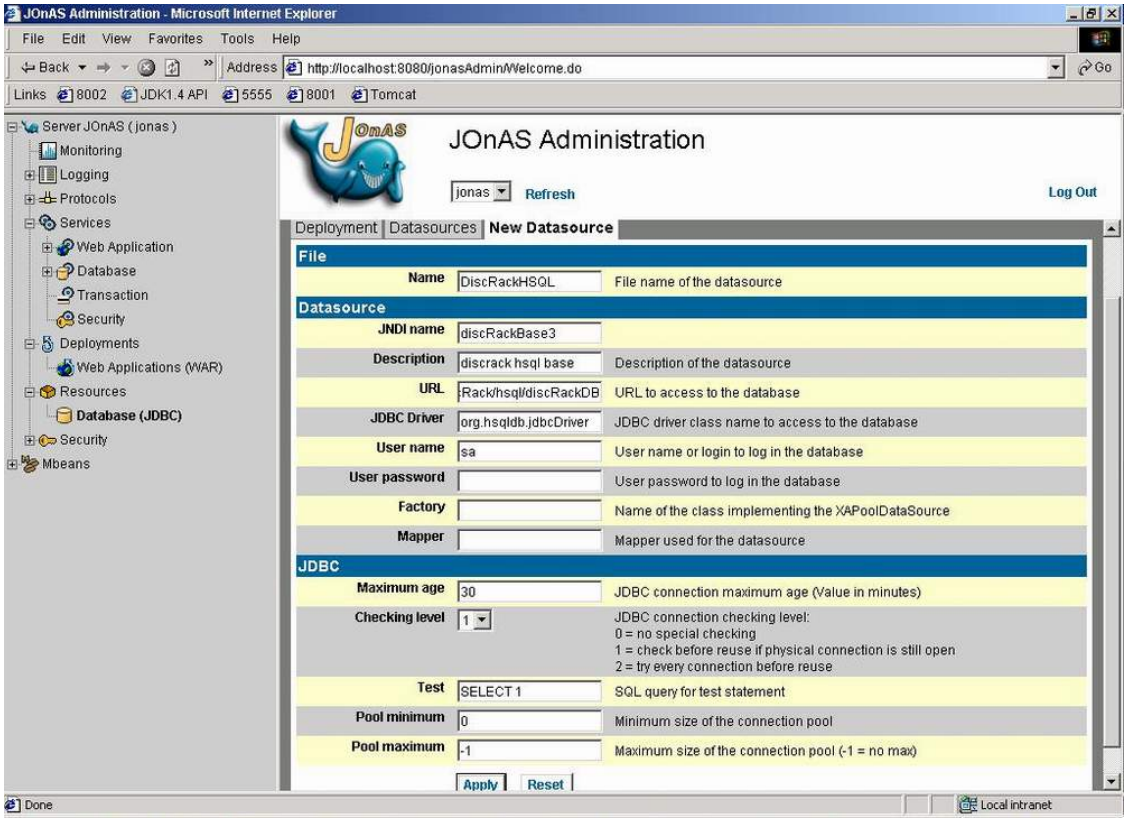

Figure 4: Filled datasource admin window

When all data are entered, we click on *Apply* button, and than on *Confirm* button after data check. In our example, file discRackHSQL.properties will be created in <ENHYDRA_HOME>/conf directory.

Now, this database (discRackHSQL) should be deployed so that can be used for reading database configuration for the application. All other databases (for that application) that won't be used, should be undeployed.

For example, the situation is: the database *discRack1*, that was used before for the discRack application, is deployed, and the database *discRackHSQL*, that will be used for the discRack application is undeployed.

Again, we go in Jonas Admin to part *Resources*, but now choose *Deployment* tab to see which databases are deployed, and which are not.

To undeploy *discRack1*, we have to choose this database in Deployed list and click on *Undeploy* button.

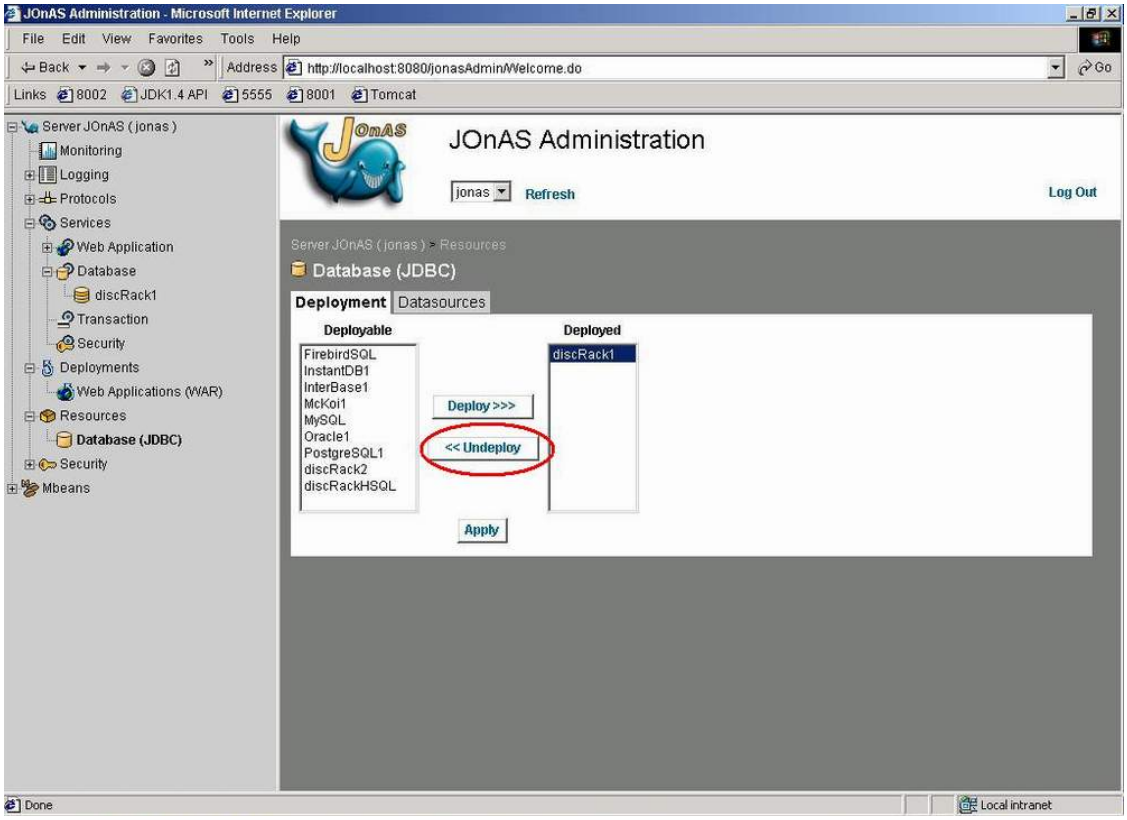

Figure 5: Database Undeployment

To deploy *discRackHSQL*, we have to choose this database in Deployable list and click on *Deploy* button.

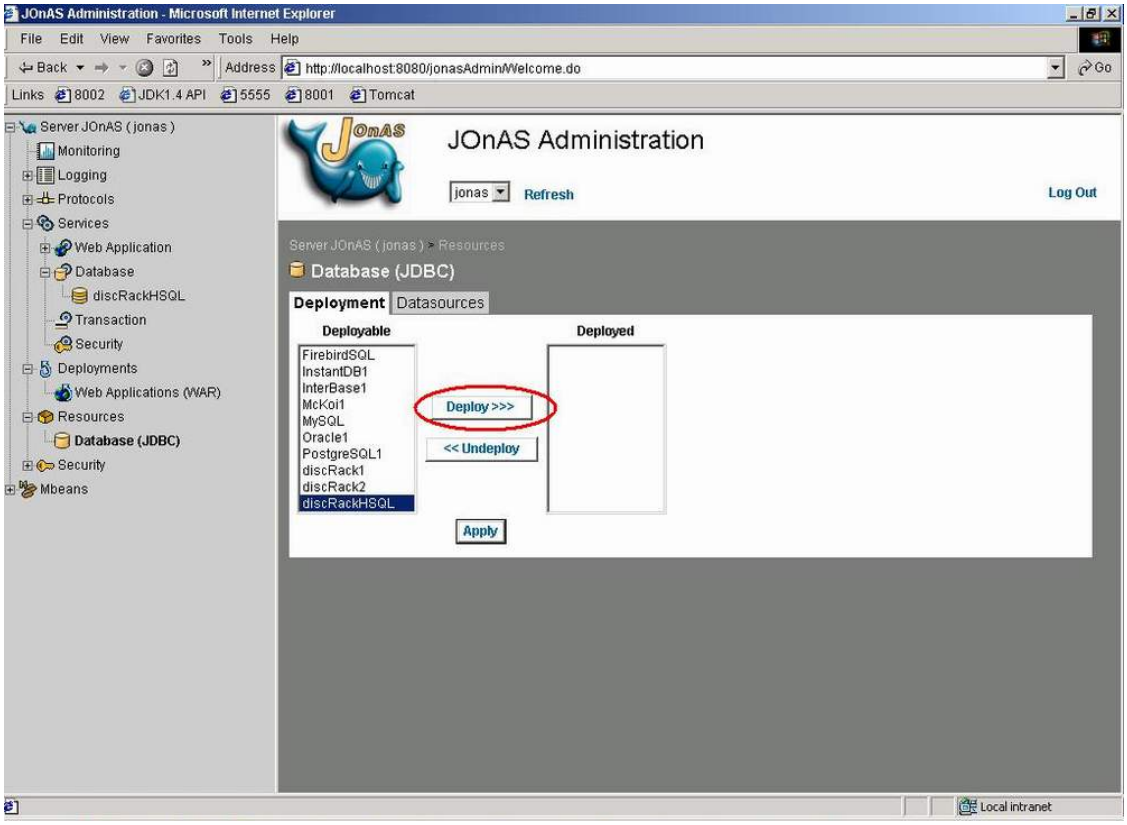

Figure 6: Database Deployment

To apply these operations, we click on *Apply* button, ant then on *Confirm* button.

After the database configuration parameters are added and deployed, the application must be restarted (or started if hasn't been active). This is done in Jonas Admin, in part *Deployments*, *Web Applications (WAR)*.

For example, we want to restart discRack application.

First, we have to undeploy the application (if deployed). We choose discRack.war in Deployed list, click on *Undeploy* button, then on *Apply* button and finally on the *Confirm* button. The application is undeployed.

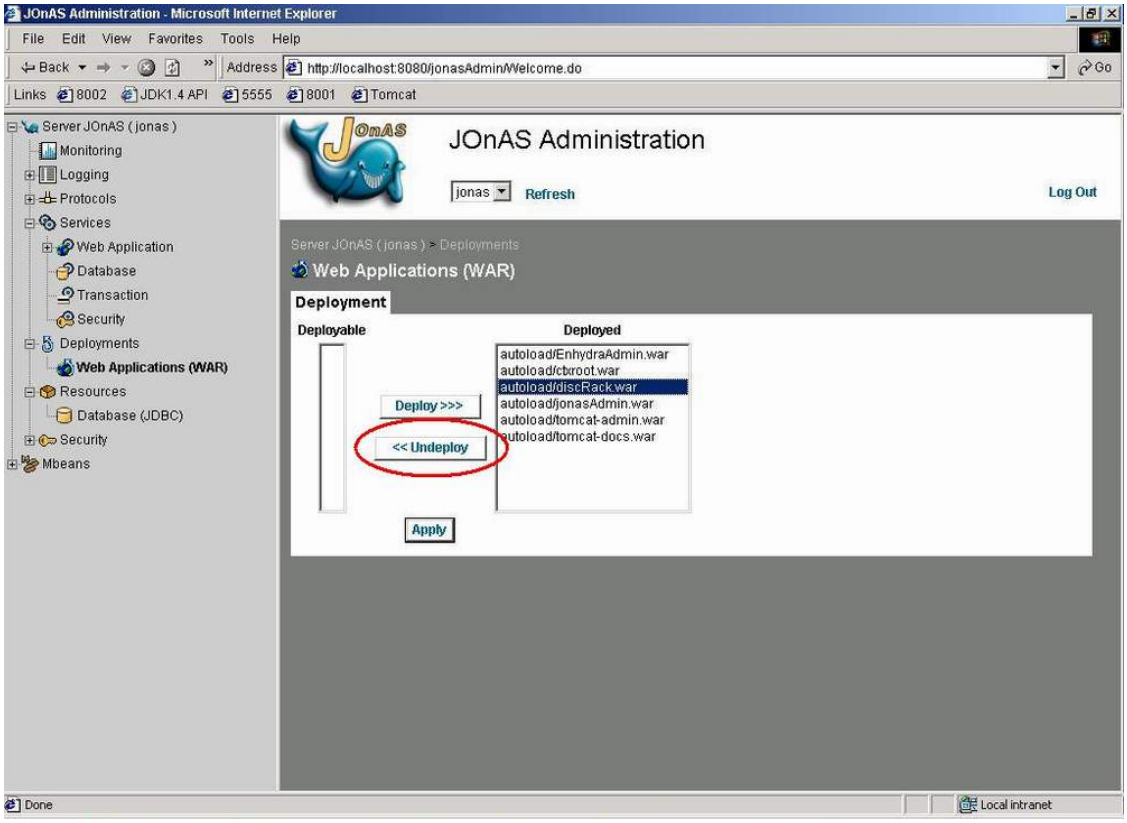

Figure 7: Application Undeployment

Second, we have to deploy the application. We choose discRack.war in Deployable list, click on *Deploy* button, then on *Apply* button and finally on the *Confirm* button. The application is deployed.

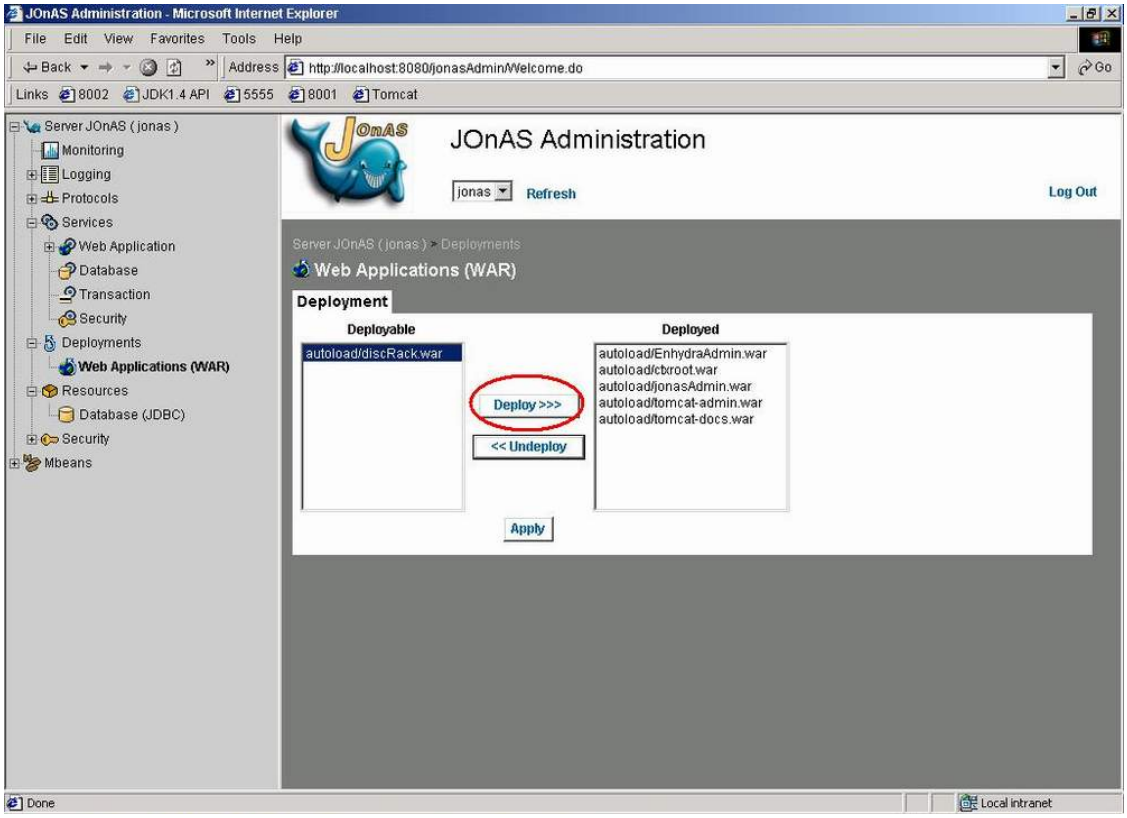

Figure 8: Application deployment

The restarted application will now work with discRackHSQL database.

NOTE: If the configuration data about the application database are entered by using Jonas Admin as described, the following parameters are not neccessary in the configuration file: *DatabaseManager.DB.<db_name>.Connection.Url*, *DatabaseManager.DB.<db_name>.JdbcDriver*, *DatabaseManager.DB.<db_name>.Connection.User* and *DatabaseManager.DB.<db_name>.Connection.Password* (for <appName>.conf file), or *DatabaseManager/DB/<db_name>/Connection/Url*, *DatabaseManager/DB/<db_name>/JdbcDriver*, *DatabaseManager/DB/<db_name>/Connection/User* and *DatabaseManager/DB/<db_name>/Connection/Password* (for web.xml file). Instead of that, only *DatabaseManager.DB.<db_name>.Connection.DataSourceName* parameter (for <appName>.conf file) or *DatabaseManager/DB/<db_name>/Connection/DataSourceName* parameter (for web.xml file) must exist in configuration file.

This way we can change many different databases for the application. When enter data for every next base, please be carefull that the database JNDI name should be the same as the one defined in configuration file. In our example, this name is *discRackBase3*.

In the case we want any of databases to be deployed during the start of Jonas, we have to set property *jonas.service.dbm.datasources* in *jonas.properties* file (in <ENHYDRA_HOME>/conf directory). This property is set with a coma-separated list of Datasource properties (file names without the '.properties' suffix).

In this section we have already created the database *discRackHSQL.properties*, and, for example let us assume that we have already created another database called *pokerBase.properties*, and that we want to laod both databases during the Jonas start. In this case, we set *jonas.service.dbm.datasources* property to values:

```
jonas.service.dbm.datasources discRackHSQL,pokerBase
```

Next time we start JOnAS Server, it will load defined data sources (in our case discRackHSQL and pokerBase), related jdbc drivers and register the data sources into JNDI.

Chapter 8. Implementation details

Configuration file

As a consequence of the existance of more than one type of the configuration file (for now <appName>.conf and web.xml), the classes for configuration file have been reorganized. They consist of:

- Interface

  `org.enhydra.util.ConfigFileInterface`

  that defines methods that every configuration file should have.

- Abstract class

  `org.enhydra.util.AbsConfigFile`

  that implements

  `org.enhydra.util.ConfigFileInterface`

  and contains non specific code for all configuration files.

- Class

  `com.lutris.util.ConfigFile`

  that extends

  `org.enhydra.util.AbsConfigFile.`

  This class implements <appName>.conf config file and defines methods that are specific for this type of configuration file.

- Class

  `org.enhydra.util.XMLConfigFile`

  that extends

  `org.enhydra.util.AbsConfigFile.`

  This class implements web.xml config file and defines methods that are specific for this type of configuration file.

The configuration object

`com.lutris.util.Config`

(which contains application configuration datails) now works with

`org.enhydra.util.ConfigFileInterface`

interface.

For adding new type of configuration file, the class

```
org.enhydra.util.AbsConfigFile
```

should be extended (or directly interface

```
 org.enhydra.util.ConfigFileInterface
```

implemented), and in web.xml file parameters *ConfFile* (name of the configuration file with the path relative to directory where is web.xml file) and *ConfFileClass* (name of the class with full package that will be used for configuration file) should be set to new values.