

Deployit System Administration Manual

Version 3.9.1

Table of Contents

Table of Contents	2
Preface	3
Installing Deployit	3
Prerequisites	3
Server Requirements	3
Determining Harddisk Space Requirements	3
Unix Middleware Server Requirements	3
Windows Middleware Server Requirements	4
Extending Middleware Support	4
Client Requirements	4
GUI Clients	4
CLI Clients	4
Installation Procedure	5
Installing the Server	5
Installing the CLI	5
Running the Server Setup Wizard	6
Simple Setup	6
Manual Setup	7
Finishing the Setup Process	9
Changing the Admin Password	9
Changing the encryption key password	9
High Availability Setup	10
Configuring Deployit	10
Configuring Security	10
Configuring LDAP Security	10
Assigning a default role to all authenticated users.	12
Hiding internal server errors	12
Configuring the Repository	12
Using a Database	12
Clustering	15
Installing Plugins	16
Advanced configuration	16
Client security configuration	16
Logging	16
Configuring Logging	16
Audit log	17
Script log	17
Starting and Stopping Deployit	17
Maintaining Deployit	18
Creating Backups	18
Restoring Backups	19
Freeing up disk space	19

Preface

This manual describes how to install and configure Deployit.

Installing Deployit

This section contains information on the installation of the Deployit server.

Prerequisites

Server Requirements

To install the Deployit server, the following prerequisites must be met:

- **Operating system:** Windows or Unix-family operating system running Java.
- **Java Runtime Environment:** JDK 1.6 (Oracle, IBM or Apple)
- **RAM:** At least 2GB of RAM available for Deployit.
- **Harddisk space:** Sufficient harddisk space to store the Deployit repository. This depends on your usage of Deployit. See section **Determining Harddisk Space Requirements**.

Depending on the environment, the following may also be required:

- **Database:** Deployit's Jackrabbit repository supports a number of different databases. For more information see section **Configuring the Repository**.
- **LDAP:** To enable group-based security, an LDAP x.509 compliant registry is needed. For more information see section **Configuring LDAP Security**.

Determining Harddisk Space Requirements

The Deployit server itself only uses about 70MB of disk space. The main harddisk space usage comes from the repository which stores your deployment packages and deployment history. The size of the repository will vary from installation to installation but depends mainly on:

- the size and storage mechanism used for artifacts
- the number of packages in the system
- the number of deployments performed (more specifically, the amount of logging information stored)

Follow this procedure to obtain an estimate of the total required disk space:

- Install and configure Deployit for your environment as described in this document. Make sure you correctly set up the database- or file-based repository.
- Estimate the number of packages to be imported (either the total number or the number per unit of time) (`NumPackages`)
- Estimate the number of deployments to be performed (either the total number or the number per unit of time) (`NumDeployments`)
- Record the amount of disk space used by Deployit (`InitialSize`).
- Import a few packages using the GUI or CLI.
- Record the amount of disk space used by Deployit (`SizeAfterImport`).
- Perform a few deployments.
- Record the amount of disk space used by Deployit (`SizeAfterDeployments`).

The needed amount of disk space in total is equal to:

```
Space Needed = ((SizeAfterImport - InitialSize) * NumPackages) +
               ((SizeAfterDeployments - SizeAfterImport) * NumDeployments)
```

If `NumPackages` and `NumDeployments` are expressed per timeunit (e.g. the number of packages to be imported per month), then the end result represents the space needed per timeunit as well.

Unix Middleware Server Requirements

Unix-based middleware servers that Deployit interacts with must meet the following requirements:

- **SSH Access:** The target systems should be accessible by SSH from the Deployit server, i.e. they should run an SSH2 server. It is also possible to handle key-based authorization. Notes:
 - The SSH daemon on AIX is known to hang with certain types of SSH traffic.
 - For security, the SSH account that is used to access a host should have limited rights.
 - A variety of Linux distributions have made SSH require a TTY by default. This setting is incompatible with Deployit and is controlled by the `Defaults requiretty` setting in the `sudoers` file.
- **Credentials:** Deployit should be able to log in to the target systems using a login/password combination that allows it to perform at least the following Unix commands:
 - `cp`
 - `ls`
 - `mv`
 - `rm`
 - `mkdir`
 - `rmdir`

If the login user cannot perform these actions, Deployit can also use a `sudo` user that can execute these commands.

Windows Middleware Server Requirements

Windows-based middleware servers that Deployit interacts with must meet the following requirements:

- **File system access:** The target file system should be accessible via CIFS from the Deployit server.
- **Host access:** The target host should be accessible from the Deployit server via WinRM or Windows Telnet server running in *stream mode*.
- **Directory shares:** The account used to access a target system should have access to the host's administrative shares such as `C$`.
- **Ports:** For CIFS connectivity, port 445 on the target system should be accessible from the Deployit server. For Telnet connectivity, port 23 should be accessible from the Deployit server. For WinRM connectivity, port 5985 (HTTP) or port 5986 (HTTPS) should be accessible from the Deployit server.

Extending Middleware Support

It is possible to connect Deployit to middleware servers that do not support SSH, Telnet or WinRM. Using the Overthere remote execution framework, a custom *access method* can be created that connects to the server. See the **Customization Manual** for more details.

Client Requirements

GUI Clients

To use the Deployit GUI, you must meet the following requirements:

- **Web browser:** The following web browsers are supported:
 - IE 8.0 or up
 - Firefox
 - Chrome
 - Safari
- **Flash Player:** A flash player is required, versions 9.0 and up are supported.

CLI Clients

To use the Deployit CLI, you must meet the following requirements:

- **Operating system:** Windows or Unix-family operating system running Java.
- **Java Runtime Environment:** JDK 1.6 (Oracle, IBM or Apple)

Installation Procedure

To begin installing Deployit, first unpack the distribution archive. The distribution archive contains the following:

- Release notes describing the changes made in this version of Deployit.
- A server archive.
- A CLI archive.

Note: see the separate **Upgrade Manual** for instructions on how to upgrade Deployit from a previous version.

Installing the Server

Follow these steps to install the Deployit server application:

1. **Login to the server where the Deployit Server will be installed.** It is recommended to install Deployit Server as a non-root user, e.g. `deployit`.
2. **Create an installation directory**, e.g. `/opt/xebialabs/deployit`.
3. **Copy the Deployit Server archive to the directory.**
4. **Extract the archive into the directory.**

Deployit Server Directory Structure

Once the Deployit installation file is extracted, the following directory structure exists in the installation directory (in the remainder of the document this directory will be referred to as *DEPLOYIT_SERVER_HOME*):

- **bin:** contains the Server binaries
- **conf:** contains Server configuration files (this directory is only present once you have configured Deployit Server)
- **doc:** contains the Deployit product documentation
- **ext:** contains Server Java extensions. See the **Customization Manual** for more information.
- **hotfix:** contains hotfixes that fix issues with the Server software
- **importablePackages:** default location for importable packages
- **lib:** contains libraries that the Server needs
- **log:** contains Server log files (this directory is only present once you have started Deployit Server)
- **plugins:** contains the Deployit middleware plugins
- **recovery.dat:** stores tasks that are in progress for recovery purposes (this file is only present once you have started Deployit Server)

Installing the CLI

Follow these steps to install the Deployit CLI application:

1. **Login to the server where the Deployit CLI will be installed.**
2. **Create an installation directory.**
3. **Copy the Deployit CLI archive to the directory.**
4. **Extract the archive into the directory.**

Deployit CLI Directory Structure

Once the Deployit installation file is extracted, the following directory structure exists in the installation directory:

- **bin:** contains the CLI binaries
- **ext:** contains CLI Python extension scripts
- **hotfix:** contains hotfixes that fix issues with the CLI software
- **lib:** contains necessary libraries
- **plugins:** contains the CLI plugins

Running the Server Setup Wizard

Run the Deployit Setup Wizard to start the Deployit server and prepare it for use. The command `server.sh -setup` starts the wizard. If you want to stop the Setup Wizard at any time, enter `exitsetup`. All changes to the configuration will be discarded.

The Setup Wizard displays the following welcome message:

```
Welcome to the Deployit setup.
You can always exit by typing 'exitsetup'.
To re-run this setup and make changes to the Deployit server configuration
you can run server.cmd -setup on Windows or server.sh -setup on Unix.

Do you want to use the simple setup?
Default values are used for all properties. To make changes to the default
properties, please answer no.
Options are yes or no.
[yes]:
```

Answer **yes** (or press Enter) to use the simple setup. Simple setup makes it easy to quickly get started with Deployit and to use the product's default configuration. See **Simple Setup** for more information.

Answer **no** to use the manual setup. Manual setup provides explicit control over all Deployit settings. See **Manual Setup** for more information.

Note: if you installed Deployit in the same location before, the Setup Wizard will ask you whether you want to edit the existing configuration or create a new one. Answer **yes** (or press Enter) to edit the existing configuration. The Setup Wizard will load all settings from the existing configuration and allow you to choose simple or manual setup. Answer **no** to start over with an empty configuration.

Simple Setup

Using simple setup, the Setup Wizard will assume default values for all configuration parameters. Specifically, the following defaults will be used:

- The server will run with security enabled.
- The server will **not** use secure communication between the Deployit GUI and the Deployit server.
- The server will listen on Deployit's standard HTTP port (4516).
- The server will use '/' as the context root.
- The server will use a minimum of 3 and a maximum of 24 threads.
- Applications can be imported from the `importablePackages` directory.

The Setup Wizard will ask one more question:

```
Do you want Deployit to initialize the JCR repository?
Options are yes or no.
[yes]:
```

Answer **yes** (or press Enter) if you want the Deployit repository to be recreated. The Setup Wizard must have write access to the repository directory. Answer **no** to leave the repository intact. This option is useful if you already have an existing repository that you want to reuse.

If you answer **yes**, the Setup Wizard will ask the following questions to help you configure your repository:

```
The password encryption key protects the passwords stored in the repository.
Do you want to generate a new password encryption key?
Options are yes or no.
[yes]:
```

Deployit ships with a default encryption key that matches the encryption key used in earlier versions of Deployit. By answering **no**, you agree to use either the Deployit-provided key or any key you previously generated. Answer **yes** to generate a new key. When you do this, you have the option of locking the keystore with a password as well:

```

The password encryption key is optionally secured by a password.
Please enter the password you wish to use. (Use an empty password to avoid a password prompt when starting
Deployit.)
New password:
Re-type password:

```

If you want to secure the keystore with a password, enter the password here. You will need to provide this password to Deployit when it starts, either interactively via a prompt or via a command line parameter. If you don't want to use a password for the keystore, press enter.

See **Finishing the Setup Wizard** for completing the setup process.

Warning: if you choose to recreate the Deployit repository and you have installed Deployit in the same location before, any information stored in the repository will be lost.

Manual Setup

The manual setup procedure contains the following steps:

Secure Communication Configuration

The Setup Wizard will show the following message:

```

Would you like to enable SSL?
Options are yes or no.
[yes]:

```

Answer **no** to use regular unsecured communication between the GUI and the server. Continue with the **Http configuration** section.

Answer **yes** (or press Enter) if you want to use a secure connection from the GUI to the server.

If you answer **yes**, the Setup Wizard will ask the following question to help you configure secure communication:

```

Would you like Deployit to generate a keystore with a self-signed
certificate for you?
N.B.: Self-signed certificates do not work correctly with some versions
of the Flash Player and some browsers!
Options are yes or no.
[yes]:

```

Answer **yes** (or press Enter) if you want the Setup Wizard to generate a digital certificate automatically. The digital certificate is required to secure communication and is normally signed by a Certificate Authority (CA). The Setup Wizard can generate a *self-signed* certificate if there is no official certificate available. Beware that using a self-signed certificate may trigger security warnings in some Flash players and browsers. Continue with the **Http configuration** section.

Answer **no** if you want to use your own keystore. Deployit uses the built-in Jetty webserver to communicate with the GUI. Jetty requires a certificate with the name `jetty` to be present in the keystore.

The Setup Wizard prompts you for the following keystore information:

```

What is the path to the keystore?
[]:

What is the password to the keystore?
[]:

What is the password to the key in the keystore?
[]:

```

Enter the filesystem location of the keystore (for example, `mykeystore.jks`), the password to unlock the keystore and the password for the `Jetty` certificate in the keystore.

Http Configuration

The Setup Wizard shows the following questions:

```

What http bind address would you like the server to listen on?
[localhost]:

What http port number would you like the server to listen on?
[4516]:

Enter the web context root where Deployit will run
[/]:

```

Note: if you chose to enable secure communication, the default port will be *4517* instead of *4516*.

Enter the host name, port number and context root that the Deployit server listens on for connections.

Thread Configuration

The Setup Wizard shows the following questions:

```

Enter the minimum number of threads the HTTP server should use (recommended:
3 per client, so 3 for single user usage)
[3]:

```

Enter the minimum number of threads that the Deployit server uses to handle incoming connections. The recommended minimum number of threads is 3 per Deployit application client.

```

Enter the maximum number of threads the HTTP server should use (recommended :
3 per client, so 24 for 8 concurrent users)
[24]:

```

Enter the maximum number of threads that the Deployit server uses to handle incoming connections. The recommended maximum number of threads is 3 per Deployit application client.

Repository Configuration

The Setup Wizard shows the following questions:

```

Where would you like Deployit to store the JCR repository?
[repository]:

```

Enter the filesystem path to a directory where Deployit will create the repository. If the directory does not exist, the Setup Wizard will create it.

```

Do you want Deployit to initialize the JCR repository?
Options are yes or no.
[yes]:

```

Answer **no** to leave the repository intact.

Answer **yes** (or press Enter) if you want the Deployit repository to be recreated. The Setup Wizard must have write access to the repository directory. The Setup Wizard will ask the following questions to help you configure your repository:

```

The password encryption key protects the passwords stored in the repository.
Do you want to generate a new password encryption key?
Options are yes or no.
[yes]:

```

Deployit ships with a default encryption key that matches the encryption key used in earlier versions of Deployit. By answering **no**, you agree to use either the Deployit-provided key or any key you previously generated. Answer **yes** to generate a new key. When you do this, you have the option of locking the keystore with a password as well:

```

The password encryption key is optionally secured by a password.
Please enter the password you wish to use. (Use an empty password to avoid a password prompt when starting
Deployit.)
New password:
Re-type password:

```


If you want to secure the keystore with a password, enter the password here. You will need to provide this password to Deployit when it starts, either interactively via a prompt or via a command line parameter. If you don't want to use a password for the keystore, press enter.

Warning: if you choose to recreate the Deployit repository and you have installed Deployit in the same location before, any information stored in the repository will be lost.

Importable Packages Configuration

The Setup Wizard shows the following question:

```
Where would you like Deployit to import packages from?
[importablePackages]:
```

Enter the filesystem path to a directory from which Deployit will import packages. The Setup Wizard assumes that this directory exists once the Deployit server starts and will not create it.

Finishing the Setup Process

Once you have completed configuration of the setup process, the Setup Wizard displays an overview of all selected options. The following text is an example:

```
Do you agree with the following settings for Deployit and would you like
to save them?
Changes will be saved in deployit.conf
SSL will be disabled
HTTP bind address is localhost
HTTP port is 4516
HTTP server will use a minimum of 3 and a maximum of 24 threads
JCR repository home is at repository
JCR repository will be initialized.
Task recovery file will be deleted
Application import location is importablePackages
[yes]:
```

Answer **yes** (or press Enter) to store the configuration settings and end the Setup Wizard. If you selected the option to initialize the repository, this will be done now.

Answer **no** to abort the Setup Wizard.

If the Setup Wizard is successfully completed, it will display the following message:

```
You can now start your Deployit server by executing the command server.cmd
on Windows or server.sh on Unix.
Note: If your Deployit server is running please restart it.
Finished setup.
```

Changing the Admin Password

By default, Deployit is installed with a special user with administrative permissions. This user has the username `admin` and password `admin`. As the last step in the installation, the admin password should be changed to something more secure. Issue the following commands in the CLI to do this:

```
adminUser = security.readUser('admin')
adminUser.password = 'newpassword'
security.modifyUser(adminUser)
```

At this point, the Deployit server must be stopped. Once this has happened, set the new admin password in the `deployit.conf` file in the `conf` directory in the server installation directory so that Deployit can access the repository with administrative permissions.

Restart Deployit and test the new credentials in the CLI using the following commands:

```
security.login('admin', 'newpassword')
```

Changing the encryption key password

The passwords in the repository are encrypted with a secret key. This password encryption key is stored in a keystore file called `conf/repository-keystore.jceks`. This keystore file is optionally protected with a password. If a password is set, you need to enter

the password when the Deployit Server starts up.

To change the keystore password, you can use the `keytool` utility that is part of the Java JDK distribution. Usage:

```
keytool -storepasswd -keystore conf/repository-keystore.jceks -storetype jceks
```

There is one restriction however: `keytool` refuses to read or set passwords that are shorter than 6 characters. If you want to change a keystore with an empty or short password, download the third-party application **KeyStore Explorer**.

Note: The `repository-keystore.jceks` is one of the two keystore concepts in Deployit. This keystore only contains the key used for encryption of passwords in the repository. If you use HTTPS, Deployit will use a second keystore file to store the (self-signed) certificate.

High Availability Setup

Deployit can be configured to ensure maximum uptime of the application. In such a high availability setup, two instances of Deployit are running in an *active/passive* configuration. At any one time, only one Deployit instance is active but as soon as a failure is detected, the passive Deployit instance is activated and the failed instance is taken down for repair.

To configure Deployit for high availability, the Deployit repository must be used in *clustering* mode. This means that each Deployit node writes changes to a shared journal in addition to applying the change to its own repository. See the section **Configuring the Repository** below for more information on setting up clustering.

Configuring Deployit

This section contains information on the configuration of the Deployit server.

Configuring Security

See the separate **Security Manual** for more information about how to configure security.

Configuring LDAP Security

By default, Deployit authenticates users and retrieves authorization information from its repository. Deployit can also be configured to use an LDAP repository to authenticate users and to retrieve role (group) membership. In this scenario, the LDAP users and groups are used as principals in Deployit and can be mapped to Deployit roles. Role membership and rights assigned to roles are always stored in the JCR repository.

Deployit treats the LDAP repository as **read-only**. This means that Deployit will use the information from the LDAP repository, but can not make changes to that information.

To configure Deployit to use an LDAP repository, the security configuration file `deployit-security.xml` must be modified. This is an example of a working `deployit-security.xml` file that uses LDAP:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:security="http://www.springframework.org/schema/security"
xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-
3.1.xsd
http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-
security-3.1.xsd
">

<bean id="ldapServer" class="org.springframework.security.ldap.DefaultSpringSecurityContextSource">
<constructor-arg value="LDAP_SERVER_URL" />
<property name="userDn" value="MANAGER_DN" />
<property name="password" value="MANAGER_PASSWORD" />
<property name="baseEnvironmentProperties">
<map>
<entry key="java.naming.referral">
<value>ignore</value>
</entry>
</map>
</property>
</bean>

<bean id="ldapProvider" class="org.springframework.security.ldap.authentication.LdapAuthenticationProvider">
<constructor-arg>
<bean class="org.springframework.security.ldap.authentication.BindAuthenticator">
<constructor-arg ref="ldapServer" />
<property name="userSearch">
<bean id="userSearch"
class="org.springframework.security.ldap.search.FilterBasedLdapUserSearch">
<constructor-arg index="0" value="USER_SEARCH_BASE" />
<constructor-arg index="1" value="USER_SEARCH_FILTER" />
<constructor-arg index="2" ref="ldapServer" />
</bean>
</property>
</bean>
</constructor-arg>
<constructor-arg>
<bean class="org.springframework.security.ldap.userdetails.DefaultLdapAuthoritiesPopulator">
<constructor-arg ref="ldapServer" />
<constructor-arg value="GROUP_SEARCH_BASE" />
<property name="groupSearchFilter" value="GROUP_SEARCH_FILTER" />
<property name="rolePrefix" value="" />
<property name="searchSubtree" value="true" />
<property name="convertToUpperCase" value="false" />
</bean>
</constructor-arg>
</bean>

<bean id="rememberMeAuthenticationProvider"
class="com.xebialabs.deployit.security.authentication.RememberMeAuthenticationProvider"/>

<bean id="jcrAuthenticationProvider"
class="com.xebialabs.deployit.security.authentication.JcrAuthenticationProvider"/>

<security:authentication-manager alias="authenticationManager">
<security:authentication-provider ref="rememberMeAuthenticationProvider" />
<security:authentication-provider ref="ldapProvider" />
<security:authentication-provider ref="jcrAuthenticationProvider"/>
</security:authentication-manager>

</beans>

```

The XML fragment above contains placeholders for the following values:

- **LDAP_SERVER_URL**: The LDAP url to connect to. (example: "ldap://localhost:389/")
- **MANAGER_DN**: The principal to perform the initial bind to the LDAP server. (example: "cn=admin,dc=example,dc=com")
- **MANAGER_PASSWORD**: The credentials to perform the initial bind to the LDAP server. (example: "secret")
- **USER_SEARCH_FILTER**: The LDAP filter to determine the LDAP dn for the user that's logging in, { 0 } will be replaced with the username that's logging in (example: "(&(uid={0})(objectClass=inetOrgPerson))")
- **USER_SEARCH_BASE**: The LDAP filter that is the base for searching for users (example: "dc=example,dc=com")
- **GROUP_SEARCH_FILTER**: The LDAP filter to determine the group memberships for the user, { 0 } will be replaced with the DN of the user (example: "(memberUid={0})")
- **GROUP_SEARCH_BASE**: The LDAP filter that is the base for searching for groups (example: "ou=groups,dc=example,dc=com")

Deployit server will need to be restarted after you have configured LDAP access.

Assigning a default role to all authenticated users.

When your Ldap is not setup to contain a group that all deployit users are assigned to, or you want to utilize such a group in the default JcrAuthenticationProvider, it is possible to configure this in your deployit-security.xml file. The following snippet will setup an 'everyone' group (The name is arbitrary, choose a different one if you wish) that is assigned to each user who is authenticated. You could then link this group up to a deployit role, and assign 'login' privileges to it for instance.

```
<beans>
  ...

  <bean id="ldapProvider" class="org.springframework.security.ldap.authentication.LdapAuthenticationProvider">
    <constructor-arg>
      ...
    </constructor-arg>

    <property name="authoritiesMapper" ref="additionalAuthoritiesMapper" />
  </bean>

  <bean id="jcrAuthenticationProvider"
    class="com.xebialabs.deployit.security.authentication.JcrAuthenticationProvider">
    <property name="authoritiesMapper" ref="additionalAuthoritiesMapper" />
  </bean>

  <bean id="additionalAuthoritiesMapper" class="com.xebialabs.deployit.security.AdditionalAuthoritiesMapper">
    <property name="additionalAuthorities">
      <list>
        <value>everyone</value>
      </list>
    </property>
  </bean>
</beans>
```

Hiding internal server errors

By default Deployit will not hide any internal server errors due to incorrect user input. This allows clients to more easily determine what went wrong and report problems with the Deployit Support team. However this behaviour can be turned off by editing the `conf/deployit.conf` file in the deployit server directory and edit the following setting:

```
hide.internals=true
```

Enabling this setting will cause the server to return a response such as the following:

```
Status Code: 400
Content-Length: 133
Server: Jetty(6.1.11)
Content-Type: text/plain

An internal error has occurred, please notify your system administrator with the following code: a3bb4df3-1eal-40c6-a94d-33a922497134
```

The code shown in the response can be used to track down the problem in the server logging.

Configuring the Repository

Deployit uses a repository to store all of it's data such as CIs, deployment packages, logging, etc. Deployit can use the filesystem or a database for binary artifacts (deployment packages) and CIs and CI history.

Out of the box, Deployit uses the filesystem to store all data in the repository.

Using a Database

Deployit can also use a database to store its repository. The built-in Jackrabbit JCR implementation must be configured to make this possible.

There are several configuration options when setting up a database repository:

- Store **only binary artifacts** in a database. This requires configuring the *DataStore* property.
- Store **only CIs and CI history** in a database. This requires configuring the *PersistenceManager* and *FileSystem* properties.

- Store **all data** (binary artifacts and CIs and CI history) in a database. This requires configuring the *DataStore*, *PersistenceManager* and *FileSystem* must be configured.

Here are some examples of configuring Deployit to use a database for various database vendors. The XML snippets below must be put into the `conf/jackrabbit-repository.xml` file.

Note: Deployit **must** initialize the repository before it can be used. Run Deployit's setup wizard and initialize the repository after making any changes to the repository configuration.

For more information about using a database with Jackrabbit, see it's [PersistenceManager FAQ](#) and [DataStore FAQ](#).

Using Deployit with MySQL

```
<DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
  <param name="driver" value="com.mysql.jdbc.Driver"/>
  <param name="url" value="jdbc:mysql://localhost:3306/deployit"/>
  <param name="databaseType" value="mysql"/>
  <param name="user" value="deployit" />
  <param name="password" value="deployit" />
</DataStore>

<Workspace name="${wsp.name}">
  <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
    <param name="driver" value="com.mysql.jdbc.Driver"/>
    <param name="url" value="jdbc:mysql://localhost:3306/deployit"/>
    <param name="schemaObjectPrefix" value="${wsp.name}_" />
    <param name="schema" value="mysql" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
  </FileSystem>

  <PersistenceManager class="org.apache.jackrabbit.core.persistence.pool.MySqlPersistenceManager">
    <param name="driver" value="com.mysql.jdbc.Driver"/>
    <param name="url" value="jdbc:mysql://localhost:3306/deployit" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
    <param name="schemaObjectPrefix" value="${wsp.name}_" />
  </PersistenceManager>

  <SearchIndex class="org.apache.jackrabbit.core.query.lucene.SearchIndex">
    <param name="path" value="${wsp.home}/index" />
    <param name="supportHighlighting" value="true" />
  </SearchIndex>
</Workspace>

<Versioning rootPath="${rep.home}/version">
  <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
    <param name="driver" value="com.mysql.jdbc.Driver"/>
    <param name="url" value="jdbc:mysql://localhost:3306/deployit"/>
    <param name="schemaObjectPrefix" value="version_" />
    <param name="schema" value="mysql" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
  </FileSystem>

  <PersistenceManager class="org.apache.jackrabbit.core.persistence.pool.MySqlPersistenceManager">
    <param name="url" value="jdbc:mysql://localhost:3306/deployit" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
    <param name="schemaObjectPrefix" value="version_" />
  </PersistenceManager>
</Versioning>
```

Note: The MySQL database is not suited for storage of large binary objects, see [the MySQL bug tracker](#).

Using Deployit with DB2

```

<DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
  <param name="driver" value="com.ibm.db2.jcc.DB2Driver"/>
  <param name="url" value="jdbc:db2://localhost:50002/deployit"/>
  <param name="databaseType" value="db2"/>
  <param name="user" value="deployit" />
  <param name="password" value="deployit" />
</DataStore>

<Workspace name="${wsp.name}">
  <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
    <param name="driver" value="com.ibm.db2.jcc.DB2Driver"/>
    <param name="url" value="jdbc:db2://localhost:50002/deployit"/>
    <param name="schemaObjectPrefix" value="${wsp.name}_ " />
    <param name="schema" value="db2" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
  </FileSystem>

  <PersistenceManager class="org.apache.jackrabbit.core.persistence.pool.BundleDbPersistenceManager">
    <param name="driver" value="com.ibm.db2.jcc.DB2Driver"/>
    <param name="url" value="jdbc:db2://localhost:50002/deployit" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
    <param name="databaseType" value="db2" />
    <param name="schemaObjectPrefix" value="${wsp.name}_ " />
  </PersistenceManager>

  <SearchIndex class="org.apache.jackrabbit.core.query.lucene.SearchIndex">
    <param name="path" value="${wsp.home}/index" />
    <param name="supportHighlighting" value="true" />
  </SearchIndex>
</Workspace>

<Versioning rootPath="${rep.home}/version">
  <FileSystem class="org.apache.jackrabbit.core.fs.db.DbFileSystem">
    <param name="driver" value="com.ibm.db2.jcc.DB2Driver"/>
    <param name="url" value="jdbc:db2://localhost:50002/deployit"/>
    <param name="schemaObjectPrefix" value="version_ " />
    <param name="schema" value="db2" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
  </FileSystem>
  <PersistenceManager class="org.apache.jackrabbit.core.persistence.pool.BundleDbPersistenceManager">
    <param name="driver" value="com.ibm.db2.jcc.DB2Driver"/>
    <param name="url" value="jdbc:db2://localhost:50002/deployit" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
    <param name="databaseType" value="db2" />
    <param name="schemaObjectPrefix" value="version_ " />
  </PersistenceManager>
</Versioning>

```

Using Deployit with **Oracle**

```

<DataStore class="org.apache.jackrabbit.core.data.db.DbDataStore">
  <param name="driver" value="oracle.jdbc.OracleDriver"/>
  <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
  <param name="databaseType" value="oracle"/>
  <param name="user" value="deployit" />
  <param name="password" value="deployit" />
</DataStore>

<Workspace name="${wsp.name}">
  <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
    <param name="driver" value="oracle.jdbc.OracleDriver"/>
    <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
    <param name="schemaObjectPrefix" value="${wsp.name}_"/>
    <param name="schema" value="oracle" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
  </FileSystem>

  <PersistenceManager
    class="org.apache.jackrabbit.core.persistence.bundle.OraclePersistenceManager">
    <param name="driver" value="oracle.jdbc.driver.OracleDriver"/>
    <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
    <param name="databaseType" value="oracle" />
    <param name="schemaObjectPrefix" value="${wsp.name}_"/>
  </PersistenceManager>

  <SearchIndex class="org.apache.jackrabbit.core.query.lucene.SearchIndex">
    <param name="path" value="${wsp.home}/index" />
    <param name="supportHighlighting" value="true" />
  </SearchIndex>
</Workspace>

<Versioning rootPath="${rep.home}/version">
  <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
    <param name="driver" value="oracle.jdbc.OracleDriver"/>
    <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
    <param name="schemaObjectPrefix" value="version_"/>
    <param name="schema" value="oracle" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
  </FileSystem>

  <PersistenceManager
    class="org.apache.jackrabbit.core.persistence.bundle.OraclePersistenceManager">
    <param name="driver" value="oracle.jdbc.driver.OracleDriver"/>
    <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
    <param name="databaseType" value="oracle" />
    <param name="schemaObjectPrefix" value="version_" />
  </PersistenceManager>
</Versioning>

```

If you use the TNSNames Alias syntax to connect to Oracle, you may need to inform the driver where to find the TNSNAMES file. See the Oracle documentation for more details.

Clustering

It is also possible to run Deployit server with its repository shared with other Deployit server instances. For this to happen, the jackrabbit JCR must be configured to run in a [clustered mode](#). This needs a cluster configuration to be present in the *jackrabbit-repository.xml* file.

File-based repository

Add the following snippet to the *jackrabbit-repository.xml* to enable clustering:

```

<Cluster id="node1">
  <Journal class="org.apache.jackrabbit.core.journal.FileJournal">
    <param name="revision" value="${rep.home}/revision.log" />
    <param name="directory" value="/nfs/myserver/myjournal" />
  </Journal>
</Cluster>

```

In the above example, the property *directory* refers to the shared journal. Both Deployit instances must be able to write to the same journal.

Database repository

The following XML snippet shows a sample clustering configuration for a JCR using Oracle as its repository.

```
<Cluster id="101" syncDelay="2000">
  <Journal class="org.apache.jackrabbit.core.journal.OracleDatabaseJournal">
    <param name="revision" value="{rep.home}/revision" />
    <param name="driver" value="oracle.jdbc.driver.OracleDriver" />
    <param name="url" value="jdbc:oracle:thin:@localhost:1521:orcl" />
    <param name="user" value="deployit" />
    <param name="password" value="deployit" />
    <param name="schemaObjectPrefix" value="JOURNAL_" />
  </Journal>
</Cluster>
```

Note that each jackrabbit cluster node should have a unique value for `id`. For more information on JCR clustering, or ways to configure clustering using other databases, please refer to the jackrabbit [clustering documentation](#).

Installing Plugins

Deployit Server supports various plugins that add functionality to the system. When it starts, the Deployit server scans the `plugins` directory and loads all plugins it finds. The additional functionality they provide is immediately available. Any plugins added or removed when Deployit server is running will not take effect until the server is restarted.

Installing a Plugin

To install a new plugin, stop the Deployit server and copy the plugin JAR archive into the `plugins` directory, then restart the Deployit server.

Uninstalling a Plugin

To uninstall a plugin, stop the Deployit server and remove the plugin JAR archive from the `plugins` directory, then restart the Deployit server.

Advanced configuration

A number of advanced configuration options are available by editing the `deployit.conf` file in the `conf` directory. Restart the server after making the necessary changes.

- **taskThreadPool.corePoolSize:** The minimum number of threads allocated to execute tasks. Default value: 10.
- **taskThreadPool.maxPoolSize:** The maximum number of threads allocated to execute tasks. Default value: 2147483647.
- **taskThreadPool.keepAliveSeconds:** The number of seconds a time is kept alive before destroying it if the number of allocated threads exceeds.
- **taskThreadPool.queueCapacity:** The capacity of the queue that holds tasks to be executed if no threads are available. Default value: 2147483647.

Client security configuration

A number of client security configuration options are available by editing the `deployit.conf` file in the `conf` directory. Restart the server after making the necessary changes.

- **client.session.timeout.minutes:** The number of minutes it takes before the session of the user is locked when he is not using the Deployit GUI. The default value of '0' means that no time-out is configured.
- **client.session.remember.enabled:** Setting this value to false disables the "Keep me logged in" option on the login screen. Default value: true.

Logging

Configuring Logging

Out of the box, Deployit server writes informational, warning and error log messages to standard output as well as `log/deployit.log` when running. In addition, Deployit writes an audit trail to the file `log/audit.log`. It is possible to change Deployits logging behavior (for instance to write log output to a file or to log output from a specific source).

Deployit uses the Logback logging framework for it's logging. To change it's behavior, edit the file `logback.xml` in the `conf` directory of the Deployit server installation directory.

The following is an example `logback.xml` file:


```

<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
    <encoder>
      <pattern>
        %d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
      </pattern>
    </encoder>
  </appender>

  <!-- Create a file appender that writes log messages to a file -->
  <appender name="FILE" class="ch.qos.logback.core.FileAppender">
    <layout class="ch.qos.logback.classic.PatternLayout">
      <pattern>%-4relative [%thread] %-5level %class - %msg%n</pattern>
    </layout>
    <File>log/my.log</File>
  </appender>

  <!-- Set logging of classes in com.xebialabs to DEBUG level -->
  <logger name="com.xebialabs" level="debug"/>

  <!-- Set logging of class HttpClient to DEBUG level -->
  <logger name="HttpClient" level="debug"/>

  <!-- Set the logging of all other classes to INFO -->
  <root level="info">
    <!-- Write logging to STDOUT and FILE appenders -->
    <appender-ref ref="STDOUT" />
    <appender-ref ref="FILE" />
  </root>
</configuration>

```

For more information see the [Logback website](#).

Audit log

Deployit writes an audit log for each human-initiated event on the server. For each event, the following information is recorded:

- the user making the request
- the event timestamp
- the component producing the event
- an informational message describing the event

For events involving CIs, the CI data submitted as part of the event is logged in XML format.

These are some of the events that are logged in the audit trail:

- the system is started or stopped
- a user logs into or out of the system
- an application is imported
- a CI is created, updated, moved or deleted
- a security role is created, updated or deleted
- a task (deployment, undeployment, control task or discovery) is started, cancelled or aborted

By default, the audit log is stored in `log/audit.log`. The audit log is rolled over daily.

Script log

The default logging configuration file also contains a (commented out) section that enables logging of all Deployit scripts to a separate log file.

Starting and Stopping Deployit

Starting the Server

Open a terminal window and change to the `DEPLOYIT_SERVER_HOME` directory. Start the Deployit server with the command:

```
bin/server.sh
```

on Unix and

```
bin/server.cmd
```

on Windows.

By starting the server with the `-h` flag, a message is printed that shows the possible options it supports:

```
java -cp deployit-server-<version>.jar [options...] com.xebialabs.deployit.Deployit arguments...
-help                : Prints this usage message
-repository-keystore-password VAL : The password to open the repository keystore file, if not given,
                        Deployit will prompt you.
-reinitialize        : Reinitialize the repository, only useful with -setup
-setup               : (Re-)run the setup for Deployit
-setup-defaults VAL  : Use the given file for defaults during setup server.sh arguments...
```

The command line options are:

- `-repository-keystore-password VAL --` tells Deployit which password to use to access the repository keystore. If not specified and the repository keystore does require a password, Deployit will prompt you for it.
- `-reinitialize --` tells Deployit to reinitialize the repository. Used only in conjunction with `-setup`. **N.B.** This flag only works if Deployit is running on the filesystem repository, not when you've configured Deployit to run against a database.
- `-setup --` runs the Deployit Setup Wizard.
- `-setup-defaults VAL --` specifies a file that contains default values for configuration properties set in the Setup Wizard.

Server Options

Any options you want to give the Deployit Server when it starts can be specified in the `DEPLOYIT_SERVER_OPTS` environment variable.

Starting Deployit in the Background

By running the `server.sh` or `server.cmd` command, the Deployit server is started in the foreground. To run the server as a background process, use:

```
nohup bin/server.sh &
```

on Unix or run Deployit as a [service on Windows](#).

Stopping the Server

It is possible to stop the Deployit server using a REST API call. The following is an example of a command to generate such a call (replace `admin:admin` with your own credentials):

```
curl -X POST --basic -u admin:admin
http://admin:admin@localhost:4516/deployit/server/shutdown
```

This requires the external `curl` command, available for both Unix and Windows. The server can also be shutdown using the CLI.

Maintaining Deployit

This section describes how to maintain the Deployit server in your environment.

Creating Backups

To create a backup of Deployit, several components may need to be backed up depending on your configuration:

- **Repository.**
 - Built-in repository: Create a backup of the built-in JCR repository by backing up the files in the `repository` directory.
 - Database repository: Create a backup of both the files in the `repository` directory, as well as the database (using the tools provided by your database vendor).
- **Configuration.** Create a backup of the Deployit configuration by backing up the files in

the `conf` directory in the installation directory.

- **Customization.** Create a backup of the Deployit customizations by backing up the files in the `ext` and `plugins` directory in the installation directory.

Note: Deployit **must not** be running when you are making a backup. Schedule backups outside planned deployment hours to ensure the server is not being used.

Restoring Backups

To restore a backup of Deployit, restore one of the following components:

- **JCR repository.**
 - Built-in repository: Remove the `repository` directory and replace it with the backup.
 - Database repository: Remove the `repository` directory and replace it with the backup. Next, restore a backup of the database using the tools provided by your vendor.
- **Configuration.** Remove the `conf` directory in the `DEPLOYIT_SERVER_HOME` directory and replace it with the backup.
- **Customization.** Remove the `ext` and `plugins` directories in the `DEPLOYIT_SERVER_HOME` directory and replace them with the backups.

Note: Deployit **must not** be running when you are restoring a backup.

Freeing up disk space

The repository is the place where Deployit stores all of its data. If you deal with lots of large binary artifacts, this can be problematic since the disk space they consume is not reclaimed when they are deleted. To reclaim the disk space after deleting a CI, use the following CLI snippet:

```
deployit.runGarbageCollector()
```