CloverDX Server

User's Guide

CloverDX Server: User's Guide

This User's Guide refers to CloverDX Server 5.0.x release.

www.cloverdx.com

Feedback welcome:

If you have any comments or suggestions for this documentation, please send them by email to support@cloverdx.com.

Consider How to speed up communication with CloverCARE support before contacting the support.

Table of Contents

Part I. Introduction

Chapter 1. What is CloverDX Server?

CloverDX Server is an enterprise runtime, monitoring and automation platform for the **CloverDX** data integration suite. It is a Java application built to J2EE standards with HTTP and SOAP Web Services APIs providing an additional automation control for integration into existing application portfolios and processes.

CloverDX Server provides necessary tools to deploy, monitor, schedule, integrate and automate data integration processes in large scale and complex projects. **CloverDX Server** supports a wide range of application servers: Apache Tomcat, IBM WebSphere, JBoss EAP and Oracle WebLogic Server.

CloverDX Server simplifies the process of:

- *Operation* - **CloverDX Server** allows you to set up (p. 54) and monitor (p. 107) the status of the Server and jobs (p. 198) and notify you via an email (p. 172) if the job fails;

- *Automation* - It allows you to efficiently handle jobflow events via listeners (p. 203) and schedule (p. 192) tasks to be triggered as one-time events or repeatedly, as required;

- *Administration* - It helps you manage users (p. 128) and groups (p. 130) and their privileges, create and configure sandboxes (p. 144) and export the configuration (p. 155) to another instance of the Server. Furthermore, the Server provides API (p. 233) and allows you to create it via Data Services (p. 256);

- *Security* - For better control over **CloverDX Server**, you can set up user lockout (p. 142) and encrypt sensitive data (p. 119).

To learn more about the architecture of **CloverDX Server**, see Chapter 2, CloverDX Server Architecture (p. 3).
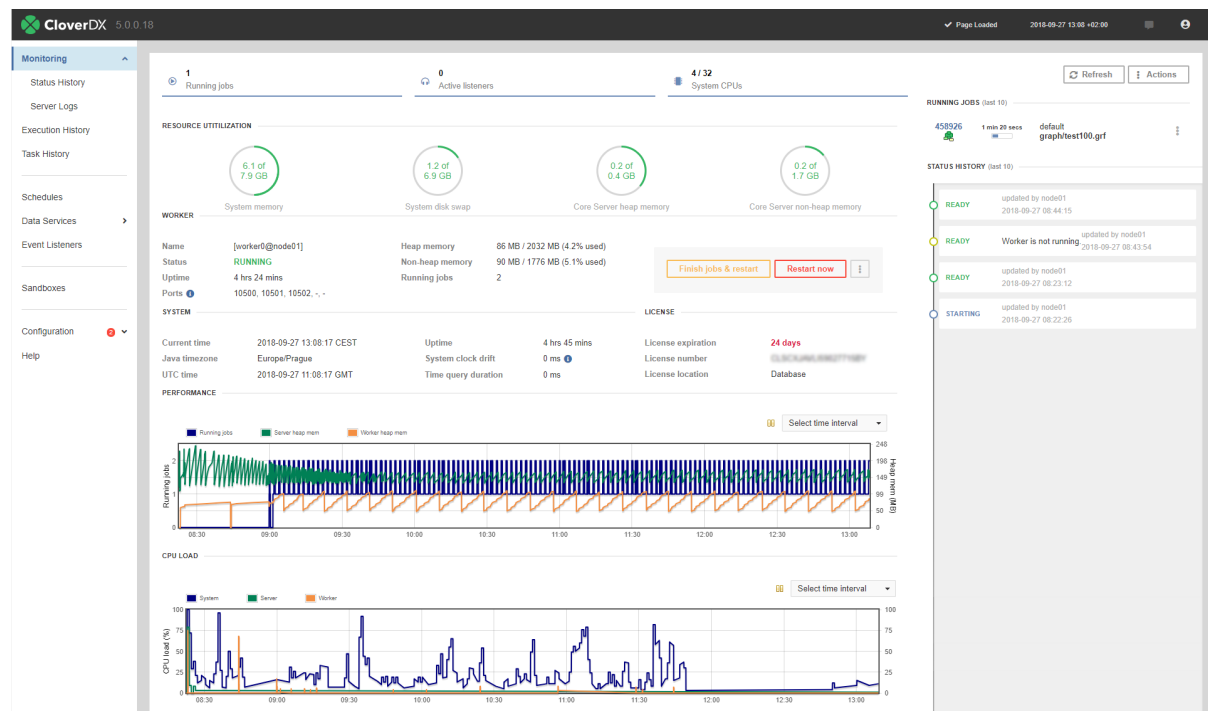
Figure 1.1. CloverDX Server User Interface

Chapter 2. CloverDX Server Architecture

CloverDX Server is a Java application distributed as a web application archive (.war) for an easy deployment on various application servers. It is compatible with Windows and Unix-like operating systems.

CloverDX Server requires Java Development Kit (JDK) to run. We **do not recommend** using Java Runtime Environment (JRE) only, since compilation of some transformations requires JDK to function properly.

The Server requires some space on the file system to store persistent data (transformation graphs) and temporary data (temporary files, debugging data, etc.). It also requires an external relational database to save run records, permission, users' data, etc.
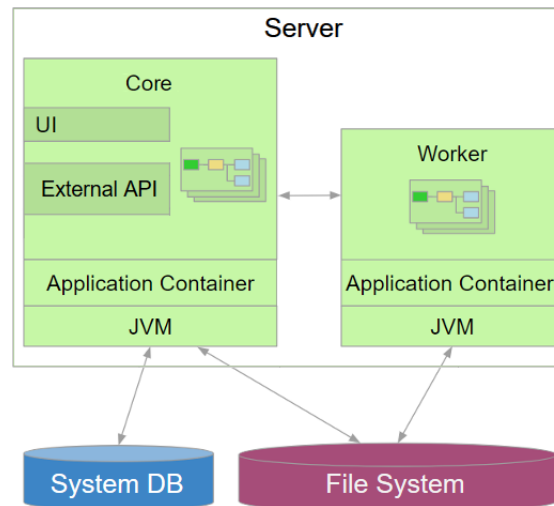
Figure 2.1. System Architecture

The **CloverDX Server** architecture consists of **Core** and **Worker**.

CloverDX Core

CloverDX Server's Core manages [users and groups](#) (p. 123), checks permissions, [schedules](#) (p. 192) execution and provides [management](#) (p. 54) and [monitoring](#) (p. 107) UI. It provides APIs for other applications: [Data Service API](#) (p. 256), [HTTP API](#) (p. 234) and [Web Service API](#) (p. 248). For more information, see Chapter 3, [CloverDX Core](#) (p. 4).

CloverDX Worker

Worker is a separate process that executes jobs: graphs, jobflows and profiler jobs. The purpose of Worker is to provide a sandboxed execution environment. For more information, see Chapter 4, [CloverDX Worker](#) (p. 5).

Dependencies on External Services

The Server requires a database to store its configuration, user accounts, execution history, etc. It comes bundled with an Apache Derby database to ease the evaluation. To use CloverDX Server in production environment, a [relational database](#) (p. 10) is needed.

The Server needs a connection to an [SMTP server](#) (p. 59) to be able to send you notification emails.

Users and groups' data can be stored in the database or read from an [LDAP server](#) (p. 124).

Server Core - Worker Communication

Server Core receives Worker's stdout and stderr. The processes communicate via TCP connections.

Chapter 3. CloverDX Core

CloverDX Core is the central point of **CloverDX Server**. It manages and monitors (p. 107)Workers (p. 5) that run the jobs. CloverDX Core launches scheduled (p. 192)jobs, checks user permissions (p. 130) sends notifications (p. 172) and forwards communication between Designer and Worker.

CloverDX Core is the visible part of the Server with a web-based user interface.

CloverDX Core connects to the system database (p. 63) and stores its configuration and service records in it. The system database is required. If it is configured, the Core connects to an SMTP server (p. 59) to send notification emails or to an LDAP server (p. 60) to authenticate users against an existing LDAP database.

Chapter 4. CloverDX Worker

Worker is a standalone JVM running separately from the Server Core. This provides an isolation of the Server Core from executed jobs (e.g. graphs, jobflows, etc.). Therefore, an issue caused by a job in Worker will not affect the Server Core.

Worker does not require any additional installation - it is started and managed by the Server. Worker runs on the same host as the Server Core, i.e. it is not used for parallel or distributed processes. In case of a Cluster, **each** Cluster node has its **own** Worker.

Worker is a relatively light-weight and simple executor of jobs. It handles job execution requests from the Server Core, but does not perform any high-level job management or scheduling. It communicates with the Server Core via an API for more complex activities, e.g. to request execution of other jobs, check file permissions, etc.

Configuration

General configuration

Worker is started by the Server Core as a standalone JVM process. The default configuration of Worker can be changed in the [Setup](#) (p. 58):

- Heap memory limits
- Port ranges
- Additional command line arguments (e.g. to tweak garbage collector (p. 40) settings)

The settings are stored in the usual Server configuration file. Worker is configured via special [configuration properties](#) (p. 90).

A full [command line](#) (p. 115) of Worker is available in the Monitoring section.

Cluster specific configuration

Cluster should use a single [portRange](#) (p. 90): all nodes should have identical value of `portRange`. That is the preferred configuration, although different ranges for individual nodes are possible.

Management

The Server manages the runtime of Worker, i.e. it is able to start, stop, restart Worker, etc. Users don't need to manually install and start Worker.

Status of Worker and actions are available in the [Monitoring Worker](#) (p. 108) section.

In case of problems with Worker, see Chapter 26, [Troubleshooting Worker](#) (p. 164).

Job Execution

By default, all jobs are executed in Worker; yet the Server Core still keeps the capability to execute jobs. It is possible to set specific jobs or whole sandboxes to run in the Server Core via the worker_execution (p. 151) property on the job or sandbox. It is also possible to disable Worker completely (p. 90), in which case all jobs will be executed in the Server Core.

Executing jobs in the Server Core should be an exception. To see where the job was executed, look in the run details in **Execution History** - in the Executor field. Jobs started in Worker also log a message in their log, e.g. `Job is executed on Worker:[worker0@node01:10500]`.

Job Configuration

The following areas of Worker configuration affect job execution:

- JNDI

Graphs running in Worker cannot use JNDI as defined in the application container of the Server Core, because Worker is a separate JVM process. Worker provides its own JNDI configuration (p. 93).

- Classpath

The classpath is not shared between the Server Core and Worker. If you need to add a library to the Worker classpath, e.g. a JDBC driver, follow the instructions in Adding Libraries to the Worker's Classpath (p. 41).

Chapter 5. CloverDX Cluster

CloverDX Cluster allows multiple instances of **CloverDX Server** to run on different hardware nodes and form a computer cluster. In this distributed environment, data transfer between **CloverDX Server** instances is performed by [Remote Edges](#) (p. 279).

CloverDX Cluster offers several advantages for Big Data processing:

- **High Availability** - All nodes are virtually equal; therefore, almost all request can be processed by any cluster node. This means that if one node is disabled, another node can substitute it.

  To achieve high availability, it is recommended to use an independent HTTP load balancer. See the configuration in the [2-nodes Cluster with Load Balancer](#) (p. 287) example.

- **Scalability** - Allows for increased performance by adding more nodes.

  There are two independent levels of scalability implemented. Scalability of transformation requests and data scalability.

For general information about **CloverDX Cluster**, see the **CloverDX Designer** documentation.

CloverDX Cluster is configured by a set of specific [Mandatory Cluster Properties](#) (p. 281) and [Optional Cluster Properties](#) (p. 282).

In Cluster environment, you can use several types of sandboxes, see Chapter 41, [Sandboxes in Cluster](#) (p. 276).

> ## Note
>
> **Note: CloverDX Cluster** requires a special license.

Part II. Installation

Chapter 6. System Requirements

Hardware Requirements

The following table shows hardware requirements for both Basic and Corporate **CloverDX Server** edition and for running **CloverDX Server** in a cluster.

Table 6.1. Hardware requirements of CloverDX Server

	Basic Edition	**Corporate Edition**	**Cluster**
RAM	4 GB (recommended 16 GB)	8 GB (recommended 64 GB)	8 GB (recommended 64 GB)
Processors	up to 4 cores	16 cores	8 cores [a]
Disk space (installation)	1 GB	1 GB	1 GB
Disk space (tempspace)	> 25 GB [b]	> 25 GB [b]	> 25 GB [b]
Disk space (data)	> 50 GB [b]	> 50 GB [b]	> 50 GB [b]
Disk space (shared) [c]	-	-	> 50 GB [b]

[a]This may vary depending on total number of nodes and cores in license.
[b]Minimum value, the disk space depends on data.
[c]Disk space for shared sandboxes is required only for **CloverDX Cluster**.

Software Requirements

Operating system

CloverDX Server is compatible with Windows and Unix-based systems, as well as with other systems supporting Java (Mac OS X, IBM System, etc.).

Java Virtual Machine

- Oracle JDK 8 or 9 64 bit
- IBM SDK 8 (for IBM WebSphere only)

Since it is a Java application, it requires Java Development Kit (JDK) to run. We **do not recommend** using Java Runtime Environment (JRE) only, since compilation of some CloverDX Server's transformations requires JDK to function properly.

Application Servers Compatible with CloverDX 5.0

- Apache Tomcat 8.5 or 9 (p. 17)
- IBM WebSphere 9 (p. 21)
- JBoss EAP 7 (p. 25)
- Oracle WebLogic Server 12cR2 (12.2.1) 32/64 bit (p. 29)

Table 6.2. Compatibility Matrix

Application Server	CloverETL 4.1 to 4.7		CloverETL 4.8 to 4.9		CloverDX 5.0 and newer	
	Java 7	Java 8	Java 7	Java 8	Java 8	Java 9
Tomcat 6	✔	✔	✘	✘	✘	✘
Tomcat 7	✔	✔	✔	✔	✘	✘
Tomcat 8	✔	✔	✔	✔	✘	✘
Tomcat 8.5	✘	✘	✔[a]	✔[a]	✔	✔
Tomcat 9	✘	✘	✘	✔	✔	✔
Pivotal tc Server Standard (3.1.9, Tomcat 8)	✘	✔	✘	✔	✘	✘
Pivotal tc Server Standard (3.2.9, Tomcat 8.5)	✘	✘	✘	✔[a]	✔	✔
Jetty 9	✔	✔	✔	✔	✘	✘
WebLogic Server 11gR1 (10.3.6)	✔	✘	4.8 ✔ / 4.9 ✘	✘	✘	✘
WebLogic Server 12cR1 (12.1.2)	✔	✘	✔	✘	✘	✘
WebLogic Server 12cR1 (12.1.3)	✔	✔	✔	✔	✘	✘
WebLogic Server 12cR2 (12.2.1)	✘	✘	✘	✘	✔	✘
JBoss AS 6	✔	✘	✔	✘	✘	✘
JBoss AS 7	✔[b]	✔[c]	✔[b]	✔[c]	✘	✘
JBoss EAP 7	✘	✘	✘	✘	✔	✘
Glassfish 3	✔	✘	✔	✘	✘	✘
WebSphere 8.5	✔	✘	✔	✘	✘	✘
WebSphere 9	✘	✘	✘	✘	✔	✘

[a]Since 4.8.2
[b]EAP 6.2
[c]EAP 6.4

Note

We support Java 8 and 9 on particular supported application server only if the application server itself officially supports it.

Database servers

We support the following database servers. The officially supported versions, we are testing against, are in parentheses.

- MySQL (5.6.12) (p. 66)
- DB2 (10.5.1) (p. 67)
- Oracle (11.2.0.2.0) (p. 70)
- MS SQL Server 2014 (12.0.5557.0) (p. 71)
- PostgreSQL (9.2.4) (p. 73)

The evaluation version uses an embedded Apache Derby database for evaluation purposes. We **do not support** using the Derby database in production environment. Instead you can choose one of several supported database servers.

Chapter 7. Installing

This chapter describes two different Server installations - Evaluation Server (p. 13) and Production Server (p. 15) - and provide instructions on installing the **CloverDX Server** License.

Evaluation Server

The Evaluation Server (p. 13) consists of **CloverDX Server** bundled with the Tomcat application container. The Server performs basic configuration during the first startup and requires no additional database server. This option is **recommended only for basic evaluation** of **CloverDX Server**'s functions.

However with further configuration, it is possible to evaluate other CloverDX Server features and even make the Evaluation Server **ready for production environment**. This process requires a connection to an external, dedicated database and subsequent configuration of services (e.g. SMTP, LDAP, etc.).

> ## Important
>
> The Apache Derby DB, bundled with the Evaluation Server, is **not** supported for production environment. Please use one of the supported database systems.

Production Server

In case of Production Server (p. 15), the **CloverDX Server** is installed on one of the several compatible application containers. This process requires additional configuration (e.g. memory allocation, database connection, etc.) but allows you to choose an application container and external database according to your preference.

Installation and Configuration Procedure

To create a fully working instance of Production **CloverDX Server**, you should:

- **Install an application server**

  CloverDX Server is compatible with several application containers. Following subsections offer detailed instructions on installation of the respective application servers and their subsequent configuration.
- **Set up limits on a number of opened files, memory allocation and firewall exceptions**

  CloverDX Server's graph transformations and evaluations may require more memory than the default limit set in the database as well as higher number of simultaneously opened files. Moreover, some components require firewall exceptions to be set. These instructions provide recommendation on adjusting both the Memory Settings (p. 37) and the Maximum Number of Open Files (p. 39) as well as Firewall Exceptions (p. 39).
- **Install CloverDX Server into the application server**

  CloverDX Server is provided as a web archive (`.war`) file for an easy deployment.
- **Create a database dedicated to CloverDX Server**

  Unlike the Evaluation Server, the Production Server requires that you have created a dedicated database for **CloverDX Server**. In the configuration phase of this manual, you will be guided to Chapter 14, System Database Configuration (p. 63) with instructions on how to properly configure the properties file of various databases.
- **Set up a connection to the database**

  You can set up the connection to the database via Server's GUI in **Configuration →Setup →Database**. Alternatively, you can set up the connection in the properties file before starting **CloverDX Server**.
- **Install a license**

  To be able to execute graphs, you need to install a valid license, see Activation (p. 32).
- **Perform additional Server configuration**

- **Set up a master password for secure parameters**

  When handling sensitive information (e.g. passwords), it is advised to define secure graph parameters. This action requires a master password (see Chapter 20, <u>Secure Parameters</u> (p. 119)).

- **Set up SMTP server connection**

  CloverDX Server lets you configure an SMTP (p. 59) connection for reporting events on the Server via emails.

- **Configure temp space**

  CloverDX Server works with temporary directories and files. To ensure components work correctly, you should configure the `Temp space` location on the file system. For details, see Chapter 19, <u>Temp Space Management</u> (p. 116).

- **Configure sandboxes**

  Lastly, you should set the content security and user's permissions for sandboxes. For details and instructions, see Chapter 22, <u>Sandboxes - Server Side Job Files</u> (p. 144).

Evaluation Server

The default installation of **CloverDX Server** uses embedded Apache Derby DB; therefore, it does not require any external database server or subsequent configuration, as **CloverDX Server** configures itself during the first startup. Database tables and some necessary records are automatically created on the first startup with an empty database.

By performing a subsequent configuration, you can evaluate other **CloverDX Server** features (e.g. sending emails (p. 172,) LDAP authentication (p. 124,) clustering (p. 276,) etc.). This way, you can also prepare the Evaluation Server for production environment. However, note that the embedded Apache Derby database is **not** supported for production environment. Therefore, before the subsequent configuration, choose one of the supported external dedicated databases.

If the **CloverDX Server** must be evaluated on application containers other than Tomcat, or you prefer a different database system, proceed with a common installation of

> ### Note
>
> Default login credentials for **CloverDX Server** Console are:
>
> **Username:** clover
>
> **Password:** clover

Installation

1. Make sure you have a compatible Java version:

   Oracle JDK or JRE v. **1.8.x** or **higher** is required. We recommend JDK 1.8.x.

2. Download and extract the **CloverDX** Evaluation Server.

   2.a Log into your CloverDX account and download the Evaluation Server Bundle.

   2.b Extract the `.zip` archive. The name of the file is `CloverDX.<version>.Tomcat-<version>.zip`.

   > ### Note
   >
   > It is recommended to place the extracted content on a path that does not contain space character(s).
   >
   > `C:\Program Files` or `/home/user/some dir` ✖
   >
   > `C:\Users\Username` or `/home/user/some_dir` ✔

3. Set the `JAVA_HOME` or `JRE_HOME` Environment Variables:

   - **Unix-like systems:**

     - Open the `/bin/setenv.sh` file and define the path at the beginning of the file:

```
export JAVA_HOME=/opt/jdk1.8.0_121
```

```
export JAVA_HOME=/opt/jdk1.8.0_121
if [ -n "$JRE_HOME" -a -n "$JAVA_HOME" ]; then
        echo "Using JAVA_HOME instead of JRE_HOME"
        unset JRE_HOME
fi
```

Figure 7.1. `setenv.sh` edited in Linux.

- **Windows system:**

  - Open the /bin/setenv.bat file and define the path at the beginning of the file:

```
set "JAVA_HOME=C:\java\jdk1.8.0"
```

```
1  @echo off
2  set "JAVA_HOME=C:\java\jdk1.8.0"
3  IF DEFINED JAVA_HOME IF DEFINED JRE_HOME (
4      echo "Using JAVA_HOME instead of JRE_HOME"
5      set JRE_HOME=%JAVA_HOME%
```

Figure 7.2. `setenv.bat` *edited in Windows.*

4. Run Tomcat.

   - **Unix-like systems:** run /bin/startup.sh.

   - **Windows system:** run \bin\startup.bat.

5. Log in **CloverDX Server**.

   5.a Type http://localhost:8083/clover/ in your browser.

   5.b Activate (p. 32) the **CloverDX Server**.

   5.c Use the **default administrator credentials** to access the web GUI:

      Username: clover

      Password: clover

6. **CloverDX Server** is now installed and prepared for basic evaluation. There are couple of sandboxes with various demo transformations installed.

> **Note**
>
> To safely stop the server, run /bin/shutdown.sh or \bin\shutdown.bat on Unix-like or Windows system respectively.

Production Server

This section describes in detail the installation of **CloverDX Server** on various application containers and its subsequent configuration required for production environment. For simple evaluation of **CloverDX Server** features, use Evaluation Server (p. 13) (note that CloverDX Evaluation Server can also be configured for production use).

CloverDX Server for production environment is shipped as a *Web application archive* (WAR file) and uses an external, dedicated database, so standard methods for deploying a web application on your application server may be used. However, each application server has specific behavior and features. Detailed information about their installation and configuration can be found in the following chapters.

List of Suitable Containers

- Apache Tomcat (p. 17)
- IBM WebSphere (p. 21)
- JBoss Enterprise Application Platform (p. 25)
- Oracle WebLogic Server (p. 29)

In case of problems during the installation see Chapter 10, Troubleshooting (p. 42).

> ### Important
>
> Oracle JDK or JRE v. **1.8.x** or **1.9.x** is required.

Installation and Configuration Procedure

To create a fully working instance of Production CloverDX Server, you should:

1. **Install an application server**

   CloverDX Server is compatible with several application containers. Following subsections offer detailed instructions on installation of the respective application servers and their subsequent configuration.

2. **Set up limits on a number of opened files, memory allocation and firewall exceptions**

   CloverDX Server's graph transformations and evaluations may require more memory than the default limit set in the database as well as higher number of simultaneously opened files. Moreover, some components require firewall exceptions to be set. These instructions provide recommendation on adjusting both the Memory Settings (p. 37) and the Maximum Number of Open Files (p. 39) as well as Firewall Exceptions (p. 39).

3. **Install CloverDX Server into the application server**

   CloverDX Server is provided as a web archive (`.war`) file for an easy deployment.

4. **Create a database dedicated to CloverDX Server**

   Unlike the Evaluation Server, the Production Server requires that you have created a dedicated database for **CloverDX Server**. In the configuration phase of this manual, you will be guided to Chapter 14, System Database Configuration (p. 63) with instructions on how to properly configure the properties file of various databases.

5. **Set up a connection to the database**

   You can set up the connection to the database via Server's GUI in **Configuration →Setup →Database**. Alternatively, you can set up the connection in the properties file before starting **CloverDX Server**.

6. **Install a license**

To be able to execute graphs, you need to install a valid license. There are three options for [Activation](p. 32).

7. **Perform additional Server configuration**

   • **Set up a master password for secure parameters**

   When handling sensitive information (e.g. passwords), it is advised to define secure graph parameters. This action requires a master password (see Chapter 20, [Secure Parameters](p. 119)).

   • **Set up SMTP server connection**

   CloverDX Server lets you configure an SMTP (p. 59) connection for reporting events on the Server via emails.

   • **Configure temp space**

   CloverDX Server works with temporary directories and files. To ensure components work correctly, you should configure the `Temp space` location on the file system. For details, see Chapter 19, [Temp Space Management](p. 116).

   • **Configure sandboxes**

   Lastly, you should set the content security and user's permissions for sandboxes. For details and instructions, see Chapter 22, [Sandboxes - Server Side Job Files](p. 144).

Apache Tomcat

> ### Important
>
> Before installation, check the software requirements, currently supported Apache Tomcat version and required Java version in the Software Requirements (p. 9) section.
>
> If you encounter any problems during the installation, see Chapter 10, Troubleshooting (p. 42) for a possible solution.

Installation of Apache Tomcat

1. Download the binary distribution: Tomcat 8.5 or Tomcat 9.

2. Extract the downloaded archive (`zip` or `tar.gz`).

3. Set up `JAVA_HOME` to point to the correct Java version:

   - **Unix-like systems:**

     Setup the path in `/etc/profile` or `/etc/bash.bashrc`:

     export `JAVA_HOME=/path/to/JDK`

   - **Windows system:**

     Under **System Variables** in **Advanced System Settings**, create a new variable named `JAVA_HOME`. The value should contain the path to the JDK installation directory.

4. Run Tomcat:

   - **Unix-like systems:**

     Run the `[Tomcat_home]/bin/startup.sh` file.

   - **Windows system:**

     Run the `[Tomcat_home]\bin\startup.bat` file.

5. Check whether Tomcat is running.

   - Open a new tab in your browser and type http://localhost:8080/ in the address bar.

Continue with: Installation of CloverDX Server (p. 19).

Apache Tomcat as a Windows Service

1. Download the **32-bit/64-bit Windows Service Installer** file in the **Binary Distributions** section on the Tomcat 8.5 or Tomcat 9 download page.

2. Use the standard installation wizard to install Apache Tomcat.

3. When Tomcat is installed as a Windows service, **CloverDX** is configured by one of the following options:

a. **Graphical configuration utility**

- Run the `[Tomcat_home]\bin\Tomcat9w.exe` file.

- In the **Apache Tomcat Properties** dialog box, select the **Java** tab and set the initial and maximum heap size in **Initial memory pool** and **Maximum memory pool** fields to 512MB and 1024MB respectively. Other configuration parameters can be defined in **Java Options** field, being separated by new line.

- Click on **Apply** and restart the service.

> **Note**
>
> The **Java** tab allows you to use alternative Java virtual machine by setup of path to `jvm.dll` file.

b. **Command Prompt tool**

- Run the `[Tomcat_home]\bin\Tomcat9.exe` file.

- If Tomcat is running, navigate to `[Tomcat_home]\bin` and stop the service by typing:

    ```
    .\Tomcat9.exe //SS//Tomcat9
    ```

    in the Command Prompt. (When using different version of Tomcat, change the number in the command to reflect the installed version.)

- Configure the service by typing the command:

    ```
    .\Tomcat9.exe          //US//Tomcat9          --JvmMs=512          --
    JvmMx=1024      --JvmOptions=-Dclover.config.file=C:\path\to\clover-
    config.properties#-XX:MaxMetaspaceSize=256m
    ```

    The parameter JvmMs is the initial and JvmMx is the maximum heap size in MB; JvmOptions are separated by '#' or ';'.

- Start the service from Windows administration console or by typing the following command in the Command Prompt:

    ```
    .\Tomcat9.exe //TS//Tomcat9
    ```

> **Tip**
>
> By default, when Apache Tomcat is run as a Windows service, it is **not available** for Java process monitoring tools (e.g., **JConsole** or **JVisualVM**). However, these tools can still connect to the process via **JMX**. In order to expose Tomcat's Java process via JMX, add the following options to the service settings:
>
> ```
> -Dcom.sun.management.jmxremote.port=3333
> -Dcom.sun.management.jmxremote.ssl=false
> -Dcom.sun.management.jmxremote.authenticate=false
> ```
>
> Once the service is run with these options, you can connect to **port 3333** using JMX and monitor the server.

More information about running Java applications as Windows Service can be found at [Apache Commons](#).

Continue with:

Apache Tomcat on IBM AS/400 (iSeries)

Additional settings are required to run **CloverDX Server** on the iSeries platform:

1. Declare you are using Java 8.0 32-bit.

2. Run Java with parameter `-Djava.awt.headless=true`.

To configure the settings, modify (or create) the `[Tomcat_home]/bin/setenv.sh` file to contain:

```
JAVA_HOME=/QOpenSys/QIBM/ProdData/JavaVM/jdk70/32bit
JAVA_OPTS="$JAVA_OPTS -Djava.awt.headless=true"
```

Continue with:

Installation of CloverDX Server

1. Check if you meet the prerequisites:

    • Oracle JDK or JRE is installed (See Java Virtual Machine (p. 9) for required Java version.)

    • `JAVA_HOME` or `JRE_HOME` environment variable is set (see Setting up JAVA_HOME (p. 17)).

    • A supported version (p. 10) of Apache Tomcat is installed.

2. It is strongly recommended to adjust the default limits for **Memory allocation** (see the Memory Settings (p. 37) section).

    You can set the **initial** and **maximum memory heap size** by adjusting the **Xms** and **Xmx** JVM parameters:

    Unix-like systems:

    • Create the `[Tomcat_home]/bin/setenv.sh` file.

    • Type or paste in the following lines:

```
export CATALINA_OPTS="$CATALINA_OPTS -XX:MaxMetaspaceSize=512m -Xms128m -Xmx1024m"
export CATALINA_OPTS="$CATALINA_OPTS -Dderby.system.home=$CATALINA_HOME/temp -server"
echo "Using CATALINA_OPTS: $CATALINA_OPTS"
```

    Windows systems:

    • Create the `[Tomcat_home]\bin\setenv.bat` file.

    • Type or paste in the following lines:

```
set "CATALINA_OPTS=%CATALINA_OPTS% -XX:MaxMetaspaceSize=512m -Xms128m -Xmx1024m"
set "CATALINA_OPTS=%CATALINA_OPTS% -Dderby.system.home=%CATALINA_HOME%/temp -server"
echo "Using CATALINA_OPTS: %CATALINA_OPTS%"
```

Tip

For performance reasons, it is recommended to run the container in the "server mode" by setting the `-server` switch, as seen in the settings above. [1]

Note that on a 64-bit capable JDK, only the Java Hotspot Server VM is supported so the `-server` option is implicit.

3. Go to the download section of your CloverDX account and download the `clover.war` (web archive) file containing **CloverDX Server** for Apache Tomcat.

4. Copy `clover.war` to the `[Tomcat_home]/webapps` directory.

5. Tomcat should automatically detect and deploy the `clover.war` file.

6. Check whether **CloverDX Server** is running:

   • Run Tomcat.

   • Open a new tab in your browser and type http://localhost:8080/clover/ in the address bar.

   • Use the **default administrator credentials** to access the web GUI: username: **clover**, password: **clover**.

Continue with:

Configuration of CloverDX Server on Apache Tomcat

Tip

Default installation (without any configuration) is only recommended for evaluation purposes. For production use, at least a dedicated, system database and SMTP server configuration is recommended.

For easy configuration of **CloverDX Server**, use Setup GUI (p. 54) in which you can configure various properties, including the connection to the database, username and password, path to the license file, private properties, clusters and much more (see Chapter 15, List of Configuration Properties (p. 82) and Chapter 42, Cluster Configuration (p. 280). We recommend you place the file in a specified (p. 50) location and define the path to the file with a system property.

The content of such a file (an example with a PostgreSQL database):

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://127.0.0.1/clover_db?charSet=UTF-8
jdbc.username=yourUsername
jdbc.password=yourPassword
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Properties File in Specified Location

The properties file is loaded from a location specified by a system property or by an environment variable `clover_config_file` or `clover.config.file`.

1. Create the `cloverServer.properties` file in a directory readable by Apache Tomcat. (If you need an example of connection to any of the supported database systems, see Chapter 14, System Database Configuration (p. 63).)

2. Edit the `[Tomcat_home]/bin/setenv.sh` file (if it does not exist, you can create it).

3. Set the system property by adding the following line into the file:

```
JAVA_OPTS="$JAVA_OPTS -Dclover_config_file=/path/to/cloverServer.properties"
```

Note

Continue with: Chapter 8, Postinstallation Configuration (p. 37)

IBM WebSphere

Important

Before installation, check the software requirements, currently supported IBM WebSphere version and required Java version in the Software Requirements (p. 9) section.

If you encounter any problems during the installation, see Chapter 10, Troubleshooting (p. 42) for a possible solution.

Installation of CloverDX Server on IBM WebSphere

1. Download and install IBM WebSphere from the official download page.

   Note: During installation, make sure the IBM WebSphere profile name does not contain the keyword **clover**, otherwise the **CloverDX Server** won't start properly.

2. It is strongly recommended to adjust the default limits for **Memory allocation** (see the Memory Settings (p. 37) section).

   You can set the limits in IBM WebSphere's **Integrated Solutions Console** (default URL: http://localhost:9060/ibm/console/).

   a. Go to **Servers →Server Types →WebSphere application servers →[Server_Name] (default name: server1) →Java and Process Management →Process definition →Java Virtual Machine**

   b. Change the value in the **Maximum heap size** field to 2048 MB. The default value (256 MB) is insufficient for ETL transformations.
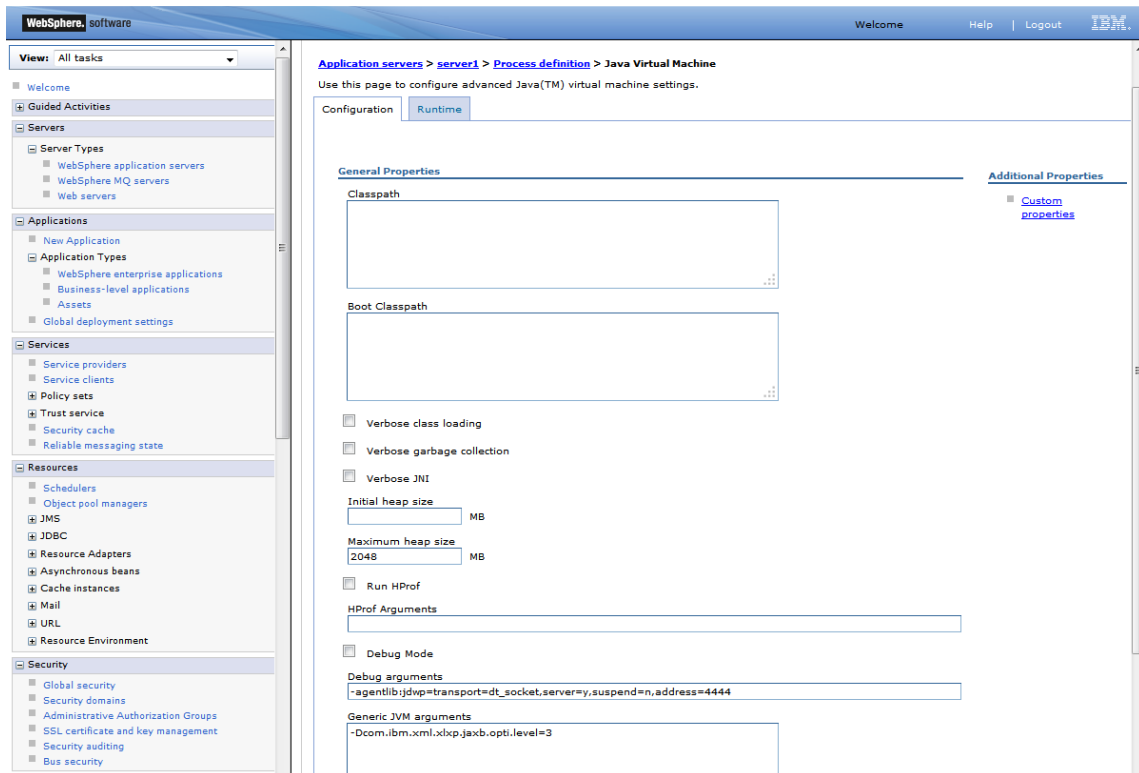
Figure 7.3. Adjusting Maximum heap size limit

c. Add the following parameters into the **Generic JVM arguments** field to set the perm space limit and direct memory limit:

    ```
    -XX:MaxPermSize=512M
    ```

    ```
    -XX:MaxDirectMemorySize=512M
    ```

d. Java runtime settings:

    Go to **Servers →Server Types →WebSphere application servers →[Server_Name] (default name: server1) →Java SDKs** and select version 1.8 as the default SDK.

e. Save the changes to configuration and restart the server so that they take effect.

3. Go to the download section of your CloverDX account and download the `clover.war` (web archive) file containing **CloverDX Server** for WebSphere.

4. Deploy the `clover.war` file.

    a. Go to **Integrated Solutions Console** (default URL: http://localhost:9060/ibm/console/).

    b. Go to **Applications →New Application →New Enterprise Application**, select the **CloverDX Server** WAR archive and deploy it to the application server, but do not start it yet.

5. Configure application class loading.

    Go to **WebSphere Enterprise Applications →clover_war (or other name of the Clover application) →Manage Modules →CloverDX** and under **Class loader order** select **Classes loaded with local class loader first (parent last)**.

6. Save the changes to the Server configuration and start the **clover_war** application.

7. Log in **CloverDX Server**.

    a. Provided you set `clover.war` as the application running with "clover" context path, use the following URL (notice the port number has changed):

       http://localhost:9080/clover

    b. Activate (p. 32) the **CloverDX Server**.

    c. Use the **default administrator credentials** to access the web GUI:

       Username: clover

       Password: clover

> ## Note
>
> Please note that some **CloverDX** features using third party libraries do not work properly on IBM WebSphere.
>
> - Hadoop is guaranteed to run only on Oracle Java 1.6+, but Hadoop developers do make an effort to remove any Oracle/Sun-specific code. See Hadoop Java Versions on Hadoop Wiki.
>
> - **AddressDoctor5** on IBM WebSphere requires additional JVM parameter `-Xmso2048k` to prevent AddressDoctor from crashing JVM. See documentation on AddressDoctor component.
>
>   Note that the parameter must be set for Worker, as well. Use the worker.jvmOptions (p. 91) property.

Configuration of CloverDX Server on IBM WebSphere

> ## Tip
>
> Default installation (without any configuration) is only recommended for evaluation purposes. For production use, at least a dedicated, system database and SMTP server configuration is recommended.

For easy configuration of **CloverDX Server**, use a Setup GUI (p. 54) in which you can configure various properties, including the connection to the database, username and password, path to the license file, private properties, clusters and much more (see Chapter 15, List of Configuration Properties (p. 82) and Chapter 42, Cluster Configuration (p. 280). We recommend you place the file in a specified (p. 50) location and specify the path to the file with a system property.

The content of such a file (an example with a PostgreSQL database):

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://127.0.0.1/clover_db?charSet=UTF-8
jdbc.username=yourUsername
jdbc.password=yourPassword
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Properties File in Specified Location

The properties file is loaded from a location which is specified by the environment/system property `clover_config_file` or `clover.config.file`.

1. Create the `cloverServer.properties` file in a directory readable by IBM WebSphere. (If you need an example of connection to any of the supported database systems, see Chapter 14, System Database Configuration (p. 63).)

2. Set a system property (or environment variable) `clover_config_file` pointing to the properties file.

a. Go to **Integrated Solutions Console** (default URL:http://localhost:9060/ibm/console/).

b. Go to **Servers →WebSphere application servers →[Server_name] →Java and Process Management →Process Definition →Java Virtual Machine →Custom Properties**.

c. Create a system property named `clover_config_file` whose value is a full path to the properties file (e.g. `cloverServer.properties`) on your file system.

3. Restart IBM WebSphere for changes to take effect.

Note

Continue with: Chapter 8, Postinstallation Configuration (p. 37)

JBoss Enterprise Application Platform

> ⚠️ **Important**
>
> Before installation, check the software requirements, currently supported JBoss EAP versions and required Java versions in the **Software Requirements** (p. 9) section.
>
> If you encounter any problems during the installation, see Chapter 10, **Troubleshooting** (p. 42) for a possible solution.

Installation of CloverDX Server on JBoss EAP

1. Download and install JBoss EAP from the **official download page**.

2. It is strongly recommended to adjust the default limits for **Memory allocation** (see the **Memory Settings** (p. 37) section).

   You can set the **initial** and **maximum memory heap size** by adjusting the "Xms" and "Xmx" JVM parameters and **classloaders memory limit** by adjusting the "XX:MaxMetaspaceSize" parameter:

   For JBoss EAP standalone mode, follow these steps:

   - **Unix-like systems:**

     - Edit the `[JBoss_EAP_home]/bin/standalone.conf` file.

     - Add the following line:

   ```
   JAVA_OPTS="$JAVA_OPTS -XX:MaxMetaspaceSize=512m -Xms128m -Xmx2048m"
   ```

   - **Windows systems:**

     - Edit the `[JBoss_EAP_home]\bin\standalone.conf.bat` file.

   ```
   JAVA_OPTS="$JAVA_OPTS -XX:MaxMetaspaceSize=512m -Xms128m -Xmx2048m"
   ```

3. Go to the download section of your CloverDX account and download the `clover.war` (web archive) file containing **CloverDX Server** for JBoss EAP.

4. Configure the database connection.

   By default, **CloverDX Server** uses an embedded Derby database; however, such setup is not supported for production use.

   You can use the database connection provided by JNDI-bound datasource deployed by JBoss EAP. In order to define the datasource, edit the file:

   ```
   [JBoss_EAP_home]/standalone/configuration/standalone.xml
   ```

   and add the definition of the datasource into the section `<subsystem xmlns="urn:jboss:domain:datasources:1.1">` under the element `<datasources>`. Here is an example of datasource connecting to a MySQL database:

```
<datasource jndi-name="java:jboss/datasources/CloverDX"
    pool-name="CloverDX-Pool" enabled="true">
    <connection-url>jdbc:mysql://localhost:3307/cloverServerDB</connection-url>
    <driver>com.mysql</driver>
    <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
    <pool>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>50</max-pool-size>
        <prefill>true</prefill>
    </pool>
    <security>
        <user-name>clover</user-name>
        <password>SecretPassword</password>
    </security>
    <statement>
        <prepared-statement-cache-size>32</prepared-statement-cache-size>
        <share-prepared-statements>true</share-prepared-statements>
    </statement>
</datasource>
<drivers>
    <driver name="com.mysql" module="mysql.driver">
        <driver-class>com.mysql.jdbc.Driver</driver-class>
    </driver>
</drivers>
```

5. The datasource definition references a module (`mysql.driver`) with the MySQL JDBC driver. Take the following steps to add the module:

> ### Note
>
> Under JBoss EAP, there are more options to set up **CloverDX Server**'s database: along with JNDI-bound data source, it is possible to use the embedded Derby database or other supported database system specified in the **CloverDX** configuration file.

In order to be able to connect to the database, you need to define a global module so that the driver is available for the **CloverDX** web application - copying the driver to the `lib/ext` directory of the Server will **not** work. Such module is created and deployed in few steps (the example is for MySQL and module's name is `mysql.driver`):

a. Create a directory `[JBoss_EAP_home]/modules/mysql/driver/main` (note that the directory path corresponds to module name `mysql.driver`)

b. Copy the driver `mysql-connector-java-5.1.5-bin.jar` to the directory and create a file `module.xml` there with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="mysql.driver">
    <resources>
        <resource-root path="mysql-connector-java-5.1.5-bin.jar" />
    </resources>
    <dependencies>
        <module name="javax.api" />
    </dependencies>
</module>
```

c. Add the module to global server modules: in case of the standalone JBoss EAP server they are defined in `[JBoss_EAP_home]/standalone/configuration/standalone.xml`. Add the module to the EE domain subsystem section:

```
<subsystem xmlns="urn:jboss:domain:ee:1.1">
    <global-modules>
        <module name="mysql.driver" slot="main" />
    </global-modules>
    <spec-descriptor-property-replacement>false</spec-descriptor-property-replacement>
```

```
    <jboss-descriptor-property-replacement>true</jboss-descriptor-property-replacement>
</subsystem>
```

6. Configure **CloverDX Server** according to a description in the .

7. Deploy WAR file.

   Copy the `clover.war` file to `[JBoss_EAP_home]/standalone/deployments`.

8. To start the JBoss platform:

   • **Unix-like systems:**

     Run `[JBoss_EAP_home]/bin/standalone.sh`.

   • **Windows system:**

     Run `[JBoss_EAP_home]\bin\standalone.bat`.

   It may take a couple of minutes for all applications to start.

9. Log in **CloverDX Server**.

   a. Type http://localhost:8080/clover in the browser.

   b. Activate (p. 32) the **CloverDX Server**.

   c. Use the **default administrator credentials** to access the web GUI:

      Username: clover

      Password: clover

Configuration of CloverDX Server on JBoss EAP

> **Tip**
>
> The default installation (without any configuration) is only recommended for evaluation purposes. For production use, at least a dedicated, system database and SMTP server configuration is recommended.

For an easy configuration of **CloverDX Server**, use a Setup GUI (p. 54) in which you can configure various properties, including the connection to the database, username and password, path to the license file, private properties, clusters and much more (see Chapter 15, List of Configuration Properties(p. 82) and Chapter 42, Cluster Configuration (p. 280). We recommend you place the file in a specified (p. 50)location and specify the path to the file with a system property.

Properties File in Specified Location

The properties file is loaded from a location which is specified by the environment/system property `clover_config_file` or `clover.config.file`.

1. • Create the `cloverServer.properties` file in a directory readable by JBoss EAP. (If you need an example of connection to any of the supported database systems, see Chapter 14, System Database Configuration (p. 63).):

```
datasource.type=JNDI
datasource.jndiName=java:jboss/datasources/CloverDXServerDS
jdbc.dialect=org.hibernate.dialect.MySQLDialect
license.file=/home/clover/config/license.dat
```

Do not forget to set a correct JDBC dialect according to your database server (Part III, "Configuration" (p. 47)). You can set the path to the license file, too.

- Alternatively, you can set "JDBC" `datasource.type` and configure the database connection to be managed directly by **CloverDX Server** (provided that you have deployed proper JDBC driver module to the Server):

```
datasource.type=JDBC
jdbc.url=jdbc:mysql://localhost:3306/cloverServerDB
jdbc.dialect=org.hibernate.dialect.MySQLDialect
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.username=clover
jdbc.password=SecretPassword
license.file=/home/clover/config/license.dat
```

2. Set the `clover.config.file` system property (or environment variable).

   It should contain the full path to the `cloverServer.properties` file created in the previous step.

   The simplest way to set the system property is to edit the configuration file `[JBoss_EAP_home]/standalone/configuration/standalone.xml`, and to add the following snippet just under the `<extensions>` section:

```
<system-properties>
    <property name="clover.config.file" value="C:/jboss-eap-6.2/cloverServer.properties" />
</system-properties>
```

3. Restart JBoss EAP for the changes to take effect.

4. Check the **CloverDX Server** application is running:

   By default, the Server's console is accessible at http://localhost:8080/clover.

   ## Note

   By default, JBoss EAP has enabled HTTP session replication. This requires session serialization that is not supported by **CloverDX Server** and produces lots of harmless errors in JBoss's console.

   To eliminate these errors, disable the session replication. Edit `[jboss-home]/standalone/configuration/standalone.xml` and comment out the whole `<cache-container name="web" aliases="standard-session-cache">` block under the `<subsystem xmlns="urn:jboss:domain:infinispan:1.5">` section.

   ## Note

   Continue with: Chapter 8, Postinstallation Configuration (p. 37)

Oracle WebLogic Server

> **Important**
>
> Before installation, check the software requirements, currently supported Oracle WebLogic versions and required Java version in the Software Requirements (p. 9) section.
>
> If you encounter any problems during the installation, see Chapter 10, Troubleshooting (p. 42) for a possible solution.

Installation of CloverDX Server on Oracle WebLogic

1. Download and install Oracle WebLogic from the official download page.

   WebLogic has to be running and a domain has to be configured. You can check it by connecting to **Administration Console**: http://localhost:7001/console/. **Username** and **password** are specified during installation.

2. It is strongly recommended to adjust the default limits for **Memory allocation** (see the Memory Settings (p. 37) section).

   You can set the **initial** and **maximum memory heap size** by adjusting the "Xms" and "Xmx" JVM parameters and **classloaders memory limit** by adjusting the "XX:MaxMetaspaceSize" parameter:

   - **Unix-like systems:**

     Edit the start script and add:

     ```
     export JAVA_OPTIONS='$JAVA_OPTIONS -Xms512m -Xmx2048m -XX:MaxMetaspaceSize=512m'
     ```

   - **Windows system:**

     See WebLogic Server Performance and Tuning.

3. Go to the download section of your CloverDX account and download the `clover.war` (web archive) file containing **CloverDX Server** for Oracle WebLogic Server.

4. Change HTTP Basic Authentication configuration

   - When WebLogic finds an "Authentication" header in an HTTP request, it tries to find a user in its own realm. This behavior has to be disabled so **CloverDX** could authenticate users itself.

   - Edit the configuration file `[domainHome]/config/config.xml` and add:

     ```
     <enforce-valid-basic-auth-credentials>false</enforce-valid-basic-auth-credentials>
     ```

     into `<security-configuration>` element (just before the end tag).

5. Deploy `clover.war` (or an application directory).

   Use the **WebLogic Server Administration Console**. See the Oracle Fusion Middleware Administrator's Guide for details.

6. Configure a license and other properties. See Configuration of CloverDX Server on Oracle WebLogic (p. 30) for details.

7. Log in **CloverDX Server**.

    a. Type http://localhost:7001/clover in the browser.

    b. Activate (p. 32) the **CloverDX Server**.

    c. Use the **default administrator credentials** to access the web GUI:

        Username: clover

        Password: clover

Configuration of CloverDX Server on Oracle WebLogic

Tip

The default installation (without any configuration) is only recommended for evaluation purposes. For production use, at least a dedicated, system database and SMTP server configuration is recommended.

For an easy configuration of **CloverDX Server**, use a Setup GUI (p. 54) in which you can configure various properties, including the connection to the database, username and password, path to the license file, private properties, clusters and much more (see Chapter 15, List of Configuration Properties (p. 82) and Chapter 42, Cluster Configuration (p. 280). We recommend you place the file in a specified (p. 50) location and specify the path to the file with a system property.

The content of such a file (example with a PostgreSQL database):

```
datasource.type=JDBC
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://127.0.0.1/clover_db?charSet=UTF-8
jdbc.username=yourUsername
jdbc.password=yourPassword
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Properties File in Specified Location

1. Create the `cloverServer.properties` file in a directory readable by WebLogic. (If you need an example of connection to any of the supported database systems, see Chapter 14, System Database Configuration (p. 63).)

   The configuration file should contain DB datasource configuration, SMTP connection configuration, etc. For details, see Part III, "Configuration" (p. 47).

2. Set the `clover_config_file` system property (or environment variable) pointing to the configuration properties file.

   • Set the JAVA_OPTIONS variable in the WebLogic domain start script `[domainHome]/startWebLogic.sh`

```
JAVA_OPTIONS="${JAVA_OPTIONS} -Dclover_config_file=/path/to/clover-config.properties
```

3. Restart WebLogic for changes to take effect.

Note: When **CloverDX Server** is deployed on WebLogic and JNDI Datasource pointing to Oracle DB is used, there must be an extra configuration property in the configuration file:

```
quartz.driverDelegateClass=org.quartz.impl.jdbcjobstore.oracle.weblogic.WebLogicOracleDelegate
```

Note

Continue with: Chapter 8,

Activation

To be able to execute graphs, **CloverDX Server** requires a valid license. You can install and run **CloverDX Server** without any license, but no graph will be executed.

There are three ways of installing the license. They work on all application servers and can be used at the same time, but **only the most recent valid license is used**.

We recommend using the first and easiest option (for other options, see CloverDX Server Activation Alternatives (p. 34)):

CloverDX Server Activation using Web Form

If the **CloverDX Server** has been started without assigning any license, click the **Activate server** link on the welcome page. You will be redirected to the **Add New License** form where you can upload the license file using the **Browse** button, or simply copy the license from the file and paste it into the **License key text** field.
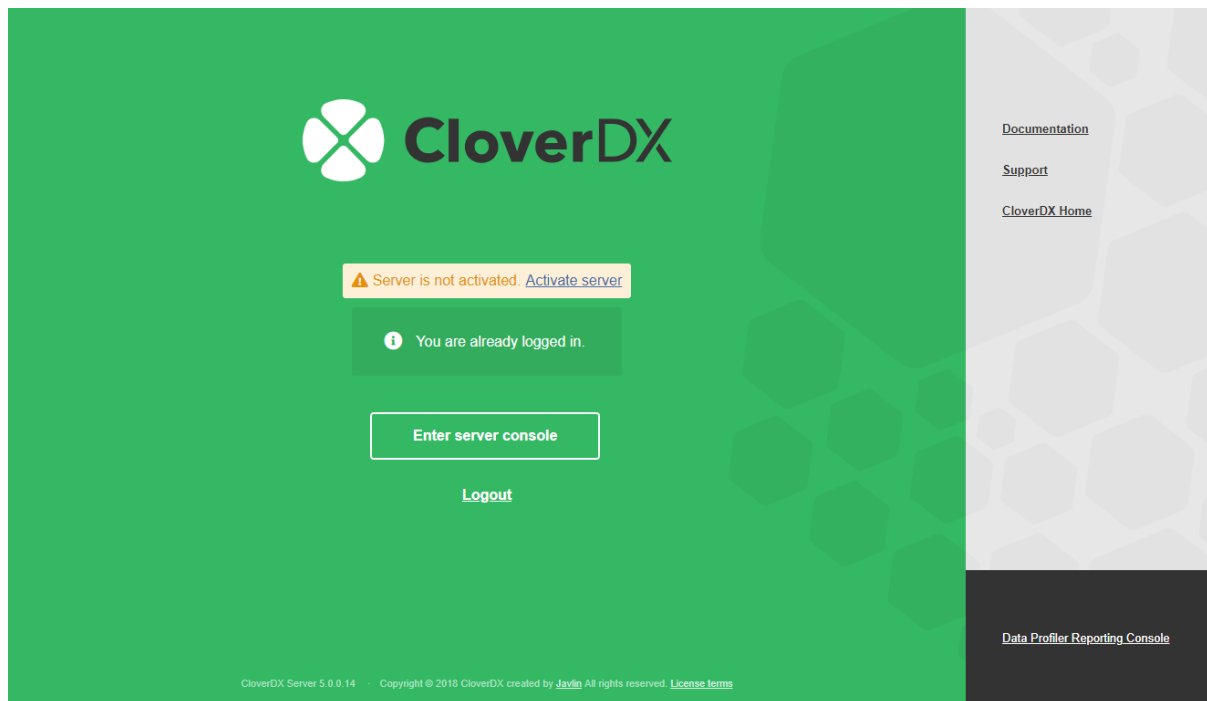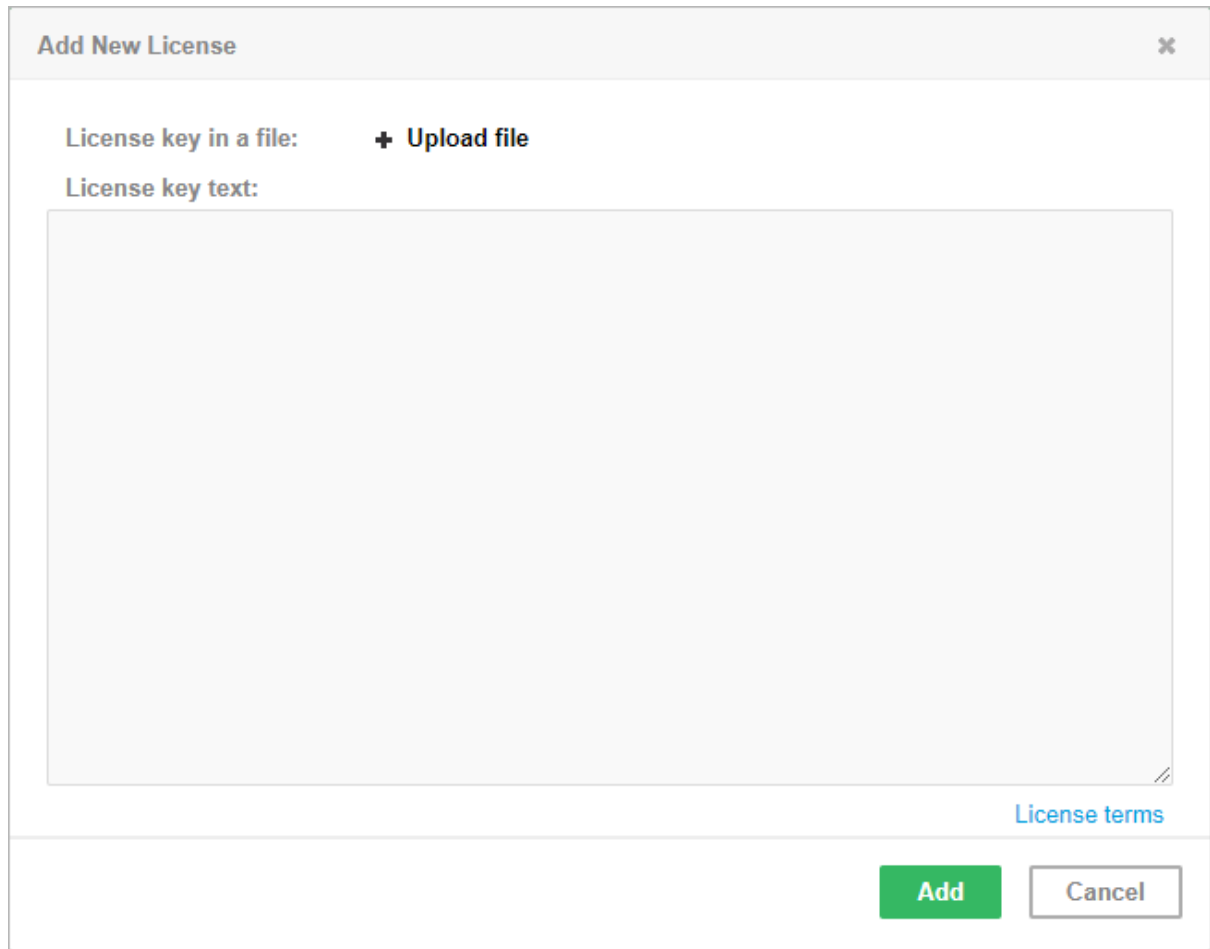

Figure 7.4. Login page of CloverDX Server without license

After clicking the **Update** button, the license is validated and saved to the database table *clover_licenses*. If the license is valid, a table with license's description appears. To proceed to **CloverDX Server** console click **Continue to server console**.

You can skip adding a license by clicking the **Close** button.

Figure 7.5. Add new license form

Updating CloverDX Server License in the Configuration Section

If the license has been already installed, you can still change it by using form in the Server web GUI.

- Go to **server web GUI** →**Configuration** →**Setup** →**License**

- Click **Update license**.

You can paste a license text into a **License key** text area or use the **Browse** button to search for a license file in the filesystem. To skip adding a license, click the **Close** button.

After clicking the **Update** button, the license is saved to the database table *clover_licenses* and reloaded.

Figure 7.6. Update license form

Tip

The **CloverDX** license can be **changed** at any time by replacing `license.dat` file. Afterwards, you have to let **CloverDX Server** know the license has changed.

- Go to **server web GUI** →**Configuration** →**Setup** →**License**

- Click **Reload license**.

- Alternatively, you can restart the **CloverDX Server** application.

Note

The license in the database is common for all nodes in the cluster. Reloading of the license occurs on each node in the cluster.

Continue with:

CloverDX Server Activation Alternatives

If, for any reason, you decide to not use the recommended way of installing the server license, you can choose one of the following options:

Activation Using license.file Property

1. Download the `license.dat` file.

2. Set the **CloverDX Server** `license.file` parameter to the full path of the `license.dat` file.

   For list of properties, see Chapter 15, [List of Configuration Properties](#) (p. 82).

3. Restart the application server.

Separate License WAR

A simple approach, but it may be used only for a standalone Server running on Apache Tomcat.

1. Download the `clover-license.war` web archive file.

2. Copy `clover-license.war` to the `[tomcat_home]/webapps` directory.

3. The war file should be detected and deployed automatically without restarting Tomcat.

4. Check whether the license web-app is running on:

   `http://[host]:[port]/clover-license/` (Note: `clover-license` contextPath is mandatory and cannot by changed)

   > **Note**
   >
   > ➡ **Continue with:**   Chapter 13, [Setup](#) (p. 54)

IBM InfoSphere MDM Plugin Installation

Downloading

IBM InfoSphere MDM Components for **CloverDX Server** are downloaded as a ZIP file containing the extension. The ZIP file is available for download under your account on www.cloverdx.com in **CloverDX Server** download area, under the **Utilities** section as the `ibm-mdm-connectors.${version}.zip` file.

Requirements

Requirements of **IBM InfoSphere MDM Components**:

- supported OS are Microsoft Windows 32 bit, Microsoft Windows 64 bit, Linux 64 bit, and Mac OS X Cocoa

- at least 512MB of RAM

- installed **CloverDX Server**

The support for 32 bit Linux was removed in 4.5.0.

Installation into Server

The following steps are needed to install **IBM InfoSphere MDM Components** into **CloverDX Server**:

1. Install **CloverDX Server**, see its documentation for details.

2. Download the ZIP file with **IBM InfoSphere MDM Components** for the Server and store it on the system where **CloverDX Server** is installed. For the download instructions, see [Downloading](#) (p. 35).

3. The ZIP file contains a **CloverDX** plugin. Your Server installation needs to be configured to find and load the plugin from the ZIP file. This is done by setting the `engine.plugins.additional.src` Server configuration property to the absolute path of the ZIP file, e.g. `engine.plugins.additional.src=c:/Server/ibm-mdm-connectors.${version}.zip` (in case the Server is configured via property file).

   Details for setting the configuration property depend on your Server installation specifics, application server used, etc. See **CloverDX Server** documentation for details. Typically the property would be set similarly to how you set-up the properties for connection to the Server's database. Updating the configuration property usually requires restart of the Server.

4. To verify that the plugin was loaded successfully, log into the Server's **Reporting Console** and look in the **Configuration** →**CloverDX Info** →**Plugins** page. In the list of plugins, you should see `cloverdx.engine.initiate`.

Troubleshooting

If you get an `Unknown component` or `Unknown connection` error when running a graph with IBM InfoSphere MDM components, it means that the **IBM InfoSphere MDM Components** plugin was not loaded by the Server successfully. Please check the above steps to install the plugin, especially the path to the ZIP file.

Chapter 8. Postinstallation Configuration

Memory Settings

Current implementation of Java Virtual Machine allows only a global configuration of memory for the JVM system process. Thus the whole application server, together with WARs and EARs running on it, share one memory space.

Default JVM memory settings is **too low** for running an application container with **CloverDX Server**. Some application servers, like IBM WebSphere, increase JVM defaults themselves, however they **still may be too low**.

The **optimal memory limits** depend on many conditions, i.e. transformations which **CloverDX** should execute. Please note that the maximum limit isn't the amount of permanently allocated memory, but limit which can't be exceeded. If the limit is exhausted, the `OutOfMemoryError` is raised.

JVM Memory Areas

JVM memory consists of several areas: **heap memory**, **PermGen space**, **direct memory** and **stack memory**. Since JVM memory is not just HEAP memory, you should not set the HEAP limit too high; in case it consumes whole RAM, JVM won't be able to allocate direct memory and stack for new threads.

Table 8.1. JVM Memory Structure

Type	Description
Heap memory	Heap is an area of memory used by JVM for dynamic memory allocation. Required heap memory size depends on various factors (e.g. complexity of graphs, number of graphs running in parallel, type of component, etc.), see the respective server container's installation guide in this documentation. (Note that current heap memory usage can be observed in CloverDX Server Console (p. 107).)
PermGen Space	Permanent Generation - separate memory space containing class definitions and related metadata. (PermGen was removed from Java 8.)
Direct Memory	Memory used by graph edges and buffers for I/O operations.
Stack Memory	Stack Memory contains local, method specific variables and references to other objects in the method. Each thread has its own stack; therefore, the memory usage depends on the number of components running in parallel.

Configuring Memory

You can set the minimum and maximum memory heap size by adjusting the "Xms" and "Xmx" JVM parameters. There are more ways to change the settings depending on the used application container.

Recommended Server Core and Worker Heap Memory Configuration

Optimal distribution of main memory between Server Core and Worker depends on nature of executed tasks. The recommended defaults of Server Core heap size and Worker heap size for different RAM sizes are in the table below.

Heap limit is *not* a limit of the full memory used by JVM. JVM uses memory in addition to the heap size for other memory spaces, e.g. direct memory. We recommend to set the heap limit to no more than 80% of system memory size, to leave space for the operating system and other JVM memory spaces.

Table 8.2.

RAM Size	Server Core Heap	Worker Heap	Remaining for OS (estimated)
4 GB	1 GB	1 GB	1 GB
8 GB	2-3 GB	2-3 GB	2 GB
16 GB	4 - 8 GB	4 - 8 GB	4 GB
32 GB	4 - 8 GB	16 - 20 GB	8 GB
64 GB	4 - 8 GB	42 - 52 GB	8 GB

Memory Configuration in Java 8

In Java 8, the memory space for loading classes (so called "Metaspace") is separated from heap, and can be set by the JVM parameter `-XX:MaxMetaspaceSize`. The default maximum Metaspace size is unlimited.

Please see the specific container section for details on memory settings.

Metaspace

We recommend you to put limit on metaspace memory. Add `-XX:MaxMetaspaceSize=size` to command line parameters of Server Core or Worker. Replace `size` with a suitably high limit. 512MB should be enough.

See https://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html

Direct memory

To avoid excessive usage of direct memory, we add `-Djdk.nio.maxCachedBufferSize=262144` to command line of Worker. We recommend you to add this system property also to the command line of Server Core. This system property is available since java 1.8.0_102.

In Apache Tomcat, add this system property to `JAVA_OPTS` environment variable, which is configured in `bin/setenv.sh` file.

Codecache Size

Some **CloverDX Server** installations can occasionally run into performance issue: JVM is running more than hundred times slower. The issue can be caused by a full code cache (Java SE Embedded: Developer's Guide - Codecache Tuning). The reserved code cache size is platform dependent and can be too small for **CloverDX Server**. It is highly recommended to increase the code cache size using the following JVM argument:

```
-XX:ReservedCodeCacheSize=256m
```

Maximum Number of Open Files

When using resource-demanding components, such as **FastSort**, or when running a large number of graphs concurrently, you may reach the system limit on simultaneously open files. This is usually indicated by the `java.io.IOException: Too many open files` exception.

The default limit is fairly low in many Linux distributions (e.g. 4096 in Ubuntu). Such a limit can be easily exceeded, considering that one FastSort component can open up to 1,000 files when sorting 10 million records. Furthermore, some application containers recommend increasing the limit themselves (8,192 for IBM WebSphere).

Therefore, it is recommended to increase the limit for production systems. Reasonable limits vary from 10,000 to about 100,000 depending on the expected load of **CloverDX Server** and the complexity of your graphs.

The current limit can be displayed in most UNIX-like systems using the **ulimit -Sn** command.

The exact way of increasing the limit is OS-specific and is beyond the scope of this manual.

Maximum Number of Processes or Threads

If you run graphs with many subgraphs containing many components, you may reach the limit on number of threads per user or per system. In this case, you can find `java.lang.OutOfMemoryError: unable to create new native thread` in graph's log.

The current limit on number of processes/threads per user can be displayed in most UNIX-like systems using the **ulimit -Su** command. Note that the documentation on **ulimit** may not distinguish between processes and threads. The limit on number of threads per system can be displayed using the **sysctl kernel.threads-max** command. The exact way of increasing the limit is OS-specific and is beyond the scope of this manual.

Firewall Exceptions

In order to function properly, **CloverDX Server** requires an outside communication. The table below describes both incoming and outgoing communication of **CloverDX Server**. Please, configure your firewall exceptions accordingly.

Table 8.3. Firewall Exceptions

Traffic	Communication	Description & Components
Incoming	HTTP(S)	Communication between Designer and Server
	JMX	Tracking and debugging information
Outgoing (depending on an actual usage)	JDBC	Connection to databases (DBInputTable, DBOutputTable, DBExecute)
	JMX	Receiving and sending JMS messages (JMSReader, JMSWriter, JMS Listener)
	HTTP(S)	Requesting and receiving responses from servers (Readers, WebserviceClient, HTTPConnector)
	SMTP	Sending data converted into emails (EmailSender)
	IMAP/ POP3	Receiving emails (EmailReader)
	FTP/ SFTP/ FTPS:	Remote file reading and writing (readers, writers)

Garbage Collector for Worker

We recommend using the G1 garbage collector for Worker, as it behaves better on huge heaps and causes shorter full stops of the Java Virtual Machine. G1 is the default garbage collector in Java 9 or newer; however, for Java 8 the Parallel garbage collector is the default and G1 is just optional. Because of that, **CloverDX** automatically enables G1 garbage collector for Worker on Java 8.

If worker.javaExecutable (p. 92) is not specified and no specific garbage collector is selected by the worker.jvmOptions (p. 91) property, **CloverDX** uses the G1 garbage collector on Worker by default on Java 8. So if you modify the JVM used by Worker, we don't set G1 by default. Also, if you specify a different garbage collector, we don't override this setting.

Selecting Garbage Collector

Java 8 uses the Parallel garbage collector by default, which we override for Worker to use the G1 garbage collector. If you wish to use a different garbage collector than G1 for Worker, then it must be specified by adding the `-XX:+UseParallelGC` (for Parallel GC) command line option for the Worker's JVM using the worker.jvmOptions (p. 91) property.

> **Note**
>

Chapter 9. Optional Installation Steps

This chapter describes optional installation steps for items not specified in the previous sections.

Adding Libraries to the Worker's Classpath

Worker may need additional libraries, e.g. a JDBC driver library or the Bouncy Castle cryptographic library. There are two ways to add the libraries.

Using default worker classpath directory

Create a `worker-lib` directory in the `${clover.home}` directory and place the libraries there. The path set by [clover.home](p. 82) (p. 82) can be found in **Server GUI** under **Configuration** →**CloverDX Info** →**Server Properties**.

Configuration Property

Create a directory containing the libraries and set the `worker.classpath` configuration property to the path to this directory.

Worker Support for SMB 2.x and 3.x

Worker supports the SMB 2.x and 3.x protocol. It utilizes the [SMBJ library](#) dependent on [Bouncy Castle](#).

Before you start using SMB 2.x/3.x & Bouncy Castle:

1. Go to the official [Latest Bouncy Castle Java Releases](#) page.

2. Locate the section "SIGNED JAR FILES" and download the latest release.

   Download a required `.jar` file. The filename consists of the name `bcprov-jdk15on`, followed by a version number, for example:

   `bcprov-jdk15on-158.jar`

3. a. **Recommended** - after you download the `.jar` file, you can point to the directory containing the file by adding the following line:

      `-Dcloveretl.smb2.bouncycastle.jar.file=path/to/bcprov-jdk15on.jar`

      to the **JVM arguments** field in the Worker (p. 58) tab of Setup GUI.

   b. **Alternatively** - create a `worker-lib` directory in the `${user.data.home}/CloverDX` directory and move the `jar` file there;

4. Now in case of using the recommended method, you only need to restart Worker. When using the alternative method, restart **CloverDX Server** for the changes to take effect.

Chapter 10. Troubleshooting

Since **CloverDX Server** is considered a universal JEE application running on various application servers, databases and JVM implementations, problems may occur during the installation. These can be solved with a proper configuration of the Server environment. This section contains tips for the configuration.

Memory Issues on Derby

If your Server suddenly starts consuming too much resources (CPU, memory) despite having been working well before, it might be caused by a running internal Derby DB. Typically, causes are incorrect/incomplete shutdown of Apache Tomcat and parallel (re)start of Apache Tomcat.

Solution: move to a standard (standalone) database.

How to fix this? Redeploy **CloverDX Server**:

1. Stop Apache Tomcat and verify there are no other instances running. If so, kill them.

2. Backup the configuration file, if you configured any.

3. Delete the `webapps/clover` directory.

4. Start the Apache Tomcat server. It will automatically redeploy **CloverDX Server**.

5. Verify you can connect from Designer and from web.

6. Shutdown Apache Tomcat.

7. Restore the configuration file and point it to your regular database.

8. Start Apache Tomcat.

JAVA_HOME or JRE_HOME Environment Variables Are Not Defined

If you are getting this error message during an attempt to start your application server (mostly Tomcat), perform the following actions.

Linux:

This command will help you set a path to the variable on the server.
`[clover@server /] export JAVA_HOME=/usr/local/jdk1.x.x`

As a final step, restart the application server.

Windows OS:

Set JAVA_HOME to your JDK installation directory, e.g. C:\Program Files\java\jdk1.8.0.

> ## Important
>
> Some **CloverDX** functions requires JDK to work correctly, therefore we do not recommend having only JRE installed.

Apache Tomcat Context Parameters Do Not Have Any Effect

Tomcat may sometimes ignore some context parameters. It may cause strange **CloverDX Server** behavior, since it appears as configured, but only partially. Some parameters are accepted, some are ignored. This issue is rare, however it may occur in some environments. Such behavior is consistent, so restart has no effect. It's possibly related to Tomcat issues: Bug #47516 and Bug #50700 To avoid this, please use a properties file instead of context parameters to configure **CloverDX Server**.

Tomcat Log File catalina.out Is Missing on Windows

Tomcat start batch files for Windows aren't configured to create the catalina.out file which contains the standard output of the application. The catalina.out file may be vital when Tomcat isn't started in the console and an issue occurs. Or even when Tomcat is executed in the console, it may be closed automatically just after the error message appears in it.

Please follow these steps to enable catalina.out creation:

- Modify [Tomcat_home]/bin/catalina.bat and add a parameter /B to the lines where the _EXECJAVA variable is set. There should be two such lines:

  ```
  set _EXECJAVA=start /B [the rest of the line]
  ```

  Parameter /B causes, that "start" command doesn't open a new console window, but runs the command in its own console window.

- Create a new startup file, e.g. [Tomcat_home]/bin/startupLog.bat, containing a single line:

  ```
  catalina.bat start > ..\logs\catalina.out 2<&1
  ```

  It executes Tomcat in the usual way, but the standard output isn't put to the console, but to the catalina.out file.

Then use the new startup file instead of [Tomcat_home]/bin/startup.bat.

clover.war as Default Context on WebSphere (Windows OS)

If you are deploying clover.war on the IBM WebSphere server without the context path specified, be sure to check whether it is the only application running in the context root. If you cannot start **CloverDX Server** on WebSphere, check the log and look for a following message:

```
com.ibm.ws.webcontainer.exception.WebAppNotLoadedException:
Failed to load webapp: Failed to load webapp: Context root /* is already bound.
Cannot start application CloverDX
```

The easiest way to fix the issue is to stop all other (sample) applications and leave only clover.war running on the server. That should guarantee the server will be available in the context root from now on (e.g. http://localhost:9080/).
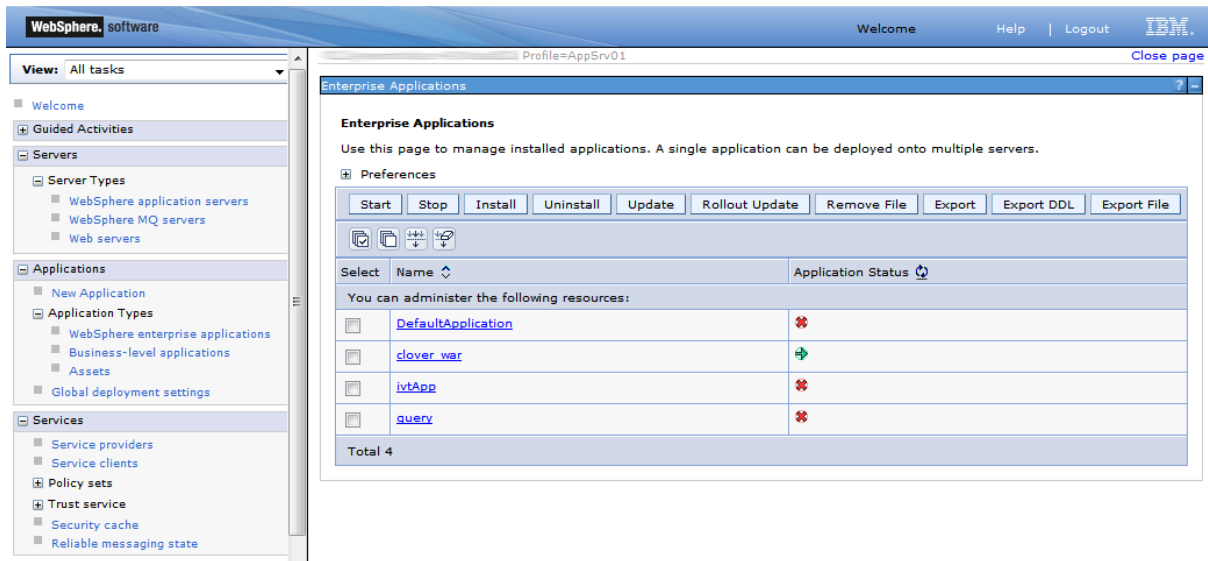

Figure 10.1. CloverDX Server as the only running application on IBM WebSphere

Derby.system.home Cannot be Accessed

If the Server cannot start and the following message is in the log:

java.sql.SQLException: Failed to start database 'databases/cloverserver'

then see the next exception for details. After that, check settings of the `derby.system.home` system property. It may point to an unaccessible directory, or files may be locked by another process. We suggest you set a specific directory as the system property.

Environment Variables and More than one CloverDX Server Instances Running on Single Machine

If you are setting environment variables like `clover_license_file` or `clover_config_file`, remember you should not be running more than one **CloverDX Server**. Therefore, if you ever need to run more instances at once, use other ways of setting parameters (see Part III, "Configuration" (p. 47) for description of all possibilities). The reason is the environment variables are shared by all applications in use causing them to share configurations and fail unexpectedly. Instead of the environment variables, you can use system properties (passed to the application container process using parameter with -D prefix: `-Dclover_config_file`).

Special Characters and Slashes in Path

When working with servers, be sure to follow the folder naming rules. Do not use any special characters in the server path, e.g. spaces, accents, diacritics are not recommended. It can produce issues which are hard to find. If you are experiencing weird errors and cannot trace the source of them, install the application server in a safe destination like:

`C:\JBoss6\`

Similarly, use slashes but never backslashes in paths inside the `*.properties` files, e.g. when pointing to the **CloverDX Server** license file. If you incorrectly use a backslash, it will be considered an escape character and the server may not work properly. This is an example of a correct path:

`license.file=C:/CloverDX/Server/license.dat`

File System Permissions

The application server must be executed by an OS user with proper read/write permissions on file system. Problem may occur, if app-server is executed by a root user for the first time, so log and other temp files are created by root user. When the same app-server is executed by another user, it will fail because it cannot write to root's files.

JMS API and JMS Third-Party Libraries

Missing JMS libraries do not cause failure of the server startup, but it is an issue of deployment on an application server, thus it is still related to this chapter.

`clover.war` itself does not contain `jms.jar`, so it has to be on an application server's classpath. Most of the application servers have `jms.jar` by default, but Tomcat, for example, does not. So if the JMS features are needed, the `jms.jar` has to be added explicitly.

If the "JMS Task" feature is used, there must be third-party libraries on a Server's classpath as well. The same approach is recommended for JMS Reader/Writer components, even if these components allow to specify external libraries. It is due to common memory leak in these libraries which causes "OutOfMemoryError: PermGen space".

Using an Unsupported JDBC Connector for MySQL

CloverDX Server requires MySQL 5 up to version 5.5 included. Using an unsupported JDBC connector for MySQL might cause an exception, for example:

```
could not execute query
You have an error in your SQL syntax; check the manual that corresponds to your MySQL s
to use near 'OPTION SQL_SELECT_LIMIT=DEFAULT' at line 1
```

Illegal Reflective Access Warning

When running **CloverDX Server** on Java 9 and later, an illegal reflective access warning may occur. For example, below is the warning which appears in the log when Worker is starting:

```
2018-10-09 15:54:14,356[    threadPool-1] WorkerProcess        ERROR        [worker0@node01]106.
```

The warning is a result of changes made in Java 9. (see [Understanding Runtime Access Warnings](#)) and has no effect on the function of **CloverDX** applications.

Note: The solution below applies only to the **CloverDX Server** + **Tomcat 9** bundle. The procedure may differ per application server, so if you run another supported application server and want to suppress the warnings, a combination of the flags (both Tomcat 9 default and those added manually) may be required. The changes must be done by your server administrator.

In **CloverDX Server** bundled with Tomcat 9, the warning has been disabled by adding the following flags:

to `JDK_JAVA_OPTS` in `setenv.sh/setenv.bat`:

```
--add-opens=java.base/java.util=ALL-UNNAMED
--add-exports=java.xml/com.sun.org.apache.xerces.internal.parsers=ALL-UNNAMED
```

on Worker's command line:

```
--add-opens=java.base/java.lang=ALL-UNNAMED
--add-opens=java.base/java.util=ALL-UNNAMED
--add-exports=java.xml/com.sun.org.apache.xerces.internal.parsers=ALL-UNNAMED
--add-opens=java.rmi/sun.rmi.transport=ALL-UNNAMED
```

Tomcat 9 default flags in `catalina.sh/catalina.bat`:

```
JDK_JAVA_OPTIONS="$JDK_JAVA_OPTIONS --add-opens=java.base/java.lang=ALL-UNNAMED"
JDK_JAVA_OPTIONS="$JDK_JAVA_OPTIONS --add-opens=java.rmi/sun.rmi.transport=ALL-UNNAMED"
export JDK_JAVA_OPTIONS
```

Part III. Configuration

Chapter 11. Introduction

This part describes in detail the configuration options for **CloverDX Server** used in a production environment. In the following chapters, you will find information on setting required properties and parameters, description of **CloverDX Server**'s Setup GUI elements, parameters for specific database configuration, list of properties used in general configuration, instructions on encrypting confidential properties and log files setting.

> ## Note
>
> We recommend the default installation (without any configuration) only for evaluation purposes. For production use, you should configure a dedicated, system database and set up an SMTP server for sending notifications.

CloverDX Server Configuration Procedure

For initial configuration after the installation of **CloverDX Server**, follow these steps:

- **Choose a configuration source**

  Choose a source of configuration data for **CloverDX Server**. There are several options for configuration sources; however, we recommend using a property file on specified location. For more information, see Chapter 12, Configuration Sources (p. 50).

- **Set up a database dedicated to CloverDX Server**

  Now, you should set up a **CloverDX Server**'s database and configure a connection to the database. Choose a supported database system (p. 10) and read Chapter 14, System Database Configuration(p. 63) for information and examples on how to create a database, add user/role for Clover, grant it required rights/ privileges, etc.

  Once you have set up the database, configure **CloverDX Server**'s connection to the database using the Setup GUI (p. 54).

- **Activate the server with a license**

  After you have set up the database, configured the connection to it and specified the source for configuration data, you can activate the Server with your license. While it is possible to activate the Server immediately after installation, we do not recommend this, since after the activation, the license information is stored in the database. To activate the Server, follow the information in the Activation (p. 32) section.

- **Configure the server**

  Finally, you can configure the server features. The Setup with a user friendly GUI allows you to configure the basic, most important features including, encryption of sensitive data, SMTP for email notifications, etc. For more information, see Chapter 13, Setup (p. 54).

- **Configure Worker**

  Worker is the executor of jobs, all jobs run in the Worker by default. It runs in a separate process (JVM), so it requires configuration in addition to the Server Core.

  Basic configuration of the Worker can be done in the Worker (p. 58) page of the Setup (p. 54) For all configuration properties of Worker see Worker - Configuration Properties(p. 90) See the Chapter 26, Troubleshooting Worker (p. 164) section for useful tips on solving issues.

  The following are the typical areas that need to be configured for Worker:

  - **Heap memory size** - required heap size for Server Core and Worker depends on the nature of executed jobs. In general, Worker should have higher heap allocated, as it runs the jobs which represent the bulk of memory consumption.

    See our recommendations (p. 37) for heap sizes of Worker and Server Core.

Heap size of Worker can be easily configured in the Worker (p. 58) tab of Setup (p. 54) or via the worker.maxHeapSize (p. 91) configuration property.

- **Classpath** - the Worker's classpath is separate from Server Core (i.e. application container classpath). Any libraries needed by jobs executed on Worker need to be added on the Worker's classpath.

  See the worker.classpath (p. 90) configuration property for more details.

- **Command line options** - Worker is started as a separate process with its own JVM. If you need to set JVM command line options, e.g. for garbage collector (p. 40) tweaking, better diagnostics, etc., then you need to set them on the Worker's JVM. See Additional Diagnostic Tools (p. 163) section for useful options for troubleshooting and debugging Worker.

  Command line options of Worker can be easily customized in the Worker (p. 58) tab of Setup (p. 54) or via the worker.jvmOptions (p. 91) configuration property.

- **JNDI** - Worker has its own JNDI pool separate from the application container JNDI pool. If your jobs use JNDI resources (to obtain JDBC or JMS connections), you have to configure the Worker's JNDI pool and its resources.

  See JNDI in worker (p. 93) section for more details.
- **Encrypt the configuration**

  As the last step, we strongly recommend you to encrypt the configuration file to protect your sensitive data.

  For more information, see the Encryption (p. 59) section.
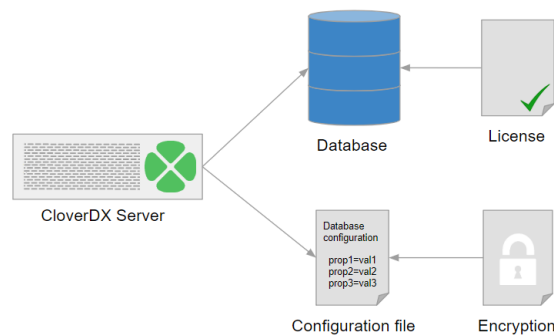
Figure 11.1. CloverDX Server's System Database Configuration

Chapter 12. Configuration Sources

Once installed, **CloverDX Server** requires the configuration of essential (database connection, license, sandboxes) and optional (SMTP, LDAP, etc.) features. The configuration is done by specifying configuration properties (p. 82) in a [property-key]=[property-value] format.

> ## Important
>
> The configuration might contain sensitive data; therefore, **CloverDX Server** enables you to encrypt the configuration properties (for more information, see Chapter 16, Secure Configuration Properties (p. 100)).

CloverDX can load the configuration properties from several sources. We **recommend** the easiest, most convenient way:

Configuration File on Specified Location

The cloverServer.properties configuration file is a text file which contains all **CloverDX** settings. You can edit the file either manually or using a much **simpler and intuitive** Setup GUI (p. 54).

Example of configuration file's content for PostgreSQL database:

```
# Modify the url, username and password for your environment.
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://hostname/clover_db?charSet=UTF-8
jdbc.username=user
jdbc.password=pass
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
```

The path to the file can be specified by system property, context parameter or environment variable.

We recommend specifying the path with the clover.config.file system property.

Example for Apache Tomcat:

Edit bin/setenv.sh (or bin\setenv.bat) and add -Dclover.config.file=/absolute/path/to/cloverServer.properties to CATALINA_OPTS.

For more information and examples on the supported application containers, see Specifying the Path to the Configuration File (p. 52).

Alternative Configuration Sources

There are other sources of configuration properties, as well. Each source containing the configuration data has a different priority. If a property isn't set, application's default setting is used.

> ## Warning
>
> Combining configuration sources could lead to a confusing configuration which would make maintenance much more difficult.

- **Environment Variables**

Environment variables are variables configured by means of your operating system (e.g. `$PATH` is an environment variable).

Set an environment variable with the `clover.` prefix, i.e. (`clover.config.file`).

Some operating systems may not use a dot (`.`) character, so underlines (`_`) may be used instead of dots. So the `clover_config_file` name works as well.

- **System Properties**

  System properties are configured by means of JVM, i.e. with the `-D` argument (`-Dclover.config.file`).

  Set a system property with the `clover.` prefix, i.e. (`clover.config.file`).

  Underlines (`_`) may be used instead of dots (`.`) so the `clover_config_file` name works as well.

- **Configuration File on Default Location**

  A text file containing configured **CloverDX** properties. By default, **CloverDX** searches for the file on the `[AppServerDir]/cloverServer.properties` path.

- **Modification of Context Parameters in web.xml**

  This way **isn't recommended**, since it requires a modification of the WAR file, but it may be useful when none of the approaches above are possible.

  Unzip `clover.war` and modify the `WEB-INF/web.xml` file. Add the following piece of code into the file:

```
<context-param>
    <param-name>[property-name]</param-name>
    <param-value>[property-value]</param-value>
</context-param>
```

- **Context Parameters (Available on Apache Tomcat)**

  Some application servers allow you to set context parameters without modification of the WAR file.

  This way of configuration is possible, but it is **not recommended**, as Apache Tomcat may ignore some context parameters in some environments. Using the configuration file is almost as convenient and much more reliable.

  Example for Apache Tomcat:

  On Tomcat, it is possible to specify context parameters in a context configuration file `[Tomcat_home]/conf/Catalina/localhost/clover.xml` which is created automatically just after deployment of the CloverDX Server web application.

  You can specify a property by adding this element:

```
<Parameter name="[propertyName]" value="[propertyValue]" override="false" />
```

  (Note: by setting the `override` attribute to `false`), the context parameter does not override the default setting associated with the owning host.)

Priorities of Configuration Sources

Configuration sources have the following priorities (from the highest to lowest):

1. **Context parameters**

Context parameters are specified in an application server or directly in a `web.xml` file (**not recommended**).

2. **External configuration file**

   The path to the external configuration file can be specified in several ways. CloverDX Server attempts to find the file in this order (only one of them is loaded):

   a. the path specified with a `config.file` context parameter;

   b. the path specified with a `clover_config_file` or `clover.config.file` system property (**recommended**);

   c. the path specified with a `clover_config_file` or `clover.config.file` environment variable;

   d. the default location (`[AppServerDir]/cloverServer.properties`).

3. **System properties**

4. **Environment variables**

5. **Default values**

Specifying the Path to the Configuration File

uses the configuration file to save the Server's settings. The path to the file is specified by the `clover.config.file` system property. Each application server has a different way to configure the path:

- **Apache Tomcat**

  Edit `bin/setenv.sh` (or `bin/setenv.bat`) and add `-Dclover.config.file=/absolute/path/to/cloverServer.properties` to `CATALINA_OPTS`.

  See also Apache Tomcat (p. 17).

- **JBoss Enterprise Application Platform**

  Edit the configuration file `/standalone/configuration/standalone.xml` and add the following snippet just under the `<extensions>` section:

```
<system-properties>
    <property name="clover.config.file" value="C:/jboss-eap-6.2/cloverServer.properties" />
</system-properties>
```

  See also JBoss Enterprise Application Platform (p. 25).

- **IBM WebSphere**

  1. Go to **Integrated Solutions Console** (default URL:http://localhost:9060/ibm/console/).

  2. Go to **Servers →WebSphere application servers →[Server_name] →Java and Process Management →Process Definition →Java Virtual Machine →Custom Properties**.

  3. Create a system property named `clover_config_file` whose value is a full path to the properties file (e.g. `cloverServer.properties`) on your file system.

  See also IBM WebSphere (p. 21).

- **Weblogic**

Set JAVA_OPTIONS variable in the WebLogic domain start script `[domainHome]/startWebLogic.sh`

```
JAVA_OPTIONS="${JAVA_OPTIONS} -Dclover_config_file=/path/to/clover-config.properties
```

See also .

Note

Continue with:

Chapter 13. Setup

While it is possible to configure **CloverDX Server** by modifying the configuration file (p. 56) in a text editor, the **Setup** with a user-friendly GUI offers a much easier way of configuring basic properties according to your preferences and requirements. Setup is accessible from **Server Console** under **Configuration →Setup**.

It lets you configure:

- License (p. 57)

- Database Connection (p. 57)

- Worker (p. 58)

- Sandbox Paths (p. 58)

- Encryption (p. 59)

- E-mail (p. 59)

- LDAP Connection (p. 60)

- Cluster Configuration (p. 61)

Remember that you should create a database for **CloverDX Server** and add a user/role for **Clover** with appropriate rights **before** you set up the connection to the database in the server's Setup GUI.

CloverDX Server requires a working database connection for storing license information. Therefore, it allows you to access the Setup and configure the connection **prior** the Server **activation** - simply log in the Server Console and click the **Close** button. Otherwise, you would have to activate the server again, after switching from Derby to a new system database.

Important

To access the Setup section, you need the Server Setup permission (p. 141).

Tip

To keep your settings and data in case of a database migration (e.g. from evaluation to production environment), see Chapter 23, Server Configuration Migration (p. 155).

Before You Start

Before you start using the Setup, you have to specify the path to the configuration file where the Setup saves the settings, and add required libraries to the classpath.

1. **Specify the Path to the Configuration File**

   Setup uses the configuration file to save the server's settings. The path to the file is specified by the `clover.config.file` system property. Each application server has a different way to configure it; for more information, see the Specifying the Path to the Configuration File (p. 52) section.

2. **Add Libraries to the Classpath**

As the next step, you should place the libraries required for further configuration on the application server's classpath (in most cases, this is done by placing the files into a specific directory). You usually need a JDBC driver for the connection to the database or a `.jar` file with an encryption provider.

For example, in case of Apache Tomcat, copy the libraries to the `TOMCAT/lib/` folder. Then, restart the application server.

3. **Configure Particular Items**

Now, you can use the Setup to configure the rest of the Server's features (see the following section). The configuration is then saved into a file defined in the `clover.config.file` property.

If you wish to encrypt the sensitive data in you configuration file, set up the **Encryption** first.

4. **Restart the Server (if needed)**

Some changes require you to restart the Server. These changes are indicated by the ⟳ icon. Other changes (e.g. License, Sandboxes, E-mail, LDAP) are applied immediately and do not require a restart.

Using Setup

Each setup page consists of a menu with setup tabs on the left, a main configuration part in the middle and a configuration status and text on the right side.

The main configuration part contains several buttons:

- **Save** saves changes made to the configuration file. The changes in the configuration must be valid.
- **Save Anyway** saves the configuration even if it is invalid. For example, a database connection is considered invalid if a required library is missing.
- **Validate** validates the configuration on a current tab. If you see the **Save** button disabled, use **Validate** to validate the configuration first.
- **Discard Changes** discards unsaved changes and returns to currently used values.

The following icons can appear in the Server GUI:

✔ configured tab

✿ inactive tab

⚠ error

⚠ warning

⟳ restart required

▣ pending changes which have not been saved yet

If an error/warning icon appears, a status message on the right side of the Setup GUI will provide relevant details.

If you start the Server without configuration, you will see decorators pointing to the Setup. The decorators mark problems which require your attention. The displayed number corresponds to the number of items.
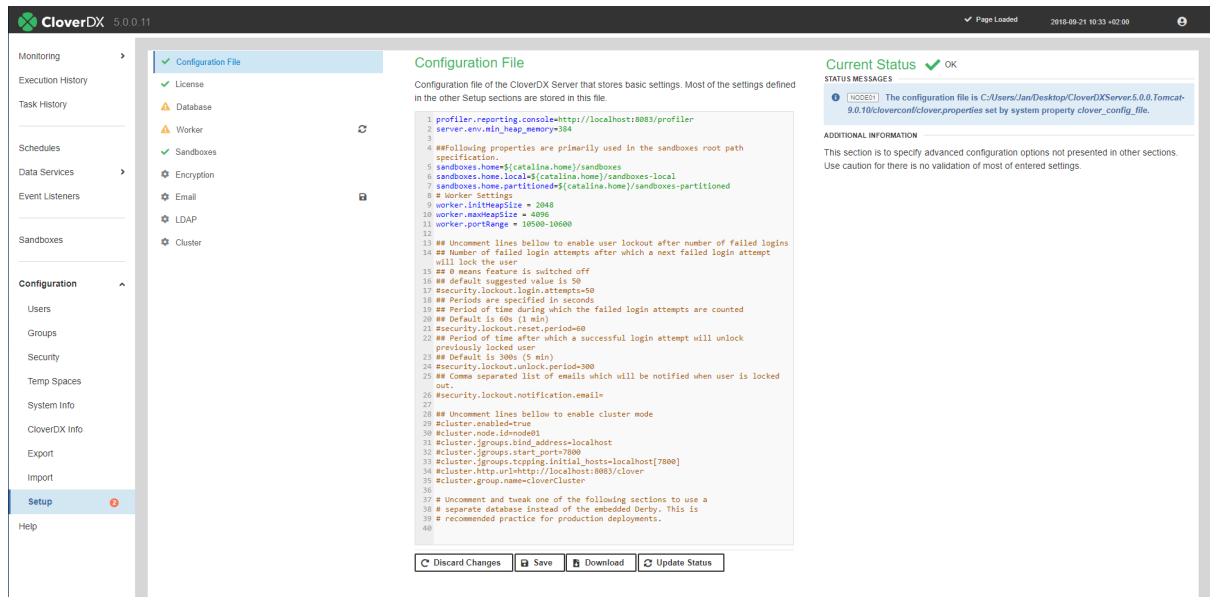
Figure 13.1. Setup GUI with decorators

Configuration File

The **Configuration** tab displays the content of the configuration file. For basic setting, you do not have to edit the content of the file manually. Instead, use the particular Setup tab to configure the corresponding subsystem. For advanced setting, see Chapter 15, List of Configuration Properties (p. 82).

Figure 13.2. Example of the Server Configuration file

License

The **License** tab lets you **add/update**, **show** or **reload** the license. The type of the license determines what **CloverDX Server**'s features are activated.

> ![Important] **Important**
>
> The license is **stored in the database**; therefore, you should configure the database prior activating the Server with a license. Otherwise, you will have to specify the license again.
>
> Similarly, if you change the database, you will be prompted to re-activate the Server.

Figure 13.3. The License tab

Database

The **Database** tab lets you configure the connection to the database. You can connect via:

- **JDBC**

  Under JDBC connection, choose your **Database** from the first drop-down list. This will enable you to choose a **Database URL** template from the second drop-down list. In this template (e.g. `jdbc:postgresql://host:5432/dbname` for PostgreSQL), replace the *host* and *dbname* keywords with proper values.

  Next, enter the **User name** and **Password** for your database (default User name/Password: clover/clover).

  > ![Note] **Note**
  >
  > An Apache Derby JDBC 4 compliant driver is bundled with **CloverDX Server**. If you use another database system, add a JDBC 4 compliant driver on the classpath .

- **JNDI**

  With JNDI, you can access the Datasource on an application server level. Select your **Database** and choose a suitable item from a JNDI tree For more information, see [JNDI DB Datasource](#) (p. 75).

Figure 13.4. Database connection configuration for JDBC (left) and JNDI (right)

Worker

The **Worker** tab lets you configure Worker properties.

You can change **Initial** and **Maximum heap size** for Worker, include **JVM arguments** and set **Port range** for communication with Worker.

Changes in Worker require restart - click on the **Actions** button in the right pane and select **Finish jobs and restart** or **Restart now**.

Figure 13.5. The Worker tab

Sandboxes

The **Sandboxes** tab lets you configure a path to shared (p. 276), local (p. 276) and partitioned (p. 277) sandboxes.

Figure 13.6. Sandbox path configuration with clustering disabled (left) and enabled (right)

Encryption

To secure your sensitive data in the configuration file, you can use the Encryption feature. In the **Encryption** tab, choose a desired **Encryption provider** and **Encryption algorithm**; among the default algorithms, the *PBEWithSHA1AndDESede* is the strongest.

However, since the default algorithms are generally weaker, we **recommend** using Bouncy Castle - a free custom JCE (Java Cryptography Extension) provider offering a **higher strength of encryption**.

Note that in case you use a custom provider, the libraries have to be added on the `classpath` (in the same way as database libraries).

Figure 13.7. Encryption configuration

E-Mail

The **E-mail** tab lets you configure a connection to an SMTP server so the **CloverDX Server** can send an email (p. 172) reporting the server status / events on the server.

To make sure the configuration of **Outgoing SMTP Server** is correct, you can check that the SMTP server can be reached by sending a **test email** from the dialog.

Figure 13.8. E-mail configuration

LDAP

The **LDAP** tab lets you use an existing LDAP database for user authentication.

Specify a connection to the LDAP server and define a pattern for user DN. **Login Test** allows you to validate the login using any user matching the pattern.

For more information, see LDAP Authentication (p. 124).

Figure 13.9. LDAP configuration

Cluster

The **Cluster** tab lets you configure clustering features.

In case your license does not allow clustering, the `Enable clustering` checkbox is grayed out and the note `The license does not allow clustering.` appears at the top.

For more information, see Part VII, "Cluster" (p. 275).

Figure 13.10. Cluster configuration

Chapter 14. System Database Configuration

The **CloverDX Server** license, as well as user's information, event listeners and other services, are saved in a database. For stability and performance reasons, the default Apache Derby database is **not** supported for production environment; therefore, you should choose one of the supported DB systems.

> ⚠ **Important**
>
> Since **CloverDX Server** stores important data in a database, you should create a system database and set up a working connection **before** you activate the Server with license and configure it.

For details on how to set up a connection to an external system database, see the list of examples below. The examples contain details on creating databases in DB systems supported by **CloverDX Server** and configuring a working connection between the database and the Server.

It is possible to specify common **JDBC** DB connection properties (see below) or a **JNDI** location of DB Datasource.

Clustered Deployment

In a clustered deployment, at least one node in the cluster must have a DB connection configured. Other nodes may have their own direct connection (to the same DB), or may use another node as a proxy for persistent operations; however, the scheduler is active only on nodes with a direct connection. For more information about the feature, see Part VII, "Cluster" (p. 275).

Setting up a CloverDX Server's System Database

1. **Create a database**

   - Choose one of the supported database systems and create a database dedicated to **CloverDX Server**. Add a user/role for Clover and grant it required rights/privileges.

2. **Configure common JDBC connection properties**

   - Some JDBC connection properties are common for all supported database systems. If you use a **properties file** for configuration, specify these properties:

jdbc.driverClassName Class name for JDBC driver name.	
jdbc.url	JDBC URL used by **CloverDX Server** to store data.
jdbc.username JDBC database username.	
jdbc.password JDBC database password.	
jdbc.dialect Hibernate dialect to use in Object-relational mapping (ORM).	

3. **Add a JDBC 4 compliant driver on the classpath.**

   - As the last step, add a JDBC 4 compliant driver on the classpath. A JDBC Driver which doesn't meet JDBC 4 won't work properly.

Below is a list of examples of individual database systems configurations.

Examples of Database Configurations

- Embedded Apache Derby (p. 65)
- MySQL (p. 66)

For officially supported versions of particular database systems, see [Database servers](#) (p. 10).

Embedded Apache Derby

The Apache Derby embedded DB is used with a default **CloverDX Server** installation. It uses the working directory as a storage for data persistence by default. This may be a problem on some systems. In case of any problems with connecting to Derby DB, we recommend you configure a connection to external DB or at least specify the Derby home directory:

Configure the `derby.system.home` system property to set path which is accessible for application server. You can specify this system property with this JVM execution parameter:

`-Dderby.system.home=[derby_DB_files_root]`

Example of a properties file configuration:

```
jdbc.driverClassName=org.apache.derby.jdbc.EmbeddedDriver
jdbc.url=jdbc:derby:databases/cloverDb;create=true
jdbc.username=user
jdbc.password=password
jdbc.dialect=com.cloveretl.server.dbschema.DerbyDialect
```

Take a closer look at the `jdbc.url` parameter. The `databases/cloverDb` part means a subdirectory for DB data. This subdirectory will be created in the directory which is set as `derby.system.home` (or in the working directory if `derby.system.home` is not set). You may change the default value `databases/cloverDb`.

A Derby JDBC 4 compliant driver is bundled with **CloverDX Server**, thus there is no need to add it on the classpath.

Note

Continue with:

MySQL

CloverDX Server supports MySQL 5, up to version 5.6 included.

Creating Database

The following steps will create a `clover_db` database and the `clover` user with `clover` password.

1. Create database `clover_db`, set charset and collate.

```
CREATE SCHEMA clover_db CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

2. Use clover_db as the current database.

```
USE clover_db;
```

3. Create a new user with password and host.

```
CREATE USER 'clover'@'%' IDENTIFIED BY 'clover';
```

4. Add all privileges to user 'clover' in DB clover_db.

```
GRANT ALL ON clover_db.* TO 'clover'@'%';
```

5. Reload privileges.

```
FLUSH privileges;
```

CloverDX Server Setup

Example of a properties file configuration:

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://127.0.0.1:3306/clover_db?useUnicode=true&characterEncoding=utf8
jdbc.username=clover
jdbc.password=clover
jdbc.dialect=org.hibernate.dialect.MySQLDialect
```

Add a JDBC 4 compliant driver on the classpath. A JDBC Driver which doesn't meet JDBC 4 won't work properly.

> **Note**
>
> **➡ Continue with:** Encrypted JNDI (p. 76) or Activation (p. 32)

DB2

Creating Database

1. Create a dedicated user for the **CloverDX** database and set a password (UNIX/Linux).

```
useradd clover
```

```
passwd clover
```

2. Create a new database.

```
db2 "CREATE DATABASE cloverdb PAGESIZE 32768 RESTRICTIVE"
```

3. Activate the database.

```
db2 activate db cloverdb
```

4. Connect to the database.

```
db2 connect to cloverdb
```

5. Grant the user DBADM authority (DBADM authority is an administrative authority for a specific database. The database administrator possesses the privileges that are required to create objects and issue database commands. By default, DATAACCESS and ACCESSCTRL authority are also granted).

```
db2 "GRANT DBADM ON DATABASE TO USER clover"
```

6. Disconnect from database

```
db2 connect reset
```

CloverDX Server Setup

Example of a properties file configuration:

```
jdbc.driverClassName=com.ibm.db2.jcc.DB2Driver
jdbc.url= jdbc:db2://localhost:50000/clover
jdbc.username=clover
jdbc.password=clover
jdbc.dialect=org.hibernate.dialect.DB2Dialect
```

Add a JDBC 4 compliant driver on the classpath. A JDBC driver which doesn't meet JDBC 4 specifications won't work properly.

Troubleshooting

Wrong pagesize

The *cloverdb* database has to be created with suitable `PAGESIZE`. DB2 has several possible values for this property: 4096, 8192, 16384 or 32768.

CloverDX Server should work on DB with `PAGESIZE` set to 16384 or 32768. If the `PAGESIZE` value is not set properly, there should be an error message in the log file after failed **CloverDX Server** startup.

The error indicating wrong pagesize:

```
ERROR:
DB2 SQL Error: SQLCODE=-286, SQLSTATE=42727, SQLERRMC=16384;
ROOT, DRIVER=3.50.152
```

SQLERRMC contains suitable value for `PAGESIZE`.

Solution:

You can create a database with a proper page size using the `PAGESIZE` command, e.g.:

```
CREATE DB clover PAGESIZE 32768;
```

The table is in the reorg pending state

On rare occasions, the ALTER TABLE commands may cause tables to remain in "reorg pending state". This behavior is specific for DB2. The ALTER TABLE DDL commands are executed only during the first start of a new **CloverDX Server** version.

The issue may return the following error messages:

```
Operation not allowed for reason code "7" on table "DB2INST2.RUN_RECORD"..
SQLCODE=-668, SQLSTATE=57016

DB2 SQL Error: SQLCODE=-668, SQLSTATE=57016, SQLERRMC=7;DB2INST2.RUN_RECORD, DRIVER=3.5(
```

In this case, the "RUN_RECORD" table is in the "reorg pending state" and "DB2INST2" is the DB instance name.

Solution:

Go to DB2 console and execute the following command (for table run_record):

```
reorg table run_record
```

DB2 console output should look like this:

```
db2 => connect to clover1
Database Connection Information

Database server        = DB2/LINUX 9.7.0
SQL authorization ID   = DB2INST2
Local database alias   = CLOVER1

db2 => reorg table run_record
DB20000I  The REORG command completed successfully.
db2 => disconnect clover1
DB20000I  The SQL DISCONNECT command completed successfully.
```

"clover1" is DB name

DB2 does not allow ALTER TABLE which trims DB column length.

This problem depends on the DB2 configuration and has been experienced only on some AS400s, so far. **CloverDX Server** applies a set of DP patches during the first installation after the application upgrade. Some of these patches may apply column modifications which trim the length of the text columns. These changes never truncate any data, however DB2 does not allow this since it "may" truncate some data. DB2 refuses these changes even in empty DB table.

Solution:

Disable the DB2 warning for data truncation, restart **CloverDX Server** which applies patches, then enable DB2 warning again.

DB2 on AS/400

The connection on AS/400 might be slightly different.

Example of a properties file configuration:

```
jdbc.driverClassName=com.ibm.as400.access.AS400JDBCDriver
jdbc.username=user
jdbc.password=password
jdbc.url=jdbc:as400://host/cloversrv;libraries=cloversrv;date format=iso
jdbc.dialect=org.hibernate.dialect.DB2400Dialect
```

Use credentials of your OS user for `jdbc.username` and `jdbc.password`.

`cloversrv` in `jdbc.url` above is the name of the DB schema.

You can create the schema in AS/400 console:

1. execute command `STRSQL` (**SQL console**)

2. execute command

```
CREATE COLLECTION cloversrv IN ASP 1
```

`cloversrv` is the name of the DB schema and it may be at most 10 characters long

Proper JDBC driver must be in the application server classpath.

Use `jt400ntv.jar` JDBC driver found in `/QIBM/ProdData/Java400` on the server.

Add a JDBC 4 compliant driver on the classpath. A JDBC driver which doesn't meet JDBC 4 specifications won't work properly.

> **Note**
>

Oracle

Creating Database

Run the following script to create a role (`cloverRole`), user (`cloverUser`) with password (`cloverPassword`) and tablespace for **CloverDX Server**:

```
-- Create a new role and grant it privileges.
CREATE ROLE cloverRole NOT IDENTIFIED;

GRANT CREATE SESSION TO cloverRole;
GRANT ALTER SESSION TO cloverRole;
GRANT CREATE TABLE TO cloverRole;
GRANT CREATE SEQUENCE TO cloverRole;
GRANT CREATE TRIGGER TO cloverRole;

-- Create a new database user with password.
CREATE USER cloverUser IDENTIFIED BY cloverPassword;

-- Set quota on tablespace.
GRANT UNLIMITED TABLESPACE TO cloverUser;

-- Connect a new role to a new user.
GRANT cloverRole TO cloverUser;
```

CloverDX Server Setup

Example of a properties file configuration:

```
jdbc.driverClassName=oracle.jdbc.OracleDriver
jdbc.url=jdbc:oracle:thin:@host:1521:db
jdbc.username=cloverUser
jdbc.password=cloverPassword
jdbc.dialect=org.hibernate.dialect.Oracle10gDialect
```

Add a JDBC 4 compliant driver on the classpath. A JDBC driver which doesn't meet the JDBC 4 specifications won't work properly.

These are the privileges which have to be granted to a schema used by CloverDX Server:

```
CONNECT
CREATE SESSION
CREATE/ALTER/DROP TABLE
CREATE/ALTER/DROP SEQUENCE

QUOTA UNLIMITED ON <user_tablespace>;
QUOTA UNLIMITED ON <temp_tablespace>;
```

> ## Note
>

Microsoft SQL Server

Creating Database

It is advised to use SQL Server Authentication instead of Windows Authentication. To enable it, select the server instance in Microsoft SQL Server Management Studio: go to **Properties** →**Security** →**Server authentication** and select the SQL Server and Windows Authentication mode. The server instance needs to be restarted.

> ## Note
>
> Make sure you have:
>
> - TCP/IP `Enabled` in **SQL Server Network Configuration** →**Protocols**
>
> - TCP Port set to **1433** in TCP/IP **Properties** →**IP Addresses** →**IPAll**

1. Create a new database

```
CREATE DATABASE clover_db;
```

2. Enable Read Committed Snapshot Isolation on the new database

```
ALTER DATABASE clover_db SET READ_COMMITTED_SNAPSHOT ON;
```

3. Create a new login role.

```
CREATE LOGIN clover WITH PASSWORD = 'clover', DEFAULT_DATABASE = clover_db;
```

4. Connect to the database.

```
USE clover_db;
```

5. Create a new database user.

```
CREATE USER clover FOR LOGIN clover;
```

6. Add a database role membership db_owner (Members of the db_owner fixed database role can perform all configuration and maintenance activities on the database, and can also drop the database).

```
EXEC sp_addrolemember 'db_owner','clover';
```

CloverDX Server Setup

Using MS SQL requires configuration of the database server:

1. Run **Microsoft SQL Server Management Studio** tool;

2. Create a new user under **Security**/**Logins**;

3. Under **Databases** create a new database (e.g. 'clover') for **CloverDX Server**, select the user from the previous step as owner of the database;

4. Under database **Options**, set the **Is Read Committed Snapshot On** option to **True**.

Example of a properties file configuration:

```
jdbc.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
jdbc.url=jdbc:sqlserver://localhost:1433;instance=SQLSERVERINSTANCE;database=clover_db
jdbc.username=clover
jdbc.password=clover
jdbc.dialect=org.hibernate.dialect.SQLServerDialect
```

Add a JDBC 4 compliant driver on the application server classpath. A JDBC driver that does not meet the JDBC 4 specifications will not work properly.

Note

▶**Continue with:**   Encrypted JNDI (p. 76) or Activation (p. 32)

PostgreSQL

Creating Database

Advanced users can create their own table space

We are going to create a database for **CloverDX** to use a 'user group' role which will own the database and a user role which we will add to the user group. This user role will be then used by the Server to access the database.

Database name: clover_db

UserGroup: cloverdx

User: clover

Password: clover

1. Optionally, you can create a new tablespace

2. Connect as postgres (default admin) to the default DB postgres and execute the following commands:

```
CREATE ROLE cloverdx NOSUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT NOLOGIN;
CREATE ROLE clover NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT LOGIN ENCRYPTED PASSWORD 'clover';
GRANT cloverdx TO clover;
CREATE DATABASE clover_db;
GRANT ALL ON DATABASE clover_db TO cloverdx;
REVOKE ALL ON DATABASE clover_db FROM public;
```

To separate the database into its own tablespace, create a tablespace before creating the database.

and use the following command to create the database:

```
CREATE DATABASE clover_db WITH OWNER cloverdx TABLESPACE tablespace_name;
```

For more information, see the PostgreSQL documentation.

CloverDX Server Setup

Example of a properties file configuration:

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://localhost/clover_db?charSet=UTF-8
jdbc.username=clover
jdbc.password=clover
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Add a JDBC 4 compliant driver on the classpath. A JDBC driver which doesn't meet the JDBC 4 specifications won't work properly.

The JDBC driver for PostgreSQL can be downloaded from the official PostgreSQL page.

Example for Apache Tomcat: place the libraries into the `TOMCAT/lib` directory.

See also [PostgreSQL documentation on jdbc URL](#).

Note

JNDI Configuration and Encryption

JNDI DB Datasource

CloverDX Server can connect to a database using JNDI Datasource which is configured in an application server.

Example for Apache Tomcat and PostgreSQL database:

- **JNDI Datasource Definition**

  First you need to **define a JNDI Datasource** in an application server. The following context resource configuration may be added to the [Tomcat_home]/conf/server.xml file to the <Host> element.

  Note: Do not put the code into the <GlobalNamingResources> element, since the resource would not be visible by the **CloverDX** webapp.

```
<Context path="/clover">
  <Resource name="jdbc/clover_server"
            auth="Container"
            type="javax.sql.DataSource"
            factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
            driverClassName="org.postgresql.Driver"
            url="jdbc:postgresql://127.0.0.1:5432/clover_db"
            username="clover"
            password=""
            maxTotal="20"
            maxIdle="10"
            maxWaitMillis="-1"/>
</Context>
```

- **JNDI Connection Configuration**

  Now that the Datasource is defined, you should **configure the connection**.

  The following parameters may be set in the same way as other parameters (in the properties file or the Tomcat context file). You can also set the parameters in the Database (p. 57) tab of the Setup (p. 54) GUI.

```
datasource.type=JNDI                                        # type of Datasource; must be set, because the default value
datasource.jndiName=jdbc/clover_server                      # JNDI location of DB Datasource; the default value is java:c
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect # Set the dialect according to DB which DataSource is connect
                                                            # The correct dialect can be found in the examples of DB conf
```

Since the DB connection contains sensitive information (e.g. username, password, etc.), **CloverDX** provides the JNDI Encryption (p. 76) feature.

> **Tip**
>
> The resource configuration may also be added to the context file [Tomcat_home]/conf/Catalina/localhost/clover.xml.

> **Important**
>
> Special characters typed in the context file have to be specified as XML entities, e.g. ampersand "&" as "&", etc.

For a detailed list of parameters which can be set up in the configuration file, see Chapter 15, List of Configuration Properties (p. 82).

JNDI Datasource Troubleshooting

JNDI Datasource in Oracle WebLogic

CloverDX's default Quartz configuration does not work with default JNDI Datasource from WebLogic. Proceed with one of the following options:

1. Configure the Quartz JDBC delegate manually **before** the server is started;

2. Disable JDBC type wrapping in the WebLogic's Datasource configuration.

Apache Tomcat's DBCP JNDI pool

The default JNDI pool **DBCP** in **Apache Tomcat** does not handle connections efficiently. With the **DBCP** JNDI pool, low performance can be seen if DBOutputTable with returning statement is used.

Therefore, **tomcat-jdbc-pool** is used instead by adding the `factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"` attribute to the definition of the JNDI resource. See The Tomcat JDBC Connection Pool

Encrypted JNDI

The encryption feature allows you to protect your sensitive data defined in the Datasource definition (e.g. username, password, etc.), which are by default stored in plain text. The configuration differs between particular application servers.

Encrypted JNDI on Tomcat (p. 77)
Encrypted JNDI on JBoss 7 (p. 78)
Encrypted JNDI on WebSphere 8.5.5.0 (p. 80)
Encrypted JNDI on WebLogic (p. 81)

Encrypted JNDI on Tomcat

You need `secure-cfg-tool` to encrypt the passwords. Use the version of `secure-cfg-tool` corresponding to the version of **CloverDX Server**. Usage of the tool is described in Chapter 16, Secure Configuration Properties (p. 100).

Use `encrypt.sh` or `encrypt.bat` for password encryption. Place the encrypted password into a configuration file, and put `cloverdx-secure-jndi-resource-{version}.jar` and `jasypt-1.9.0.jar` files on the classpath of the application server. The `.jar` files can be found in the `tomcat-secure-jndi-resource` directory packed in secure-cfg-tool.

The `tomcat-secure-jndi-resource` directory contains a useful `README` file with further details on encrypted JNDI.

Example of encrypted JNDI connection for PostgreSQL

Encrypt the password:

1. `./encrypt.sh -a PBEWithSHA1AndDESede`

2. The configuration is placed in `${CATALINA_HOME}/conf/context.xml`. Note that the encryption algorithm PBEWithSHA1AndDESede is not default.

```
<Resource name="jdbc/clover_server"
        auth="Container"
        factory="com.cloveretl.secure.tomcatresource.Tomcat8SecureDataSourceFactory"
        secureAlgorithm="PBEWithSHA1AndDESede"
        type="javax.sql.DataSource"
        driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://127.0.0.1:5432/clover_db?charSet=UTF-8"
        username="conf#rPz5Foo7HPn4dFTRV5Ourg=="
        password="conf#4KlNp8/FVDR+rTWX0dEqWA=="
        maxTotal="20"
        maxIdle="10"
        maxWaitMillis="-1"/>
```

If you use other JCE (e.g. Bouncy Castle), it has to be added to the classpath of the application server (`${CATALINA_HOME}/lib`). The encrypt command requires the path to directory with JCE, too.

```
./encrypt.sh                    -l                    ~/lib/                    -c
org.bouncycastle.jce.provider.BouncyCastleProvider                    -a
PBEWITHSHA256AND256BITAES-CBC-BC
```

```
<Resource name="jdbc/clover_server"
        auth="Container"
        factory="com.cloveretl.secure.tomcatresource.Tomcat8SecureDataSourceFactory"
        secureProvider="org.bouncycastle.jce.provider.BouncyCastleProvider"
        secureAlgorithm="PBEWITHSHA256AND256BITAES-CBC-BC"
        type="javax.sql.DataSource"
        driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://127.0.0.1:5432/clover_db?charSet=UTF-8"
        username="conf#Ws9IuHKo9h7hMjPllr31VxdI1A9LKIaYfGEUmLet9rA="
        password="conf#Cj1v59Z5nCBHaktn6Ubgst4Iz69JLQ/q6/32Xwr/IEE="
        maxTotal="20" maxIdle="10"
        maxWaitMillis="-1"/>
```

Encrypted JNDI on JBoss 7
JBoss 7 - JBoss EAP 6.2.0.GA - AS 7.3.0.Final-redhat-14

> ### Note
>
> For details, see [Using Encrypted DataSource Password in JBoss AS7](#).

Configuration steps are similar to configuring of JBoss 6.

The configuration takes place in a single configuration file, e.g. for standalone profile `JBOSS_HOME/standalone/configuration/standalone.xml`.

Original data source:

```xml
<datasources>
    <datasource jndi-name="java:/MysqlDS" pool-name="MySQLPool">
        <connection-url>jdbc:mysql://localhost:3306/clover</connection-url>
        <driver>mysql</driver>
        <pool>
            <max-pool-size>30</max-pool-size>
        </pool>
        <security>
            <user-name>user</user-name>
            <password>password</password>
        </security>
    </datasource>

    <drivers>
        <driver name="mysql" module="com.cloveretl.jdbc">
            <driver-class>com.mysql.jdbc.Driver</driver-class>
        </driver>
    </drivers>
<datasources>
```

1. In JBOSS_HOME directory run the cli command:

```
java -cp modules/system/layers/base/org/picketbox/main/picketbox-4.0.19.SP2-redhat-1.jar:client/jboss-logging.jar
```

   The command will return an encrypted password, e.g. 5dfc52b51bd35553df8592078de921bc.

2. Add a new security-domain to security-domains, the password value is a result of the command from the previous step.

```xml
<security-domain name="EncryptDBPassword" cache-type="default">
    <authentication>
        <login-module code="org.picketbox.datasource.security.SecureIdentityLoginModule" flag="required">
            <module-option name="username" value="user"/>
            <module-option name="password" value="5dfc52b51bd35553df8592078de921bc"/>
            <module-option name="managedConnectionFactoryName" value="jboss.jca:service=LocalTxCM,name=MysqlPo
        </login-module>
    </authentication>
</security-domain>
```

3. Replace user and password with a reference to the security domain.

```xml
<datasources>
    <datasource jndi-name="java:/MysqlDS" pool-name="MysqlPool" enabled="true" use-java-context="true">
        <connection-url>jdbc:mysql://localhost:3306/clover</connection-url>
        <driver>mysql</driver>
        <pool>
            <max-pool-size>30</max-pool-size>
        </pool>
        <security>
```

```
                <security-domain>EncryptDBPassword</security-domain>
            </security>
        </datasource>

        <drivers>
            <driver name="mysql" module="com.cloveretl.jdbc">
                <driver-class>com.mysql.jdbc.Driver</driver-class>
            </driver>
        </drivers>
    </datasources>
```

It is possible that the same mechanism can also be used for JMS.

Encrypted JNDI on WebSphere 8.5.5.0

In WebSphere, user credentials aren't saved in plain text, but as J2C authentication data. (see How to Create a
WAS JDBC Provider, J2C Authentication Alias, and Data Source for the IBM i).

The same mechanism can also be used for JMS connection (see IBM's instructions on Configuring an external
JMS provider).

Encrypted JNDI on WebLogic

Password in a JNDI datasource file is encrypted by default when created by admin's web console (Service/ Datasource).

Example of datasource file (located in the `DOMAIN/config/jdbc/` directory):

```
<?xml version='1.0' encoding='UTF-8'?>
<jdbc-data-source xmlns="http://xmlns.oracle.com/weblogic/jdbc-data-source" xmlns:sec="http://xmlns.oracle.com/web
  <name>MysqlDS</name>
  <jdbc-driver-params>
    <url>jdbc:mysql://127.0.0.1:3306/clover</url>
    <driver-name>com.mysql.jdbc.Driver</driver-name>
    <properties>
      <property>
        <name>user</name>
        <value>user</value>
      </property>
    </properties>
    <password-encrypted>{AES}zIiq6/JutK/wD4CcRPXlpOueIlKqc6uRVxAnZZcC3pI=</password-encrypted>
  </jdbc-driver-params>
  <jdbc-connection-pool-params>
    <test-table-name>SQL SELECT 1</test-table-name>
  </jdbc-connection-pool-params>
  <jdbc-data-source-params>
    <jndi-name>jdbc/MysqlDS</jndi-name>
    <global-transactions-protocol>OnePhaseCommit</global-transactions-protocol>
  </jdbc-data-source-params>
</jdbc-data-source>
```

The same mechanism is also used for encrypting password in the JMS connection (see Oracle's instructions on Configuring an external JMS provider).

Note

81

Chapter 15. List of Configuration Properties

Below you can find the *configuration properties* available in **CloverDX Server**. The essential properties can be configured using the [Setup GUI (p. 54)](#). Other properties serve to tweak various features of **CloverDX Server**. However, these properties have to be configured manually, e.g. by editing the configuration file.

In **CloverDX Server** UI, you can view the properties and their values in Configuration > CloverDX Info > Server Properties.

Additional properties used for cluster configuration can be found in Chapter 41, [Sandboxes in Cluster](#) (p. 276).

> ## Important
>
> *Configuration property* and *system property* are not the same. *Configuration properties* can be configured in **Setup** section or in `cloverdx.properties` file. *System properties* serve to configure the JVM. E.g. in Apache Tomcat, they are configured in `bin/setenv.[bat|sh]` file using `-D` prefix.

General Configuration Properties

Table 15.1. General configuration

Key	Description	Default Value
Configuration		
clover.config.file	Absolute path to location of a CloverDX Server configuration file	/absolute/path/to/ cloverServer.properties
clover.home	By default, this property is commented out and has a dynamically computed value: path containing `CloverETL` value for current (pre 5.0) installations and `CloverDX` for new (5.0 and newer) installations. If defined by the user, value has a higher priority. The property can be overridden using: environment variable `clover.clover.home`	${user.data.home}/ CloverETL or ${user.data.home}/ CloverDX

Key	Description	Default Value
	context parameter `<Parameter name="clover.home">` or system property `-Dclover.clover.home=`	
License		
license.file	Absolute path to location of a CloverDX Server license file (license.dat)	
license.context_names	A comma-separated list of web-app contexts which may contain license. Each of them has to start with a slash! Works only on Apache Tomcat.	/clover-license,/ clover_license
Engine		
engine.config.file	location of a **CloverDX** engine configuration properties file	properties file packed with **CloverDX**
engine.plugins.additional.src	This property may contain an absolute path to some "source" of additional **CloverDX** engine plugins. These plugins are not a substitute for plugins packed in WAR. "Source" may be a directory or a zip file. Both, a directory and a zip, must contain a subdirectory for each plugin. Changes in the directory or the ZIP file apply only when the server is restarted. For details see Chapter 34, Extensibility - CloverDX Engine Plugins (p. 231).	empty
Sandboxes		
sandboxes.home	This property is primarily intended to be used as a placeholder in the sandbox root path specification. So the sandbox path is specified with the placeholder and it's resolved to the real path just before it's used. The sandbox path may still be specified as an absolute path, but placeholder has some significant advantages: * sandbox definition may be exported/imported to another environment with a different directory structure * user creating sandboxes doesn't have to care about physical location on the filesystem * each node in cluster environment may have a different "sandboxes.home" value, so the directory structure doesn't have to be identical For backward compatibility, the default value uses the content of the clover.home (p. 82) configuration property.	${clover.home}/ sandboxes
sandboxes.access.check.boundaries.enabled	true | false If it is set to false, then the path relative to a sandbox root may point out of the sandbox. No file/folder outside of the sandbox is accessible by the relative path otherwise.	true
Database connection		
datasource.type	Set this explicitly to JNDI if you need **CloverDX Server** to connect to a DB using JNDI datasource.	JDBC

Key	Description	Default Value
	In such case, "datasource.jndiName" and "jdbc.dialect" parameters must be set properly. Possible values: JNDI | JDBC	
datasource.jndiName	JNDI location of a DB DataSource. It is applied only if "datasource.type" is set to "JNDI".	java:comp/env/jdbc/ clover_server
jdbc.driverClassName	class name for JDBC driver name	
jdbc.url	JDBC URL used by **CloverDX Server** to store data	
jdbc.username	JDBC database user name	
jdbc.password	JDBC database password	
jdbc.dialect	hibernate dialect to use in ORM	
quartz.driverDelegateClass	SQL dialect for quartz. Value is automatically derived from "jdbc.dialect" property value.	
Security		
private.properties	List of server properties which are used only by the **CloverDX Server** code. So these properties are not accessible outside of the ServerFacade. By default, there are all properties which may contain password in the list, so their values are not visible for web GUI users. The values are replaced by a single star "*". Changes in this list may cause unexpected behavior of some server API.	jdbc.password, executor.password, security.ldap.password, clover.smtp.password
security.session.validity	Session validity in milliseconds. When the request of logged-in user/client is detected, validity is automatically prolonged.	14400000
security.session.exchange.limit	Interval for exchange of invalid tokens in milliseconds.	360000
security.default_domain	Domain in which all new users are included. Stored in user's record in the database. Shouldn't be changed unless the "clover" must be white-labelled.	clover
security.basic_authentication.features_list	List of features which are accessible using HTTP and which should be protected by Basic HTTP Authentication. The list has form of semicolon separated items; Each feature is specified by its servlet path.	/request_processor;/ simpleHttpApi;/ launch;/launchIt;/ downloadStorage;/ downloadFile;/ uploadSandboxFile;/ downloadLog;/ webdav
security.basic_authentication.realm	Realm string for HTTP Basic Authentication.	CloverDX Server
security.digest_authentication.features_list	List of features which are accessible using HTTP and which should be protected by HTTP Digest Authentication. The list has form of semi-colon separated items. Each feature is specified by its servlet path. Please keep in mind that HTTP Digest Authentication is feature added to the version 3.1. If you upgraded your older **CloverDX Server** distribution, users created	

Key	Description	Default Value
	before the upgrade cannot use the HTTP Digest Authentication until they reset their passwords. So when they reset their passwords (or the admin does it for them), they can use Digest Authentication as well as new users.	
security.digest_authentication.storeA1.enabled	Switch whether the A1 Digest for HTTP Digest Authentication should be generated and stored or not. Since there is no **CloverDX Server** API using the HTTP Digest Authentication by default, it's recommended to keep it disabled. This option is not automatically enabled when any feature is specified in the security.digest_authentication.features_list property.	false
security.digest_authentication.realm	Realm string for HTTP Digest Authentication. If it is changed, all users have to reset their passwords, otherwise they won't be able to access the server features protected by HTTP digest Authentication.	CloverDX Server
security.digest_authentication.nonce_validity	Interval of validity for HTTP Digest Authentication specified in seconds. When the interval passes, server requires new authentication from the client. Most of the HTTP clients do it automatically.	300
security.lockout.login.attempts	The number of failed login attempts after which a next failed login attempt will lock the user. Set the value to 0 to disable the function. Since 4.8.0M1.	50
security.lockout.reset.period	Period of time in seconds during which the failed login attempts are counted. Since 4.8.0M1.	60
security.lockout.unlock.period	Period of time in seconds after which a successful login attempt will unlock the previously locked user. Since 4.8.0M1.	300
security.csrf.protection.enabled	Enable/disable protection of Simple HTTP API against CSRF attacks, enabled by default. The CSRF protection requires presence of the `X-Requested-By` header in the requests. For more details, see the section called "CSRF Protection" (p. 234).	true
SMTP		
clover.smtp.transport.protocol	SMTP server protocol. Possible values are "smtp" or "smtps".	smtp
clover.smtp.host	SMTP server hostname or IP address	
clover.smtp.port	SMTP server port	
clover.smtp.authentication	true/false If it is false, username and password are ignored.	
clover.smtp.username	SMTP server username	
clover.smtp.password	SMTP server password	
clover.smtp.additional.*	Properties with a "clover.smtp.additional." prefix are automatically added (without the prefix) to the	

Key	Description	Default Value
	Properties instance passed to the Mailer. May be useful for some protocol specific parameters. The prefix is removed.	
Logging		
logging.project_name	Used in log messages where it is necessary to name the product name.	CloverDX
logging.default_subdir	Name of a default subdirectory for all server logs; it is relative to the path specified by system property "java.io.tmpdir". Don't specify as an absolute path, use properties which are intended for absolute path.	cloverlogs
logging.logger.server_audit.enabled	Enables logging of operations called on ServerFacade and JDBC proxy interfaces. The name of the output file is "server-audit.log". It is stored in the same directory as other **CloverDX Server** log files by default. The default logging level is DEBUG so it logs all operations which may process any change.	false
logging.logger.server_integration.enabled	Enables logging of Designer-Server calls. The name of the output file is "server-integration.log". It is stored in the same directory as other **CloverDX Server** log files by default. The default logging level is INFO. Username is logged, if available. JDBC and CTL debugging is not logged.	true
launch.log.dir	Location, where server should store launch requests logs. See Chapter 39, <u>Launch Services</u>(p. 249) for details.	${java.io.tmpdir}/ [logging. default_subdir]/ launch where ${java.io.tmpdir} is system property
graph.logs_path	Location, where server should store Graph run logs. See Chapter 17, <u>Logging</u> (p. 103) for details.	${java.io.tmpdir}/ [logging. default_subdir]/ graph where ${java.io.tmpdir} is system property
logging.appender.jobs.pattern_layout	Pattern of the jobs' log messages	%d %-5p %-3X{runId} [%t] %m%n
logging.appender.jobs.encoding	Encoding of the jobs' log files	UTF-8
logging.mem_appender.WORKER.pattern_layout	Format of log that can be seen in **Monitoring** > **Logs** > **Worker**.	
logging.mem_appender.WORKER.size_limit	Size of log that can be seen in **Monitoring** > **Logs** > **Worker**.	
Thread Manager		
threadManager.pool.corePoolSize		4

Key	Description	Default Value
	Number of threads which are always active (running or idling). Related to a thread pool for processing server events.	
threadManager.pool.queueCapacity Max size of the queue (FIFO) which contains tasks waiting for an available thread. Related to a thread pool for processing server events. For queueCapacity=0, there are no waiting tasks, each task is immediately executed in an available thread or in a new thread.		0
threadManager.pool.maxPoolSize Max number of active threads. If no thread from a core pool is available, the pool creates new threads up to "maxPoolSize" threads. If there are more concurrent tasks then maxPoolSize, thread manager refuses to execute it.		8192
threadManager.pool.allowCoreThreadTimeOut Switch for idling threads timeout. If true, the "corePoolSize" is ignored so all idling threads may be time-outed		false
threadManager.pool.keepAliveSeconds timeout for idling threads in seconds		20
Archivator		
task.archivator.batch_size	Max number of records deleted in one batch. It is used for deleting of archived run records.	50
task.archivator.archive_file_prefix Prefix of archive files created by the archivator.		cloverArchive_
Properties resolver		
properties_resolver.resolve_server_props.server_props_list_additional A list of properties from a subset of properties, which values are resolved. The properties' values may use system properties or environment variables as placeholders. The values are resolved during the server startup. If the system property is changed later, the resolved **CloverDX Server** property value doesn't change. Users may use this property, if some property they need to resolve is missing in the property: properties_resolver.resolve_server_props.server_props_list_default. If the property to resolve is already specified by the property properties_resolver.resolve_server_props.server_props_list_default, don't add it to this property.		
properties_resolver.resolve_server_props.server_props_list_default A list of properties from a subset of properties, which values are resolved. The properties' values may use system properties or environment variables as placeholders. Values are resolved during the server		clover.home, sandboxes.home, sandboxes.home.local, sandboxes.home.partitioned, cluster.jgroups.bind_address,

Key	Description	Default Value
	startup. If the system property is changed later, the resolved **CloverDX Server** property value doesn't change. Users are discouraged from modification of the property, unless it's necessary. Instead, users may add more properties by modifying property: properties_resolver.resolve_server_props.server_props_list_additional	cluster.jgroups.start_port, cluster.jgroups.external_address, cluster.jgroups.external_port, cluster.jgroups.tcpping.initial_hosts, cluster.group.name, cluster.ssl.http.url
properties_resolver.placeholders.server_props_list_default	A list of properties from a subset of properties, that may be used as placeholders and shall be resolved if used in paths. The properties can be used if you define a path to the root of a sandbox, or to locations of local or partitioned sandboxes, or path to a script, or path in archiver job. Users are strongly discouraged from modification of the property. The property name changed since **CloverDX 4.2**, however the obsolete name is also still accepted to maintain backwards compatibility.	clover.home, sandboxes.home, sandboxes.home.local, sandboxes.home.partitioned, user.data.home
Data Services		
dataservice.invocation.record.max.age	It sets the maximal age in minutes before the record is removed from the database. The default is 1440 min = 24 h.	1440
dataservice.failure.ratio.min.record.count	Used for Data Service failure indication. It represents the minimum number of invocations required to evaluate whether the percentage of failures is over the threshold. Ensures that during periods of low traffic the endpoint does not switch to failing state. 10 by default.	10
API		
launch.http_header_prefix	Prefix of HTTP headers added by launch services to the HTTP response.	X-cloveretl
http.api.enabled	Enables or disables simple HTTP API. If the HTTP API is disabled, there is no link to HTTP API operations in login page, the HTTP API tab in Launch Service that is accessible under **Test** button is not visible, and the HTTP API, the /clover/httpapi.jsp and HTTP API servlet are not accessible. Available since 4.8.0M1. See Chapter 36, Simple HTTP API (p. 234).	true
webDav.method.propfind.maxDepth	Maximum depth for webDAV method PROPFIND. When the depth is not specified, the default is supposed to be infinite (according to the rfc2518), however it's necessary to set some limit, otherwise the webDav client might overload the server filesystem. Also if the depth value specified by webDAV client in the request is higher than the pre-configured max depth, only the pre-configured maximum is used.	40

Key	Description	Default Value
JVM		
server.env.min_heap_memory	Sets the required minimal heap memory threshold. If the configuration of **CloverDX Server** is set to less heap memory, a warning is displayed. Experienced users can change the default value to avoid the warning when running the server on a system with lower memory. The threshold is in megabytes.	900
server.env.min_nonheap_memory	Sets the required minimal non-heap memory threshold. If the configuration of **CloverDX Server** is set to less non-heap memory, a warning is displayed. Experienced users can change the default value to avoid the warning when running the server on a system with lower memory. The threshold is in megabytes.	256
jvm.implementation.check.enabled	Displays warnings when unsupported Java implementation is used.	true
Misc		
temp.default_subdir	Name of a default subdirectory for server tmp files; it is relative to the path specified by system property "java.io.tmpdir".	clovertmp
graph.pass_event_params_to_graph_in_old_style	Since 3.0. It is a switch for backwards compatibility of passing parameters to the graph executed by a graph event. In versions prior to 3.0, all parameters are passed to executed graph. Since 3.0, just specified parameters are passed. Please see Start a Graph (p. 178) for details.	false
cluster.node.invocation.record.info.interval	Sets the interval for synchronization of the Data Services health state between the cluster nodes. The time is in milliseconds.	30000
clover.event.fileCheckMinInterval	Interval of the timer, running file event listener checks (in milliseconds). See File Event Listeners (remote and local) (p. 220) for details.	1000
clover.event.groovyCheckMinInterval		1000
clover.inDevelopment		1000
launchservices.visible	Display Launch Services (p. 249) in the main menu in Server UI. Can be true or false.	false

Worker - Configuration Properties

Table 15.2. Server - Worker configuration

Key	Description	Default Value
worker.initialWorkers	Enable/disable the Worker. To enable Worker, set to 1 (this is the default). **To disable Worker and run all jobs in Core Server, set to 0.** Starting more than one Worker is currently (in 4.9.0) not supported.	1
worker.portRange	Port range used for communication between Server Core and Worker and between Workers on different cluster nodes. Communication between Server Core and Worker is done on localhost. Workers on different cluster nodes communicate directly with each other over these ports - in Cluster setup, this port range should be open in firewall for other Cluster nodes. This property can be easily configured in the Worker (p. 58) tab of Setup (p. 54). `worker.portRange` should contain at least 5 ports for 1 node (depending on other options, a node takes at most 5 ports from the range). We recommend to use `portRange` of at least 10 ports to avoid possible problems with occupied ports after restart of Worker. If more cluster nodes run on the same machine, make sure that there are enough free ports for Workers of all cluster nodes on the machine. The default configuration of `worker.portRange` is sufficient for that.	10500-10600
worker.connectTimeout	Timeout for connection initialization between Worker and Server Core, in both directions. The timeout is in milliseconds. This setting can be useful in case of handling communication issues between Server Core and Worker, typically under high load you might want to increase the timeout.	5000
worker.readTimeout	Read timeout for communication requests between Worker and Server Core, in both directions. If a request is not completely served before reaching this limit, the connection is terminated. The timeout is in milliseconds. This setting can be useful in case of handling communication issues between Server Core and Worker, typically under high load you might want to increase the timeout.	600000
worker.classpath	A directory with additional `.jar` files to be added to the Worker's classpath. The `.jar` files would typically be libraries used by graphs (e.g. JDBC drivers for database connections) or JDBC drivers used in JNDI connections defined in Worker (see Worker - JNDI Properties (p. 93)).	${clover.home}/ worker-lib

Key	Description	Default Value

	The Worker's classpath is separate from Server Core (i.e. application container classpath). Any libraries needed by jobs executed on Worker need to be added on the Worker's classpath.
	
	For backward compatibility, the default value uses the content of the [clover.home](#) (p. 82) configuration property.
	
	The property can contain paths to multiple directories. The separator between the directories can be a colon (on Linux and Mac) or semicolon (Linux, Mac and Windows), e.g.:
	
	`worker.classpath=/home/clover/`
	`worker-lib;/opt/worker-lib-2`
	
	If a directory is added on the Worker's classpath, its subdirectories are automatically added too.
	
	Some basic wildcards are supported: `directory-*` and `directory-?`.	

worker.maxHeapSize	The maximum Java heap size of Worker in MB, it will be translated to the `-Xmx` option for the Worker's JVM. Jobs executed in the Worker require heap memory based on their complexity, dataset size, etc. See our recommendations (p. 37) for heap sizes of Worker and Server Core. This property can be easily configured in the Worker (p. 58) tab of Setup (p. 54). Setting to 0 uses Java default heap size (automatically determined by Java). This setting is not recommended for production usage.	0
worker.initHeapSize	The initial Java heap size of Worker in MB, it will be translated to the `-Xms` option for the Worker's JVM. We recommend to set this to the same value as `worker.maxHeapSize` This property can be easily configured in the Worker (p. 58) tab of Setup (p. 54). Setting to 0 uses Java default initial heap size (automatically determined by Java). This setting is not recommended for production usage.	0
worker.jvmOptions	Adds Java command line options for the Worker's JVM. This property is useful to tweak the configuration of the Worker's JVM, e.g. to tune garbage collector settings. These command line options override default options of the JVM. For example to enable parallel garbage collector: `-XX:` `+UseParallelGC`.	

Key	Description	Default Value
	See Additional Diagnostic Tools (p. 163) section for useful options for troubleshooting and debugging Worker. This property can be easily configured in the Worker (p. 58) tab of Setup (p. 54).	
worker.enableDebug	Remote Java debugging of Worker, enables JDWP. Enabling this allows you to connect a Java debugger remotely to the running Worker process, to debug your Java transformations, investigate issues, etc. The port used by the debugger is determined dynamically and can be seen in the Worker section of the Monitoring page.	false

worker.inheritSystemProperties	Sets whether system Java properties are inherited from the Server Core process to the Worker process. We automatically inherit some system properties to simplify the Worker configuration, see below for a list.
	
	This functionality is enabled by default. Use this property to disable this behavior in case some of the inherited properties would cause issues.
	
	The following system Java properties are inherited from the Server Core to Worker:
	
	`# Clover properties`
	`com.opensys.cloveretl.addressdoctor.setConfigFile, com.opensys.c`
	`cloveretl.smb2.bouncycastle.jar.file`
	
	`# Standard Java properties`
	`java.library.path`
	`java.io.tmpdir`
	`XX:MaxPermSize`
	`XX:MaxMetaspaceSize`
	
	`# SSL related properties`
	`javax.net.ssl.keyStore, javax.net.ssl.keyStorePassword`
	`javax.net.ssl.trustStore, javax.net.ssl.trustStorePassword`
	`javax.net.ssl.keyAlias`
	`https.protocols`
	
	`# Proxy configuration`
	`*.proxyHost, *.proxyPort, *.proxyUser, *.proxyPassword, *.nonPro`
	`socksProxyHost, socksProxyPort, socksProxyVersion, java.net.sock`
	`java.rmi.server.hostname`
		true

worker.javaExecutable	Absolute path to the Java binary for Worker process, e.g. `/user/local/java/bin/java`. Use this property if you need to use a specific Java binary for running the Worker.	Value is automatically determined based on $JAVA_HOME environment variable.

Worker - JNDI Properties

The Worker has its own JNDI pool separate from the application container JNDI pool. If your jobs use JNDI resources (to obtain JDBC or JMS connections), you have to configure the Worker's JNDI pool and its resources.

The worker JNDI properties must be configured using the `clover.properties` configuration file. Libraries used by the JNDI resources must be added to the Worker's classpath, see worker.classpath (p. 90).

It is possible to define multiple datasources pointing to different databases or JMS queues, see examples below. The datasources are indexed in configuration, their properties have suffix [0], [1], etc. Even a single datasource must have the [0] index.

JDBC Datasources

Worker uses the Apache DBCP2 pool for its JNDI functionality. Any DBCP2 configuration attribute is supported, see [DBCP attributes](). The only mandatory properties are `jndiName` and `url`.

See table below for basic JNDI properties.

You can monitor the state of the datasources via JMX. See Additional Diagnostic Tools (p. 163) for details on how to enable JMX on Worker. Then you can connect to the Worker's JMX interface with tools like `jconsole` and monitor the JNDI datasources, e.g. for the number of currently open connections. The related MBeans are under the `Tomcat/DataSource/localhost///javax.sql.DataSource` path:
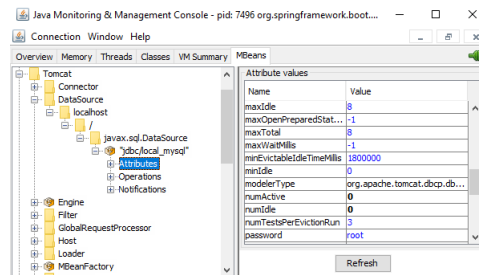
Figure 15.1. MBean for a JNDI datasource in jconsole

Table 15.3. Properties for JDBC JNDI Resources in Worker

Key	Description	Example
worker.jndi.datasource[0].jndiName	The name of the JNDI datasource. Mandatory.	jdbc/database_name
worker.jndi.datasource[0].url	The JDBC connection URL. Mandatory.	jdbc:postgresql://hostname:5432/database_name
worker.jndi.datasource[0].username	The user name for a database connection.	clover
worker.jndi.datasource[0].password	The password for a database connection. The password value can be encrypted using the secure configuration tool, see Secure Configuration Properties (p. 100).	clover
worker.jndi.datasource[0].driverClassName	The database driver classname. The database driver must be on the Worker classpath, see worker.classpath (p. 90).	org.postgresql.Driver
worker.jndi.datasource[0].maxIdle	The maximum number of idle database connections in a pool. Set to -1 for no limit.	10
worker.jndi.datasource[0].maxTotal	The maximum number of database connections in a pool. Set to -1 for no limit.	20
worker.jndi.datasource[0].maxWaitMillis	The maximum time Worker waits for a database connection to become available. In milliseconds, set to -1 for no limit.	30000
worker.jndi.datasource[0].dbcpAttribute	Any DBCP2 attribute, e.g. `worker.jndi.datasource[0].initSQL`. See [DBCP attributes]().	

The following example shows configuration of two JDBC Datasources.

```
worker.jndi.datasource[0].jndiName=jdbc/postgresql_finance
worker.jndi.datasource[0].url=jdbc:postgresql://finance.example.com:5432/finance
worker.jndi.datasource[0].maxIdle=5
worker.jndi.datasource[0].maxTotal=10
worker.jndi.datasource[0].maxWaitMillis=-1
worker.jndi.datasource[0].username=finance_user
worker.jndi.datasource[0].password=conf#eCflGDlDtKSJjh9VyDlRh7IftAbI/vsH
worker.jndi.datasource[0].driverClassName=org.postgresql.Driver

worker.jndi.datasource[1].jndiName=jdbc/MysqlDB
worker.jndi.datasource[1].url=jdbc:mysql://marketing.example.com:3306/marketing?useUnicode=true&amp;characterEncod
worker.jndi.datasource[1].maxIdle=10
worker.jndi.datasource[1].maxTotal=20
worker.jndi.datasource[1].maxWaitMillis=-1
worker.jndi.datasource[1].username=marketing_user
worker.jndi.datasource[1].password=conf#JWsMa2okg7Dq2gtLBM84sE==
worker.jndi.datasource[1].driverClassName=com.mysql.jdbc.Driver
```

JMS Connections

Worker can use any JMS broker to define JMS connections in JNDI. Any JMS broker configuration attribute is supported. The mandatory properties are `jndiName`, `factoryClass`, `typeClass` and `typeInterface`. See table below for basic JNDI properties for JMS resources.

Table 15.4. Properties for JMS JNDI Resources in Worker

Key	Description	Example
worker.jndi.jms[0].jndiName	The name of the JNDI JMS resource. Mandatory.	`jms/jms_queue`
worker.jndi.jms[0].factory	Factory class for creating the JMS resource. This is JMS broker specific. Mandatory.	`org.apache.activemq.jndi.JN`
worker.jndi.jms[0].type	Implementation class of the JMS resource. This is JMS broker specific. Mandatory.	`org.apache.activemq.command`
worker.jndi.jms[0].jmsProperty	Configuration property for the JMS resource. Any configuration property supported by the JMS broker can be used.	`worker.jndi.jms[0].brokerUr`

The following example shows configuration of several JMS resources.

```
worker.jndi.jms[0].jndiName=jms/CloverConnectionFactory
worker.jndi.jms[0].type=org.apache.activemq.ActiveMQConnectionFactory
worker.jndi.jms[0].factory=org.apache.activemq.jndi.JNDIReferenceFactory
worker.jndi.jms[0].brokerUrl=tcp://localhost:61616?jms.prefetchPolicy.queuePrefetch=1
worker.jndi.jms[0].brokerName=LocalActiveMQBroker

worker.jndi.jms[1].jndiName=jms/CloverQueue
worker.jndi.jms[1].type=org.apache.activemq.command.ActiveMQQueue
worker.jndi.jms[1].factory=org.apache.activemq.jndi.JNDIReferenceFactory
worker.jndi.jms[1].physicalName=TestQueue
```

Worker - SSL Properties

In Cluster, Workers of each node communicate with each other directly for increased performance. This communication is used to transport data of cluster remote edges in clustered jobs between the nodes. For increased security, it is possible to use SSL for the remote edge communication.

SSL communication between Workers needs to be enabled and configured separately from SSL of the application container that runs Server Core. The `worker.ssl.enabled` property is used to enable/disable SSL. If a Cluster node's "self" URL is using HTTPS, we automatically set the property to true. Configuration of SSL consists of setting paths and passwords of KeyStore and TrustStore, see the table below for details.

Note that if the standard SSL related system properties (`javax.net.ssl.keyStore`, `javax.net.ssl.keyStorePassword`, `javax.net.ssl.keyAlias`, `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword`) are used to configure KeyStore/TrustStore for the Server Core, they are propagated to Worker; therefore, their respective `worker.ssl` properties do not need to be configured.

Recommended steps to enable SSL for inter-worker communication are:

* Enable SSL for each cluster node, via the application container settings. Configure TrustStore and KeyStore via the standard `javax.net.ssl.*` properties.
* Set `cluster.http.url` for each node to point to its own HTTPS URL
* Check that communication between Cluster nodes over SSL works and that the nodes can correctly see each other. The Monitoring page of Server Console should show the whole cluster group and its nodes correctly.
* Worker should automatically inherit the above SSL configuration.
* Run a clustered job on Worker

Table 15.5. Properties for SSL communication in Worker

Key	Description	Example
worker.ssl.enabled	Enables or disables an SSL connection for Worker. Note that if the Server runs on HTTPS, SSL is enabled automatically; however, this property has a higher priority.	`true/false`
worker.ssl.keyStore	Absolute path to the KeyStore file.	path/to/keyStore.file
worker.ssl.keyStorePassword	The KeyStore password.	
worker.ssl.keyAlias	The alias of the key in keyStore. **Optional** - the property does not have to be specified if there is only one key in the KeyStore.	
worker.ssl.port	The port for SSL communication with Worker. The property is configured automatically and the value is set from [worker.portRange](#) (p. 90).	
cluster.ssl.disableCertificateValidation	Disable validation of certificates in HTTPS connections of remote edges. Disabling the validation affects jobs run on both Worker and Server Core.	`true/false`

Job Execution Properties

Table 15.6. Defaults for job execution configuration - see Job Config Properties *(p. 150) for details*

Key	Description	Default Value
executor.tracking_interval	An interval in milliseconds for scanning of a current status of a running graph. The shorter interval, the bigger log file.	2000
executor.log_level	Log level of graph runs. TRACE | DEBUG | INFO | WARN | ERROR	INFO
executor.max_job_tree_depth	Defines maximal depth of the job execution tree, e.g. for recursive job it defines the maximal level of recursion (counting from root job).	32
executor.max_running_concurrently	Amount of graph instances which may exist (or run) concurrently. 0 means no limits.	0
executor.max_graph_instance_age	Specifies how long can a graph instance be idling before it is released from memory. Interval is in milliseconds. 0 means no caching. This property has been renamed since 2.8. Original name was executor.maxGraphInstanceAge	0
executor.classpath	Classpath for transformation/processor classes used in the graph. Directory [Sandbox_root]/trans/ does not have to be listed here, since it is automatically added to a graph run classpath.	
executor.skip_check_config	Disables check of graph configuration. Increases performance of a graph execution; however, it may be useful during graph development.	true
executor.password	This property is deprecated. The password for decoding encoded DB connection passwords.	
executor.verbose_mode	If true, more descriptive logs of graph runs are generated.	true
executor.use_jmx	If true, the graph executor registers JMX mBean of the running graph.	true
executor.debug_mode	If true, edges with enabled debug store data into files in debug directory.	false

List of all properties

Chapter 16. Secure Configuration Properties

Some configuration properties can be confidential (e.g. a password to a database, mail client, etc.) and thus it's desirable to encrypt them. For this purpose, there is a command-line utility *secure-cfg-tool.jar*.

Basic Utility Usage

1. Download the utility archive file (`secure-cfg-tool.zip`) and unzip it.

   The utility is available in the download section of your **CloverDX** account - at the same location as the download of **CloverDX Server**.

2. Execute the script given for your operating system, `encrypt.bat` for MS Windows, `encrypt.sh` for Linux. You will be asked for inserting a value of a configuration property intended to be encrypted.

   Example:

   C:\secure-cfg-tool>encrypt.bat

   ****
   Secure config encryption (use --help or -h option to show help)
   ****

   ****** Config settings ******
   Provider: SunJCE
   Algorithm: PBEWithMD5AndDES
   ****

   Enter text to encrypt: mypassword
   Text to encrypt: "mypassword"
   Encrypted text: conf#eCflGDlDtKSJjh9VyDlRh7IftAbI/vsH

   C:\secure-cfg-tool>

   If you want to configure the way the values are encrypted, see

3. The encrypted string has *conf#encrypted_property* format and can be used as a value of a configuration property in the properties file, `clover.xml` file or `web.xml` file (see details about configuration sources in Chapter 12,).

   Example of a configuration property file with encrypted password:

   ```
   jdbc.driverClassName=org.postgresql.Driver
   jdbc.url=jdbc:postgresql://127.0.0.1/clover_db?charSet=UTF-8
   jdbc.username=yourUsername
   jdbc.password=conf#eCflGDlDtKSJjh9VyDlRh7IftAbI/vsH
   jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
   ```

   Alternatively, you can use the following command:

   ```
   java -jar secure-cfg-tool.jar
   ```

> **Important**
>
> Values encrypted by a Secure parameter form (Chapter 20, Secure Parameters (p. 119)) **cannot** be used as a value of a configuration property.

Advanced Usage - Custom Settings

The way of encrypting configuration values described above uses default configuration settings (a default provider and algorithm). If you need to customize the settings, use the following parameters of the *secure-cfg-tool.jar* utility.

Table 16.1. Parameters

Parameter	Description	Example
--algorithm, -a	algorithm to encrypt	--algorithm PBEWithMD5AndDES
--file, -f	config file location	-f C:\User\John \cloverServer.properties
--help, -h	show help	--help
--providerclass, -c	custom provider class	-c org.provider.ProviderClass
--providerlocation, -l	path to jar/folder containing a custom provider class (it will be added to the classpath)	--providerlocation C:\User\John\lib \customprovider.jar, -l C:\User\John \lib\
--providers, -p	print available security providers and their algorithms	--providers

> **Note**
>
> To demonstrate usage of an external provider the Bouncy Castle provider is used.

To find out a list of algorithms, use **-p** or **--providers**

```
C:\secure-cfg-tool>encrypt.bat -p
```

If you want to find out a list of algorithms of an external provider, you must pass the provider's class name and path to jar file(s).

```
C:\secure-cfg-tool>encrypt.bat                        -p                        -c
org.bouncycastle.jce.provider.BouncyCastleProvider  -l C:\User\John\bcprov-
jdk15on-152.jar
```

Result might look like this:

```
***** List of available providers and their algorithms *****
    Provider: SunJCE
    Provider class: com.sun.crypto.provider.SunJCE
      Algorithms:
        PBEWithMD5AndDES
        PBEWithSHA1AndDESede
        PBEWithSHA1AndRC2_40
    Provider: BC
    Provider class: org.bouncycastle.jce.provider.BouncyCastleProvider
      Algorithms:
        PBEWITHMD2ANDDES
        PBEWITHMD5AND128BITAES-CBC-OPENSSL
        PBEWITHMD5AND192BITAES-CBC-OPENSSL
        PBEWITHMD5AND256BITAES-CBC-OPENSSL
```

The provider class is displayed on the row starting with *Provider class*, algorithms are strings with *PBE* prefix. Both can be used to configure encryption.

Configuring the Encryption Process

The algorithm and provider can be passed to the utility in two ways.

- **Using command line arguments**

  To change the algorithm, use the argument **-a**. The provider remains default (SunJCE in case of Oracle Java):

  ```
  C:\secure-cfg-tool>encrypt.bat -a PBEWithMD5AndDES
  ```

  To use an external provider, you must specify the provider's class name (the `--providerclass` or `-c` arguments) and add jar(s) to the classpath (the `--providerlocation` or `-l` arguments). Provider location must point to a concrete jar file or directory containing the jar(s) and can be used several times for several paths:

  ```
  C:\secure-cfg-tool>encrypt.bat  -a  PBEWITHSHA256AND256BITAES-CBC-BC  -c
  org.bouncycastle.jce.provider.BouncyCastleProvider -l C:\User\John\bcprov-
  jdk15on-152.jar
  ```

- **Using configuration file**

  A configuration file is a common properties file (text file with key-value pairs):

  ```
  [property-key]=[property-value]
  ```

  See the following example of `secure.config.example.properties` distributed within `secure-cfg-tool.zip`):

  ```
  security.config_properties.encryptor.providerClassName=org.bouncycastle.jce.provider.BouncyCastleProvider
  security.config_properties.encryptor.algorithm=PBEWITHSHA256AND256BITAES-CBC-BC
  security.config_properties.encryptor.provider.location=C:\\User\\libs
  ```

  You must also set the path to the file using the **-f** argument:

  ```
  C:\secure-cfg-tool>encrypt.bat -f path/to/secure.config.example.properties
  ```

  > **Note**
  >
  > More jar locations can be set in the **security.config_properties.encryptor.providerLocation** property. The locations are delimited by semicolon.

Configuring an application server

CloverDX Server application needs to know how the values have been encrypted, therefore the properties must be passed to the server (see details in Part III, "Configuration" (p. 47)). For example:

```
...
security.config_properties.encryptor.providerClassName=org.bouncycastle.jce.provider.BouncyCastleProvider
security.config_properties.encryptor.algorithm=PBEWITHSHA256AND256BITAES-CBC-BC
...
```

> **Important**
>
> If a third-party provider is used, its classes must be accessible to the application server. Property **security.config_properties.encryptor.providerLocation** will be ignored.

Chapter 17. Logging

Main Logs

The **CloverDX Server** uses the log4j library for logging. The WAR file contains the default log4j configuration. The log4j configuration file `log4j.xml` is placed in `WEB-INF/classes` directory.

By default, log files are produced in the directory specified by the `java.io.tmpdir` system property in the `cloverlogs` subdirectory.

The `java.io.tmpdir` system property usually points to a common system temp directory, i.e. `/tmp`. On Apache Tomcat, it is usually the `$TOMCAT_HOME/temp` directory.

The default logging configuration (`log4j.xml` bundled in the clover.war) may be changed to another log4j configuration file using system property `log4j.configuration`. If you override the configuration, only the properties from the new file are used.

The `log4j.configuration` should contain the URL of the new log4j configuration file, not a simple file system path, for example:

```
log4j.configuration=file:/home/clover/config/log4j.xml
```

Tip

It is better to copy the original file and modify the copy, than to create a new file.

Please note that `log4j.configuration` is not a **CloverDX Server** configuration property, but a system property, thus it must be set on the JVM command line by `-Dlog4j.configuration` or in other way suitable for the application container. See the Installation (p. 11) chapter for more information on how to set a system property for each application container.

Since such a configuration overrides the default configuration, it may influence Graph run logs. So your own log configuration has to contain following fragment to preserve Graph run logs:

```
<logger name="Tracking" additivity="false">
  <level value="debug"/>
</logger>
```

Another Useful Logging Settings

These system properties allow for logging of HTTP requests/responses to stdout:

Client side:

`com.sun.xml.ws.transport.http.client.HttpTransportPipe.dump=true` (For more information, see the Integrating **CloverDX Designer** with **CloverDX Server** chapter of the **CloverDX Designer** User's Guide.)

Server side:

```
com.sun.xml.ws.transport.http.HttpAdapter.dump=true
```

Access Log in Apache Tomcat

If you need to log all requests processed by the server, add the following code to `$CATALINA_HOME/conf/server.xml`.

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
       prefix="localhost_access_log" suffix=".txt"
       pattern="%h %l %u %t %D %r %s %b" />
```

The format defined above has following meaning

```
[IP address] [date-time] [processing duration in milliseconds] [method] [URL] [protocol-
```

The log will look like the next line

```
172.17.30.243 - - [13/Nov/2014:12:53:03 +0000] 2 "POST /clover/spring-rpc/clusterNodeAp
```

See also [Valve](#) in documentation on Apache Tomcat.

Application Server Logs

If you use Apache Tomcat, it logs into `$CATALINA_HOME/logs/catalina.out` file.

Graph Run Logs

Each graph or jobflow run has its own log file – for example, in the Server Console, section Execution History (p. 198).

By default, these log files are saved in the subdirectory cloverLogs/graph in the directory specified by `java.io.tmpdir` system property.

It's possible to specify a different location for these logs with the CloverDX `graph.logs_path` property. This property does not influence main Server logs.

Server Audit Logs

Server Audit Log logs operations called on *ServerFacade* and *JDBC proxy* interfaces.

Audit logging can be enabled by setting (adding) the value of CloverDX property `logging.logger.server_audit.enabled` to true. In server GUI, you can change the property value in **Configuration** →**Setup** →**Configuration File**. *Audit logging* is disabled by default.

The name of output file is `server-audit.log`. The file is in the same directory as main server log files. Default log level is DEBUG, so all operations which may do any change or another important operations (e.g. login or openJdbcConnection) are logged. To enable logging of all operations, change log level to TRACE in the log4j configuration.

Each logged operation is logged by two messages: entering method and exiting method (if the exception is raised, it's logged instead of output parameters)

• Entering method (marked as "inputParams"). All method's parameters (except for passwords) are printed.
• Exiting method (marked as "outputParams"). Method's return value is printed.

- Exception in method (marked as "EXCEPTION"). Exception's stacktrace is printed.

Message also contains:

- username, if the user is known
- client IP address, if it's known
- cluster node ID
- Interface name and the operation name

Values of transient and lazy initialized (in entity classes) fields and fields with binary content are not printed.

Designer-Server Integration Logs

The logging of Designer-Server integration can be enabled with logging.logger.server_integration.enabled (p. 86) configuration property. The name of the log file is `server-integration.log`.

The log format is date and time, IP address of Designer, user name, operation, result of the operation (success/failure) and duration in milliseconds.

```
2018-03-07 16:42:00,525 10.0.3.2 user=clover, operation=executeGraph SUCCESS duration=576 ms
```

Worker Log

Worker log logs operation performed by the Worker. The Worker is configured with a separate `log4j.xml` configuration file.

Part IV. Administration

Chapter 18. Monitoring

Monitoring section in the server Web GUI displays useful information about current performance of the standalone CloverDX Server or all cluster nodes if the clustering is enabled.

Monitoring section of the standalone server has slightly different design from cluster environment. In case of standalone server, the server-view is the same as node detail in cluster environment.

The section is refreshed every 15 seconds so the displayed data is up-to-date. The page can also be refreshed manually by the **Refresh** button.

Standalone Server Detail

Standalone server detail view displays info collected from the standalone Server. The info is grouped in several panels. The following panels are displayed by default.

- Resource Utilization (p. 108)

- Worker (p. 108)

- System and License (p. 108)

- Performance (p. 109)

- CPU Load

- Last 10 running jobs

- Status History (p. 109)

You can display the hidden actions with **Actions** button: choose **Actions →Show details**.
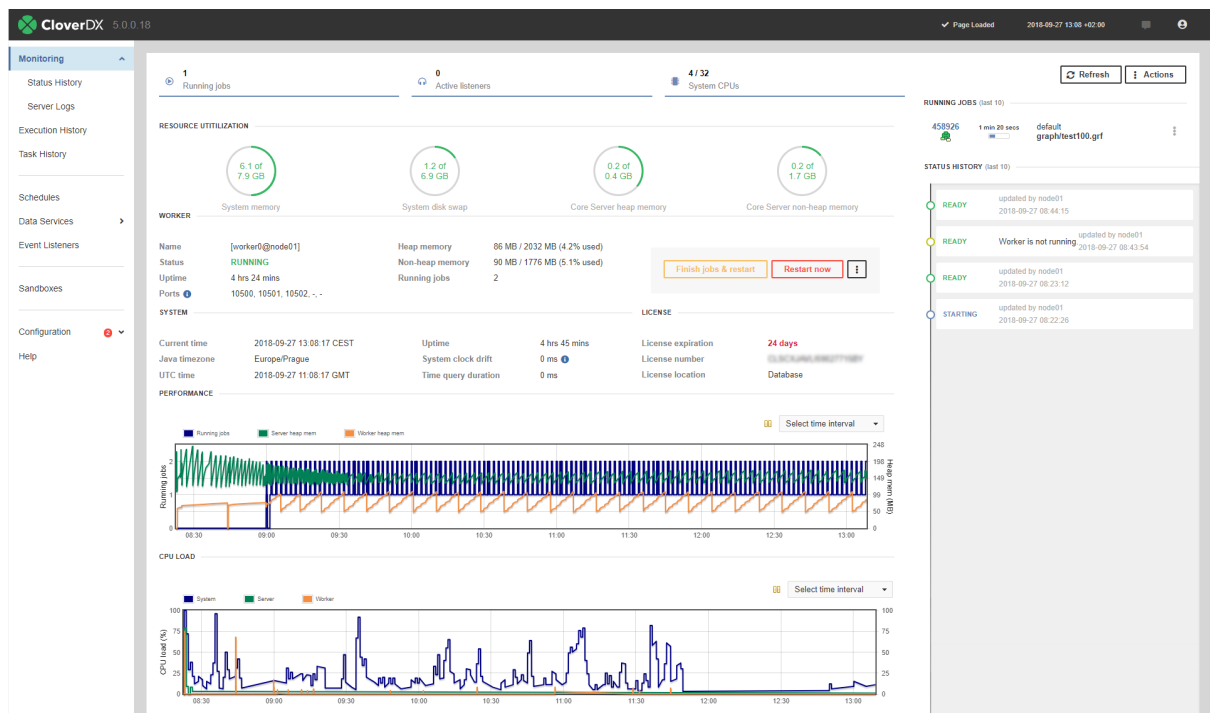
Figure 18.1. Standalone server detail

Overview

The Overview bar displays basic information about the Server:

- **Running jobs** - the number of currently running jobs
- **Active listeners** - the number of active listeners (i.e. listeners performing periodical checks).
- **System CPUs** - the number of current/total system CPUs

Figure 18.2. Standalone Server Overview

Resource Utilization

Resource Utilization panel shows amount of used System memory, System disk swap Core Server heap memory and Core Server non-heap memory.
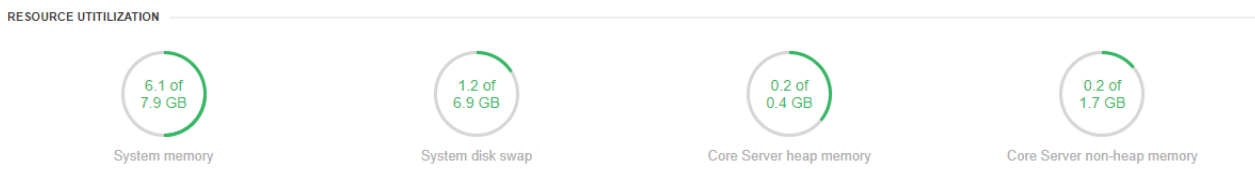
Figure 18.3. Resource Utilization

Worker

Worker panel contains basic information about Worker and offers several actions for managing Worker.
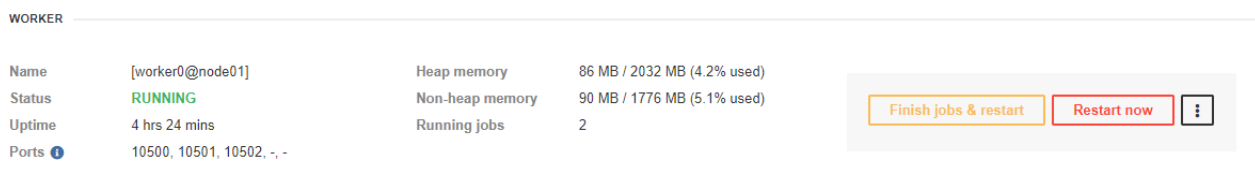
Figure 18.4. Worker

System and License

System panel contains info about operating system and license.
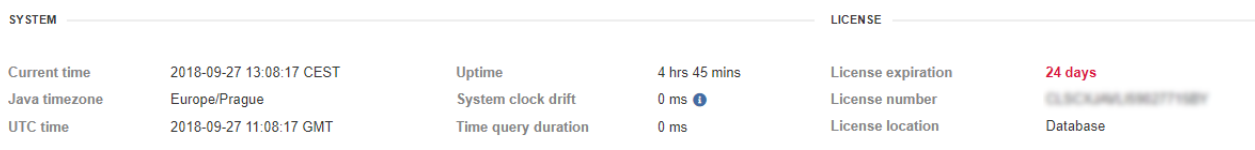
Figure 18.5. System

Performance

The Performance panel contains a chart with three basic performance statistics: a number of running jobs, amount of used up Server and Worker heap memory. The graph displays values gathered within a specific interval. The interval can be set up with the combo box above the graph or configured by the `cluster.node.sendinfo.history.interval` config property.

Note that the heap memory is constantly oscillating, even in idle state, since it is periodically managed by JVM garbage collector (i.e. the temporary data required for running **CloverDX Server** and Worker is periodically removed from/allocated to the heap memory).
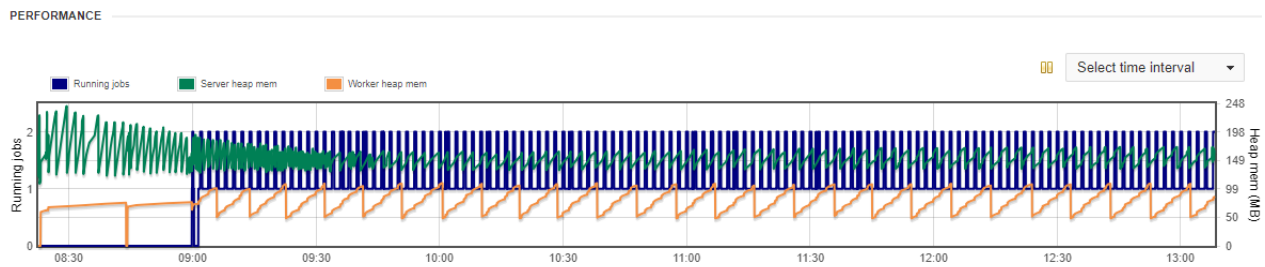
Figure 18.6. Performance

CPU Load

The CPU Load panel displays a chart with info about total CPU load and CPU load caused by JVM (both Core Server and Worker).
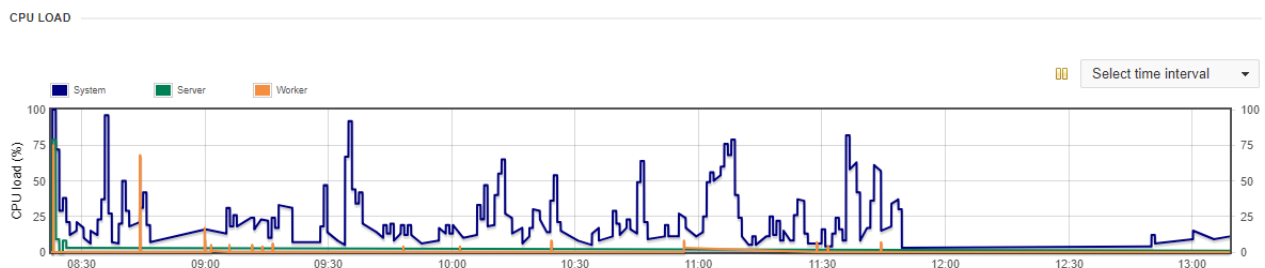
Figure 18.7. CPU Load

Running Jobs

Running jobs panel lists currently running jobs, 10 oldest runs are displayed.

Figure 18.8. Running jobs

Status History

Status history panel displays node statuses history since restart of the Server.
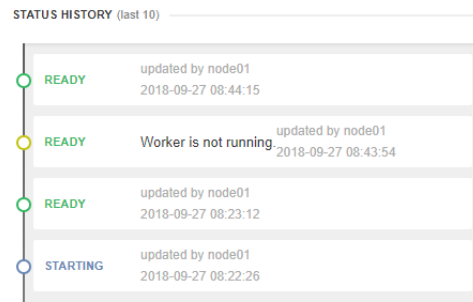
Figure 18.9. Status History

User's Access

User's Access panel lists info about activities on files performed by users. The list displays a timestamp of an event, username, address and name of the method.

Users' Accesses

Date	Username	Address	Method
2018-09-27 09:01:26 CEST	clover	0	createSchedule
2018-09-27 09:00:59 CEST	clover	0	updateSchedule
2018-09-27 09:00:13 CEST	clover	0	killJob
2018-09-27 08:59:58 CEST	clover	0	fireScheduleEventAndSaveScheduleState
2018-09-27 08:59:51 CEST	clover	0	createSchedule
2018-09-27 08:44:45 CEST	clover	0	createEventListener
2018-09-27 08:43:49 CEST	clover	0	restartWorkerNow
2018-09-27 08:24:58 CEST	clover	0	updateLicense

Figure 18.10. Users' Accesses panel

Classloader cache

Classloader cache lists all currently cached classloaders. The classloader cache may be empty as classloader caching is disabled by default.

Status

Status panel displays current node status since last server restart. It displays current server status, exact Java version, exact **CloverDX Server** version, way of access to database, etc.

Status

Last status check time	2018-09-27 13:32:01
Available heap memory (used / commited / max)	124 MB / 264 MB / 456 MB
Available non-heap memory (used / commited / max)	249 MB / 268 MB / 1,776 MB
Status	READY
CloverDX version	5.0.0.18
Java version	1.8.0_144
Active threads	88
Loaded classes	22032
Running jobs	[458943, 458944]
Active locks	[]
Event listeners	
Number of job pools	2
Number of jobs in the pools	0
Number of transformation definitions in use	2
Nodes accessible by asynchronous messaging	
Nodes accessible by synchronous messaging	
Database access	local Apache Derby Embedded JDBC Driver 10.11.1.1 - (1616546)
Asynchronous messaging info	

Figure 18.11. Status

Heartbeat

Heartbeat panel displays a list of heartbeat events and their results.

Heartbeat

Current time	System clock drift	Heap mem used	Active threads	Loaded classes	Running jobs
2018-09-27 13:33:03	0 ms	126 MB	86	22032	[458943]
2018-09-27 13:33:01	0 ms	126 MB	86	22032	[458943]
2018-09-27 13:32:59	0 ms	125 MB	86	22032	[458943]
2018-09-27 13:32:57	0 ms	125 MB	86	22032	[458943]
2018-09-27 13:32:55	0 ms	125 MB	86	22032	[458943]
2018-09-27 13:32:53	0 ms	148 MB	86	22032	[458943]
2018-09-27 13:32:51	0 ms	146 MB	85	22032	[458943]
2018-09-27 13:32:49	0 ms	146 MB	85	22032	[458943]

Figure 18.12. Heartbeat

Threads

Threads panel lists Java threads and their states.

Threads

Thread name	Thread state	Waited time (ms)	Blocked time (ms)
AbortedJobsCleanup	TIMED_WAITING	-1	-1
AsyncFileHandlerWriter-491044090	TIMED_WAITING	-1	-1
Attach Listener	RUNNABLE	-1	-1
ContainerBackgroundProcessor[StandardEngine[	TIMED_WAITING	-1	-1
DRDAConnThread_2	WAITING	-1	-1
DRDAConnThread_3	WAITING	-1	-1
FileHandlerLogFilesCleaner-1	WAITING	-1	-1
Finalizer	WAITING	-1	-1
GC Daemon	TIMED_WAITING	-1	-1
JMX notification receiver	WAITING	-1	-1
NetworkServerThread_1	RUNNABLE	-1	-1
NioBlockingSelector.BlockPoller-1	RUNNABLE	-1	-1
NioBlockingSelector.BlockPoller-2	RUNNABLE	-1	-1

Figure 18.13. Threads

Quartz

Quartz panel lists scheduled actions: their name, description, start time, end time, time of previous event, time of next event and expected final event.

Quartz

Name	Description	Start time	End time	Previous event (within this up-time)	Next event	Expected final event
trigger_294913	Delete old debug files	2018-09-27 08:22:34		2018-09-27 13:30:00	2018-09-27 13:40:00	
trigger_294915	RunGraphLong	2018-09-27 09:01:26		2018-09-27 13:31:26	2018-09-27 13:36:26	
trigger_294914	RunGraph	2018-09-27 08:59:51		2018-09-27 13:35:51	2018-09-27 13:37:51	

Figure 18.14. Quartz

Cluster Overview

Cluster overview displays info collected from all cluster nodes. The info is grouped in several panels:

- Overview (p. 108) similar to standalone Server with some Cluster specific information (e.g. Cluster name, node to which you are connected, etc.)

- List of nodes with a toolbar - allows manipulation with selected nodes

- Status history - Displays last 10 status changes for all cluster nodes

- Node detail - Displays several basic performance attributes for selected nodes. It's visible on the right side only when activated by button on the toolbar.

- Running jobs - It's displayed only when there are running jobs.
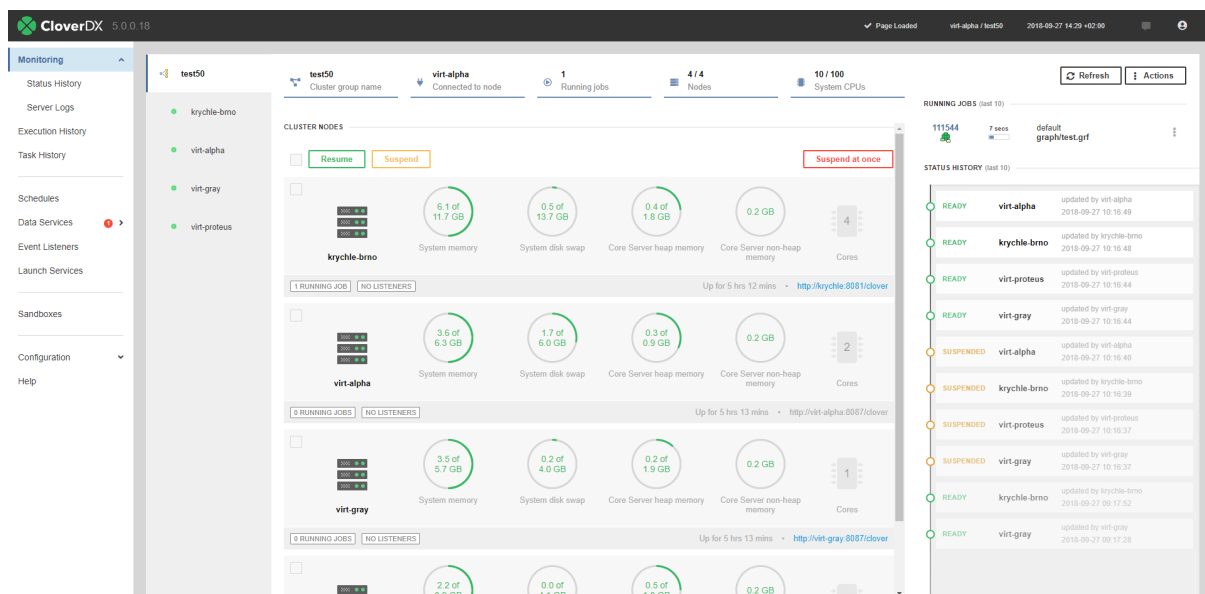
Figure 18.15. Cluster overview

Node Detail

Node Detail is similar to the [Standalone Server Detail](#) (p. 107) mentioned above, however it displays detail info about node selected in the menu on the left.
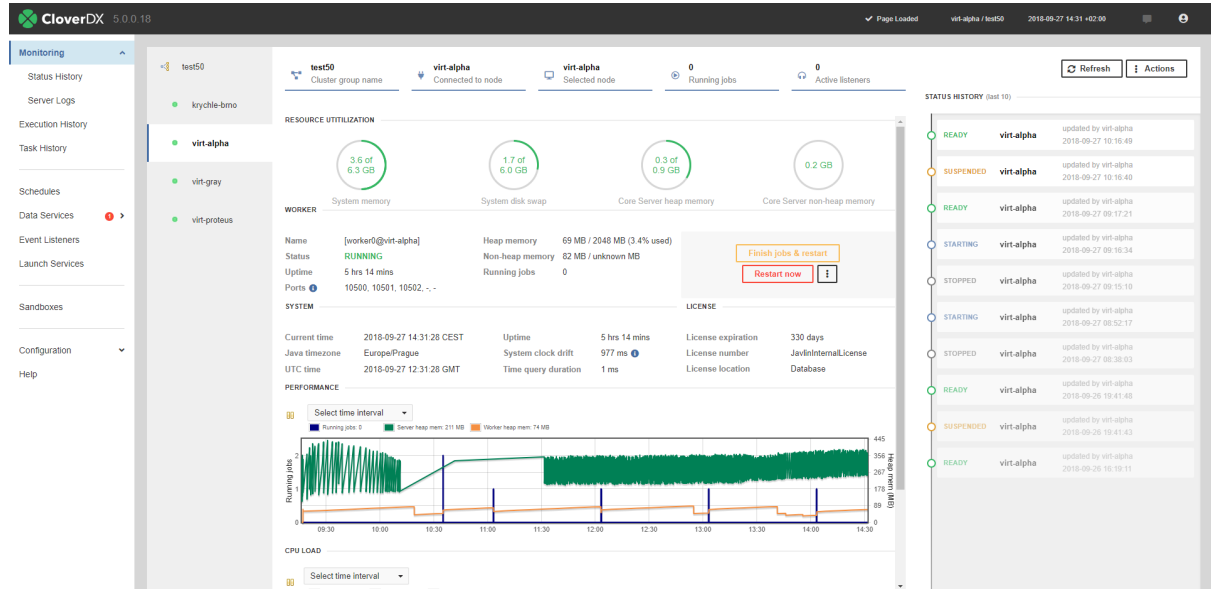
Figure 18.16. Node detail

Server Logs

Server Logs tab allows user to investigate log messages logged on other cluster nodes. Since the log messages are collected in memory, the maximum number of collected messages is relatively low by default, however it's customizable.

There are different "Log types":

- COMMON - Ordinary server logs as stored in log files. It contains information on successful and unsuccessful logins, start and end of job execution etc.

- WORKER - Worker related log.

  The log file is `${java.io.tmpdir}/cloverlogs/worker-[node_name].log`.

- CLUSTER - Only cluster - related messages are visible in this log. It contains information on job delegation and other types of messages related to cluster communication.

- LAUNCH_SERVICES - Only requests for launch services (deprecated)

- AUDIT - Detail logging of operations called on the **CloverDX Server** Core. Since the full logging may affect server performance, it's disabled by default. See Server Audit Logs (p. 104) for details

  The log file is `${java.io.tmpdir}/cloverlogs/user-action.log`.

- USER_ACTION - Contains some of user operations, e.g. login, logout, user creation, job execution, file synchronization (upload to server)

  The corresponding log file is `${java.io.tmpdir}/cloverlogs/user-action.log`.

- SERVER_INTERACTION - Interaction between Designer and Server.

  The log file is `${java.io.tmpdir}/cloverlogs/server-integration.log`.
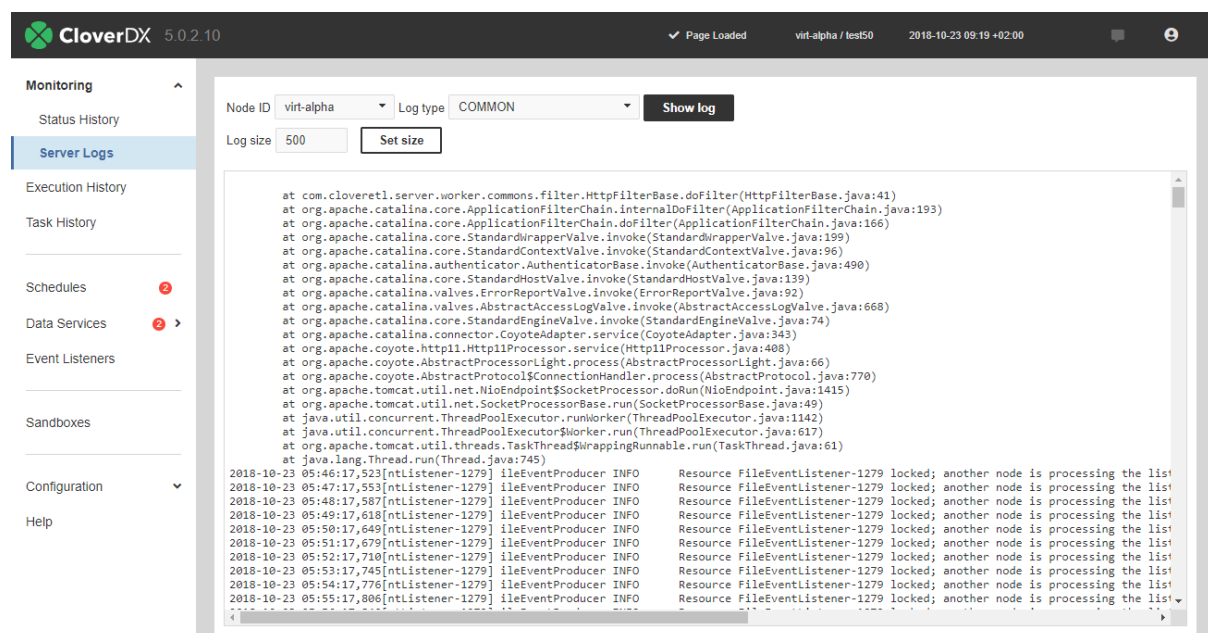
Figure 18.17. Server Logs

See also: Chapter 17, Logging (p. 103).

Using the Monitoring

Restarting the Worker

Switch to **Monitoring** > **Status** tab.

In the worker tile, click **Actions** button and select **Finish jobs and restart** from the menu.

The worker will be restarted.

See also .

Showing Worker's Command Line Arguments

Switch to **Monitoring** > **Status** tab.

In the worker tile, click the **Actions** button and select **Show command line**.

The command line arguments and parameters will be displayed.

See also .

Suspending the Server

Switch to **Monitoring** > **Status** tab.

Click the **Actions** button (in the upper left corner) and select **Suspend** from the menu.

The server will be suspended. If there is a job running, the node will wait until it finishes.

If you need to suspend the node immediately without waiting for jobs to finish, use **Suspend at once** instead of **Suspend**.

Resuming the Server

Switch to **Monitoring** > **Status** tab.

Click the **Actions** button (in the upper right corner) and select **Resume** from the menu.

The server will be resumed.

Displaying List of Threads of the Server Core

Switch to **Monitoring** > **Status** tab.

Click the **Actions** button (in the upper right corner) and select **Show Details**. Several new tiles will appear. Search for the **Threads** tile.

Chapter 19. Temp Space Management

Many of the components available in the **CloverDX Server** require temporary files or directories in order to work correctly. *Temp space* is a physical location on the file system where these files or directories are created and maintained. **CloverDX Server** allows you to configure and manage temp spaces - you can specify their locations, see usage of the filesystem etc.

To access this administration section, you need Temp Space Management permission (p. 140).

Overview

The overview of temp spaces defined in **CloverDX Server** is available under *Configuration > Temp Spaces*.

The overview panel displays list of temp spaces for each node in the Cluster. These properties are displayed for each temp space:

- **Node** - name of the node on which the temp space is located (only in Clustered environment)

- **Root path** - location of the temp space with unresolved placeholders (see note below for placeholders)

- **Resolved path** - location of the temp space with resolved placeholders (see note below for placeholders)

- **Free space** - remaining space for the temp space

- **File system size** - all available space for the temp space (actual size of the filesystem where the temp space resides)

- **File system usage** - size of used space in percentage

- **Available** - the directory exists and is writable

- **Status** - current status of temp space, can be `Active` or `Disabled`

> **Note**
>
> It is possible to use system properties and environment variables as placeholders. See Using environment variables and system properties (p. 117).

Node	Root path	Resolved path	Free space	File system size	File system usage	Available	Status	
virt-alpha	${java.io.tmpdir}/clover_temp_virt-alpha	/opt/apache-tomcat-9.0.12/temp/clover_temp_virt-alpha	9.8 GB	44.7 GB	78%	✓	Active	⋮
virt-gray	${java.io.tmpdir}/clover_temp_virt-gray	/opt/tomcat9/temp/clover_temp_virt-gray	19.9 GB	31.4 GB	36%	✓	Active	⋮
virt-proteus	${java.io.tmpdir}/clover_temp_virt-proteus	/opt/clover-geo-cluster/tempspace/clover_temp_virt-proteus	13.3 GB	24.9 GB	46%	✓	Active	⋮
krychle-brno	${java.io.tmpdir}/clover_temp_krychle-brno	/opt/clover-geo-cluster/tempspace/clover_temp_krychle-brno	287.7 GB	299.9 GB	4%	✓	Active	⋮

Figure 19.1. Configured temp spaces overview - one default temp space on each cluster node

Management

Temp space management offers an interface with options to add, disable, enable and delete a temp space.

Initialization

When **CloverDX Server** is starting the system checks temp space configuration: in case no temp space is configured a new default temp space is created in the directory where `java.io.tmpdir` system property points. The directory is named as follows:

- `${java.io.tmpdir}/clover_temp` in case of a standalone Server

- `${java.io.tmpdir}/clover_temp_<node_id>` in case of Server Cluster

Adding Temp Space

In order to define new temp space, click the **New Temp Space** button and specify its path. In case of Cluster environment, specify the node on which the new temp space should be created. If the directory entered does not exist, it will be created.

> ### Tip
> The main point of adding additional temp spaces is to enable higher system throughput - therefore the paths entered should point to directories residing on different physical devices to achieve maximal I/O performance.

Using environment variables and system properties

Environment variables and system properties can be used in the temp space path as a placeholder; they can be arbitrarily combined and resolved paths for each node may differ in accord with its configuration.

> ### Note
> The environment variables have higher priority than system properties of the same name. The path with variables are resolved after system has added new temp space and when the Server is starting. In case the variable value has been changed, it is necessary to restart the Server so that the change takes effect.

Examples:

- Given that an environment variable `USERNAME` has a value `cloverdxUser.` and is used as a placeholder in the path `C:\Users\${USERNAME}\tmp`, the resolved path is `C:\Users\cloverdxUser\tmp`.

- Given that Java system property `java.io.tmpdir` has a value `C:\Users\cloverdxUser\AppData\Local\Temp` and the property is used as a placeholder in the path `${java.io.tmpdir}\temp_folder`, the resolved path is `C:\Users\cloverdxUser\AppData\Local\Temp\temp_folder`.

- Node `node01` has been started with `-Dcustom.temporary.dir=C:\tmp_node01` parameter. Node `node02` has been started with `-Dcustom.temporary.dir=C:\tmp_node02` parameter. The declared

path is `${custom.temporary.dir}`. The resolved path is different for each node, `C:\tmp_node01` for `node01` and `C:\tmp_node02` for `node02`.

- When the declared path is `${java.io.tmpdir}\${USERNAME}\tmp_folder`, the resolved path is `C:\tmp\cloverdxUser\tmp_folder`.

Disabling Temp Space

To disable a temp space, click on the three vertical dots menu on the right side of the respective temp space and select **Disable**. Once the temp space has been disabled, no new temporary files will be created in it, but the files already created may be still used by running jobs. In case there are files left from previous or current job executions a notification is displayed.

> **Note**
>
> The system ensures that at least one enabled temp space is available.

Enabling Temp Space

To enable a temp space, click on the three vertical dots menu on the right side of the respective disabled temp space and select **Enable**. Enabled temp space is active, i.e. available for temporary files and directories creation.

Removing Temp Space

To remove a temp space, click on the three vertical dots menu on the right side of the respective temp space and select **Delete**. Only disabled temp spaces may be removed. If there are any running jobs using the temp space, the system will not allow its removal.

Chapter 20. Secure Parameters
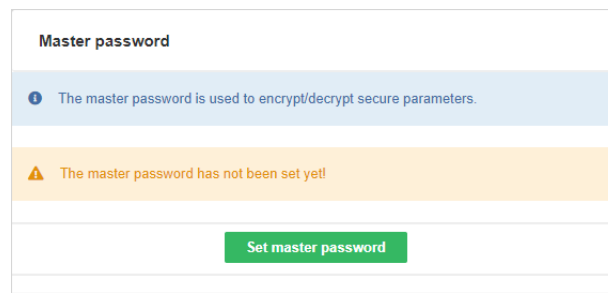
Transformation graphs in **CloverDX Server** environment allow you to define secure graph parameters. Secure graph parameters are regular graph parameters, either internal or external (in a `*.prm` file), but the values of the graph parameters are not stored in plain text on the file system - encrypted values are persisted instead. This allows you to use graph parameters to handle sensitive information, typically credentials such as passwords to databases.

Secure parameters are only available in **CloverDX Server** environment, including working with **CloverDX Server** Projects in **CloverDX Designer**.
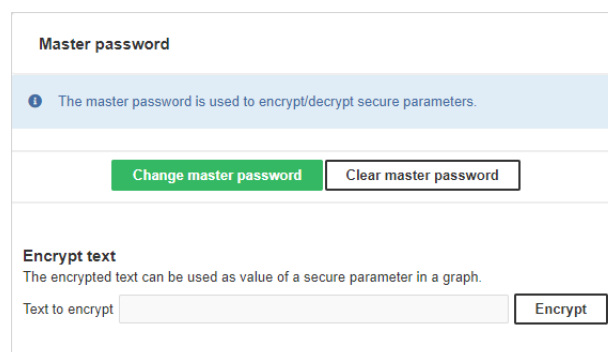
The encryption algorithm must be initialized with a **master password**. The master password has to be manually set after server installation in *Configuration > Security > Master password*. Secure parameters cannot be used before the master password is set.

The maximum length of the master password is 255 characters; there are no other restrictions or complexity requirements.

Figure 20.1. Master password initialization

After setting the master password secure parameters are fully available in **Graph parameter editor** in **CloverDX Designer**. When setting value of a secure parameter, it will be automatically encrypted using the master password. Secure parameters are automatically decrypted by server in graph runtime. A parameter value can also be encrypted in the **CloverDX Server** Console in the *Configuration > Security > Secure Parameters* page - use the **Encrypt text** section.

Figure 20.2. Graph parameters tab with initialized master password

If you change the master password, the secure parameters encrypted using the old master password cannot be decrypted correctly anymore. In that case existing secure parameters need to be encrypted again with the new master password. That can be accomplished simply by setting their value (non-encrypted) again in the **Graph parameter editor**. Similar master password inconsistency issue can occur if you move a transformation graph with some secure parameters to another server with a different master password. So it is highly recommended to use the identical master password for all your **CloverDX Server** installations.

See documentation of secure parameters in **CloverDX Designer** manual for further details.

Secure parameters configuration

Encryption of secure parameters can be further customized via server configuration parameters.

Table 20.1. Secure parameters configuration parameters

Property name	Default value	Description
security.job_parameters.encryptor.algorithm	PBEWithMD5AndDES	The algorithm to be used for encryption. This algorithm has to be supported by your JCE provider (if you specify a custom one, or the default JVM provider if you don't). The name of algorithm should start with *PBE* prefix. The list of available algorithms depends on your JCE provider, e.g. for the default *SunJCE* provider you can find them on SunJCEProvider or for the *Bouncy Castle* provider on Bouncy Castle Specifications (section *Algorithms/PBE*)).
security.job_parameters.encryptor.master_password_encryption.password	clover	The password used to encrypt values persisted in the database table *secure_param_passwd* (the master password is persisted there).
security.job_parameters.encryptor.providerClassName	Empty string. The default JVM provider is used (e.g. for Oracle Java the SunJCE provider is used)	The name of the security provider to be asked for the encryption algorithm. It must implement *java.security.Provider* interface. For example set to *org.bouncycastle.jce.provider.BouncyCastleProvider* for the *Bouncy Castle* JCE provider, see below.

Installing Bouncy Castle JCE provider

Algorithms provided by JVM could be too weak to satisfy an adequate security. Therefore it is recommended to install a third-party JCE provider. Following example demonstrates installation of one concrete provider, *Bouncy Castle* JCE provider. Another provider would be installed similarly.

1. Download Bouncy Castle provider jar (e.g. bcprov-jdk15on-150.jar) from http://bouncycastle.org/ latest_releases.html

2. Add the jar to the classpath of your application container running **CloverDX Server**, e.g. to directory `WEB-INF/lib`

3. Set value of the *security.job_parameters.encryptor.providerClassName* attribute to *org.bouncycastle.jce.provider.BouncyCastleProvider* in the `config.properties` file.

4. Set value of the *security.job_parameters.encryptor.algorithm* attribute to the desired algorithm (e.g. *PBEWITHSHA256AND256BITAES-CBC-BC*).

Example of configuration using Bouncy Castle:

```
security.job_parameters.encryptor.algorithm=PBEWITHSHA256AND256BITAES-CBC-BC
security.job_parameters.encryptor.providerClassName=org.bouncycastle.jce.provider.BouncyCastleProvider
```

Chapter 21. Users and Groups

The **CloverDX Server** has a built-in security module that manages users and groups. User groups control access permissions to sandboxes and operations the users can perform on the Server, including authenticated calls to Server API functions. A single user can belong to multiple groups.

LDAP or Active Directory can be configured with the Server to authenticate users and optionally assign their effective groups (and permissions) from a global directory.

You can manage users and user groups in **Configuration > Users and Groups**. Please note that you need a **List users** and **List groups** permissions, respectively.

LDAP Authentication

Since **CloverETL 3.2**, it is possible to configure the Server to use an LDAP server for users authentication. This way, the credentials of users registered in LDAP may be used for authentication to any **CloverDX Server** interface (API or web console).

However, the authorization (access levels to sandboxes content and privileges for operations) is still handled by the **CloverDX** security module. Each user, even when logged-in using LDAP authentication, must have their own "user" record (with related groups) in the **CloverDX** security module. So there must be a user with the same username and domain set to "LDAP". Such a record has to be created by a Server administrator before the user can log in.

To configure LDAP authentication, use the Setup GUI (p. 60).

Use LDAP for user authentication only

If this **Authentication Policy** is selected, **CloverDX Server** uses LDAP directory to verify only the user's password:

1. The user specifies the LDAP credentials in the login form to the Server web console.

2. **CloverDX Server** looks up the user's record and checks whether it has the "LDAP" domain set.

3. The Server attempts to connect to the LDAP server using the user's credentials. If it succeeds, the user is logged in.

Use LDAP for user authentication and user synchronization

In this mode, **CloverDX Server** verifies user's credentials and synchronizes additional information (group, name and email) with those stored in LDAP.

In case the Server is configured for user authentication and group synchronization, the procedure is as follows:

1. The user specifies the LDAP credentials in the login form to the Server web console.

2. **CloverDX Server** looks up the user's record and checks whether it has the "LDAP" domain set.

3. **CloverDX Server** connects to the LDAP server and checks whether the user exists (it uses specified search to lookup in LDAP).

4. If the user exists in LDAP, **CloverDX Server** performs authentication.

5. If the authentication is successful, **CloverDX Server** searches LDAP for user's groups.

6. **CloverDX** user is assigned to the **CloverDX** groups according to his current assignation to the LDAP groups.

7. User is logged-in.

> ## Note
>
> Switching domains:
>
> - If a user was **created as LDAP** and then switched to clover domain, you have to **set a password** for him in the **Change password** tab.

- If a user was **created as clover** and then switched to LDAP domain, they have a password in clover domain, but it is overridden by the LDAP password. After switching back to clover domain, the **original password is re-used**. It can be reset in the **Change password** tab, if needed.

Configuration

By default **CloverDX Server** allows only its own internal mechanism for authentication. To enable authentication with LDAP, set the configuration property "security.authentication.allowed_domains" properly. It is a list of user domains that are used for authentication.

Currently there are 2 authentication mechanism implemented: "LDAP" and "clover" ("clover" is an identifier of **CloverDX** internal authentication and may be changed by security.default_domain property, but only for white-labeling purposes). To enable LDAP authentication, set value to "LDAP" (only LDAP) or "clover,LDAP". Users from both domain may login. It's recommended to allow both mechanisms together, until the LDAP is properly configured. So the admin user can still login to web GUI although the LDAP connection isn't properly configured.

You can use **Setup** to configure LDAP authentication. See [LDAP](#) (p. 60) in Chapter 13, [Setup](#) (p. 54).

Basic LDAP connection properties

```
# Implementation of context factory.
security.ldap.ctx_factory=com.sun.jndi.ldap.LdapCtxFactory
# URL of LDAP server.
security.ldap.url=ldap://hostname:port
# User DN pattern that will be used to create LDAP user DN from login name.
security.ldap.user_dn_pattern=uid=${username},dc=company,dc=com
```

Depending on the LDAP server configuration the property `security.ldap.user_dn_pattern` can be pattern for user's actual distinguished name in the LDAP directory, or just the login name - in such case just set the property to `${username}`.

Configuration of user and group lookup

In order to be able to synchronize the **Clover** groups with those defined in LDAP directory, the `security.ldap.user_dn_pattern` has to be left unspecified. There are additional properties required so that the server is able to search the LDAP directory.

```
# User DN of a user that has sufficient privileges to search LDAP for users and groups.
security.ldap.userDN=cn=Manager,dc=company,dc=com
# The password for user mentioned above.
security.ldap.password=
```

There are optional settings affecting how the LDAP directory is searched.

```
# Timeout for queries searching the directory.
security.ldap.timeout=5000
# Maximal number of records that the query can return.
security.ldap.records_limit=2000
# How LDAP referrals are processed, possible values are: 'follow', 'ignore' and 'throw'.
# The default depends on the context provider.
security.ldap.referral=
```

Specified values work for this specific LDAP tree:

- dc=company,dc=com
  - ou=groups

- cn=admins
  (objectClass=groupOfNames,member=(uid=smith,dc=company,dc=com),member=(uid=jones,dc=company,dc=com))
- cn=developers (objectClass=groupOfNames,member=(uid=smith,dc=company,dc=com))
- cn=consultants (objectClass=groupOfNames,member=(uid=jones,dc=company,dc=com))
- ou=people
  - uid=smith (fn=John,sn=Smith,mail=smith@company.com)
  - uid=jones (fn=Bob,sn=Jones,mail=jones@company.com)

Following properties are necessary for lookup for the LDAP user by his username. (step [4] in the login process above)

```
# Base specifies the node of LDAP tree where the search starts.
security.ldap.user_search.base=dc=company,dc=eu
# Filter expression for searching the user by his username.
# Note, that this search query must return just one record.
# Placeholder ${username} will be replaced by username specified by the logging user.
security.ldap.user_search.filter=(uid=${username})
# Scope specifies type of search in "base". There are three possible values: SUBTREE | ONELEVEL | OBJECT
# http://download.oracle.com/javase/8/docs/api/javax/naming/directory/SearchControls.html
security.ldap.user_search.scope=SUBTREE
```

Following properties are names of attributes from the search defined above. They are used for getting basic info about the LDAP user in case the user record has to be created/updated by **CloverDX** security module: (step [6] in the login process above)

```
security.ldap.user_search.attribute.firstname=fn
security.ldap.user_search.attribute.lastname=sn
security.ldap.user_search.attribute.email=mail
# This property is related to the following step "searching for groups".
# Groups may be obtained from specified user's attribute, or found by filter (see next paragraph).
# Leave this property empty if the user doesn't have such attribute.
security.ldap.user_search.attribute.groups=memberOf
```

In the following step, **CloverDX** tries to find groups which the user is assigned to. (step [4] in the login process above). There are two ways how to get list of groups which the user is assigned to. The user-groups relation is specified on the "user" side. The user record has some attribute with list of groups. It's "memberOf" attribute usually. Or the relation is specified on the "group" side. The group record has an attribute with list of assigned users. It's "member" attribute usually.

In case the relation is specified on users side, please specify property:

```
security.ldap.user_search.attribute.groups=memberOf
```

Leave it empty otherwise.

In case the relation is specified on the groups side, set properties for searching:

```
security.ldap.groups_search.base=dc=company,dc=com
# Placeholder ${userDN} will be replaced by user DN found by the search above.
# If the filter is empty, searching will be skipped.
security.ldap.groups_search.filter=(&(objectClass=groupOfNames)(member=${userDN}))
security.ldap.groups_search.scope=SUBTREE
```

Otherwise, please leave property security.ldap.groups_search.filter empty, so the search will be skipped.

Clover user record will be assigned to the clover groups according to the LDAP groups found by the search (or the attribute). (Groups synchronization is performed during each login)

```
# Value of the following attribute will be used for lookup for the CloverDX group by its code.
```

```
# So the user will be assigned to the CloverDX group with the same "code".
security.ldap.groups_search.attribute.group_code=cn
```

Users

This section is intended to users management. It offers features in dependence on user's permissions (i.e. a user may enter this section, but cannot modify anything, or they may modify, but cannot create new users).

The **Users** section of the **Configuration** menu allows you to:

After default installation on an empty database, the admin user is created automatically.

Table 21.1. Admin user

User name	Description
clover	Clover user has admin permissions, thus default password `clover` should be changed after installation.

Create New User

When creating a new User, you must enter the following information:

Table 21.2. User attributes

Attribute	Description
Domain	Domain which is the origin of the user.r Currently, there are only two possible values: "clover" or "ldap".
Username	A common user identifier. Must be unique, cannot contain spaces or special characters, just letters and numbers.
First name	The user's first name.
Last name	The user's last name.
E-mail	Email address which may be used by CloverDX administrator or by CloverDX Server for automatic notifications. See Send an Email (p. 172) for details.
Password	Case sensitive password. If the user looses his password, the new one must be set. The password is stored in an encrypted form for security reasons, so it cannot be retrieved from a database and must be changed by the user who has proper permission for such operation.
Verify password	Verify the entered password.

Edit Users Record

A user with a **Create user** or **Edit user** permission can use this form to set basic user parameters.

Change Users Password

If user looses his password, the new one must be set. So a user with the **Change passwords** permission can use this form to do it.

Assign Users to Groups

Assignment to groups gives the user proper permissions. Only logged user with the **Groups assignment** permission can access this form and specify groups which the user is assigned in. For details about permissions, see Groups (p. 130).

Disable / Enable Users

Since a user record has various relations to the logs and history records, it can't be deleted. So it is disabled instead. This means that the record doesn't display in the list and the user can't login.

However, a disabled user may be enabled again. **Note** that the disabled user is removed from their groups, so groups should be assigned properly after re-enabling.

Groups

Group is an abstract set of users, which gives assigned users some permissions. So it is not necessary to specify permissions for each single user.

There are independent levels of permissions implemented in **CloverDX Server**

- *permissions to Read/Write/eXecute in sandboxes* - The sandbox owner can specify different permissions for different groups. For details, see Sandbox Content Security and Permissions (p. 147).
- *permissions to perform some operation* - user with an operation permission **Permission assignment** may assign specific permission to existing groups.
- *permissions to launch specific service* - For details, see Chapter 39, Launch Services (p. 249), Chapter 40, Data Services (p. 256).

Table 21.3. Default groups created during installation

Group name	Description
admins	This group has an operation permission **all** assigned, which means, that it has unlimited permission. Default user **clover** is assigned to this group, which makes him administrator.
all users	By default, every single **CloverDX** user is assigned to this group. It is possible to remove a user from this group, but it is not a recommended approach. This group is useful for some permissions to sandbox or some operation, which you would like to make accessible for all users without exceptions.

Users Assignment

Relation between users and groups is N:M. Thus in the same way, how groups are assignable to users, users are assignable to groups.

Groups permissions

Groups permissions are structured as a tree, where permissions are inherited from the root to leafs. Thus if some permission (tree node) is enabled (blue dot), all permissions in sub tree are automatically enabled (white dot). Permissions with red cross are disabled.

Thus for the **admin** group just the **all** permission is assigned, every single permission in the sub tree is assigned automatically.

With none of the following privileges, a user can: log into the Server console, create a server project (in Designer) from its own sandbox, create a file in its own existing sandbox, and run graphs.

- **all**

  The user with this permission has all available permissions. The **Admin** group has all permissions by default.

  - **Unlimited access to sandboxes**

    Allows the user to perform operations on all sandboxes, even if the sandbox accessibility is not specified explicitly.

    This permission does not include the suspend sandbox permission (p. 138).

  - **Sandboxes**

    Allows the user to work with sandboxes. This permission contains all the permissions below. The user can perform operations only on sandboxes owned by himself or on sandboxes with explicitly added access to him, see Chapter 22, Sandboxes - Server Side Job Files (p. 144).

- **List sandbox**

  In the Server web interface, it allows the user to list their sandboxes and sandboxes with **read** permission granted to the user's group.

  In the Server web interface, this permission is necessary to create, edit, or delete sandboxes.

  Within a sandbox with the **write** access granted, the user can edit or remove files and create or delete directories even without this permission.

- **Create sandbox**

  Allows the user to create new sandboxes.

  If a sandbox is to be created in web interface, the user must have the list sandbox permission (p. 131).

- **Delete sandbox**

  Allows the user to delete sandboxes.

  If a sandbox is to be deleted in web interface, the user must have the list sandbox permission (p. 131).

- **Edit sandbox**

  Allows the user to edit sandboxes.

  If a sandbox is to be modified in web interface, the user must have the list sandbox permission (p. 131).

- **May delete files missing in uploaded ZIP**

  In **Sandbox →Upload ZIP**, it allows the user to use a checkbox to delete files missing in the ZIP to be uploaded. If the user does not have this permission, the checkbox to delete mission files in ZIP is not displayed.

  If a sandbox is to be uploaded from a ZIP file in the Server web interface, the user must have the list sandbox permission (p. 131).

- **Scheduling**

  Allows the user to manage schedules, see Chapter 30, Scheduling (p. 192).

  - **List schedule**

    Allows the user to list all schedules.

    - **List schedule limited**

      Allows the user to list the enabled schedules.

  - **Create schedule**

    Allows the user to create new schedules.

    The user must have the list schedule limited permission (p. 131) to access the scheduling section to create a new schedule.

  - **Delete schedule**

    Allows the user to delete schedules.

The user must have the list schedule limited permission(p. 131)or list schedule permission(p. 131) to access the scheduling section to delete the schedule.

- **Edit schedule**

  Allows the user to edit schedules.

  The user must have the list schedule limited permission(p. 131)or list schedule permission(p. 131) to access the scheduling section to edit the schedule.

- **Event listeners**

  Allows the user to manage event listeners, see Chapter 32, Listeners (p. 203).

  - **List of Event Listeners**

    Allows the user to list all event listeners.

    - **List of Jobflow Event Listeners unlimited**

      Allows the user to list jobflow event listeners.

      See Jobflow Event Listeners (p. 211)

      - **List of Jobflow Event Listeners limited**

        Allows the user to list jobflow event listeners of sandboxes the user can read from.

    - **List of Graph Event Listeners unlimited**

      Allows the user to list all graph event listeners, see Graph Event Listeners (p. 205).

      - **List of Graph Event Listeners limited**

        Allows the user to list graph event listeners from sandboxes the user can read from.

    - **List of File Event Listeners unlimited**

      Allows the user to list all file event listeners, see File Event Listeners (remote and local) (p. 220).

      - **List of File Event Listeners limited**

        Allows the user to list all file event listeners.

    - **List of JMS Event Listeners unlimited**

      Allows the user to list all JMS listeners, see JMS Message Listeners (p. 213).

      - **List of JMS Event Listeners limited**

        Allows the user to list all JMS listeners.

    - **List of Universal Event Listeners unlimited**

      Allows the user to list all universal event listeners, see Universal Event Listeners (p. 218).

      - **List of Universal Event Listeners limited**

        Allows the user to list all universal event listeners.

        See Universal Event Listeners (p. 218).

- **List of Task Event Listeners unlimited**

  Allows the user to list all task event listeners, see [Task Failure Listeners](#) (p. 228).

  - **List of Task Event Listeners limited**

    Allows the user to list all task event listeners from sandboxes the user can read from.

    See [Task Failure Listeners](#) (p. 228).

- **Create Event Listener**

  Allows the user to create event listeners.

  If an event listener is to be created in the Server web interface, the user must have permission to list the event listeners of the particular type.

  - **Create Jobflow Event Listener**

    Allows the user to create new Jobflow Event listeners.

    If a Jobflow event listener is to be created in the Server web interface, the user must have the [list of jobflow event listeners limited permission](#) (p. 132).

    See [Jobflow Event Listeners](#) (p. 211).

  - **Create Graph Event Listener**

    Allows the user to create graph event listeners.

    If a graph event listener is to be created in the Server web interface, the user must have the [list of graph event listeners limited permission](#) (p. 132).

    See [Graph Event Listeners](#) (p. 205).

  - **Create File Event Listener**

    Allows the user to create graph event listeners.

    If a file event listener is to be created in the Server web interface, the user must have the [list of file event listeners limited permission](#) (p. 132).

    See [File Event Listeners (remote and local)](#) (p. 220).

  - **Create JMS Listener**

    Allows the user to create JMS event listeners.

    If a JMS event listener is to be created in the Server web interface, the user must have the [list of JMS event listeners limited permission](#) (p. 132).

    See [JMS Message Listeners](#) (p. 213).

  - **Create Universal Event Listener**

    Allows the user to create universal event listeners.

    If a universal event listener is to be created in the Server web interface, the user must have the [list of universal event listeners limited permission](#) (p. 132).

    See [Universal Event Listeners](#) (p. 218).

- **Create Task Event Listener**

  Allows the user to create task event listeners.

  If a task event listener is to be created in the Server web interface, the user must have the <u>list of task event listeners limited permission</u> (p. 133).

  See <u>Task Failure Listeners</u> (p. 228).

- **Edit Event Listener**

  Allows the user to edit event listeners.

  If an event listener is to be created in the Server web interface, the user must have permission to list event listener of the particular type.

  - **Edit Jobflow Event Listener**

    Allows the user to edit jobflow event listeners.

    If a jobflow event listener is to be edited in the Server web interface, the user must have the <u>list of jobflow event listeners limited permission</u> (p. 132).

    See <u>Jobflow Event Listeners</u> (p. 211).

  - **Edit Graph Event Listener**

    Allows the user to edit graph event listeners.

    If a graph event listener is to be edited in the Server web interface, the user must have the <u>list of graph event listeners limited permission</u> (p. 132).

    See <u>Graph Event Listeners</u> (p. 205).

  - **Edit File Event Listener**

    Allows the user to edit file event listeners.

    If a file event listener is to be edited in the Server web interface, the user must have the <u>list of file event listeners limited permission</u> (p. 132).

    See <u>File Event Listeners (remote and local)</u> (p. 220).

  - **Edit JMS Event Listener**

    Allows the user to edit JMS event listeners.

    If a JMS event listener is to be edited in the Server web interface, the user must have the <u>list of JMS event listeners limited permission</u> (p. 132).

  - **Edit Universal Event Listener**

    Allows the user to edit universal event listeners.

    If a universal event listener is to be edited in the Server web interface, user must have permission <u>list of universal event listeners limited permission</u> (p. 132).

    See <u>Universal Event Listeners</u> (p. 218).

  - **Edit Task Event Listener**

Allows the user to edit task event listeners.

If a task event listener is to be edited in the Server web interface, user must have permission <u>list of task event listeners limited permission</u> (p. 133).

See <u>Task Failure Listeners</u> (p. 228).

- **Delete Event Listener**

Allows the user to delete event listeners.

- **Delete Jobflow Event Listener**

Allows the user to delete jobflow event listeners.

The user must have the <u>delete graph event listener permission</u> (p. 135) to delete a jobflow event listener.

It a jobflow event listener is to be deleted in the Server web interface, the user must have the <u>list of jobflow event listeners limited permission</u> (p. 132)

- **Delete Graph Event Listener**

Allows the user to delete graph event listeners.

If a graph event listener is to be deleted in the Server web interface, the user must have the <u>list of graph event listeners limited permission</u> (p. 132).

See <u>Graph Event Listeners</u> (p. 205).

- **Delete File Event Listener**

Allows the user to delete file event listeners.

The user must have the <u>delete graph event listener permission</u> (p. 135) to delete a file event listener.

If a file event listener is to be deleted in the Server web interface, the user must have the <u>list of file event listeners limited permission</u> (p. 132).

See <u>File Event Listeners (remote and local)</u> (p. 220).

- **Delete JMS Event Listener**

Allows the user to delete JMS Event Listeners.

The user must have the <u>delete graph event listener permission</u> (p. 135) to delete a JMS event listener.

If a graph event listener is to be deleted in the Server web interface, the user must have the <u>list of JMS event listeners limited permission</u> (p. 132).

- **Delete Universal Event Listener**

Allows the user to delete universal event listeners.

The user must have the <u>delete graph event listener permission</u> (p. 135) to delete universal event listener.

If a universal event listener is to be deleted in the Server web interface, the user must have the <u>list of universal event listeners limited permission</u> (p. 132).

See <u>Universal Event Listeners</u> (p. 218).

- **Delete Task Event Listener**

  Allows the user to delete task event listeners.

  If a task event listener is to be deleted in the Server web interface, the user must have the list of task event listeners limited permission (p. 133).

  See Task Failure Listeners (p. 228).

- **Manual task Execution**

  Allows the user to manually execute a task (send an email, execute a script, etc.) with an immediate effect.

  See Chapter 29, Manual Task Execution (p. 191).

- **Unlimited access to execution history**

  Allows the user to perform the same operations as unlimited access to execution history list permission (p. 136).

  - **Unlimited access to execution history list**

    Allows the user to view execution history of all jobs.

    - **Limited access to execution history list**

      Allows the user to view execution history of jobs from sandboxes the user can read from. In Designer, this permission is required to be able to view **Execution log** in Designer's console and execution history in **Execution** tab.

- **Launch Services**

  Allows the user to list, create, edit, and delete launch services, see Chapter 39, Launch Services (p. 249).

  - **List Launch Services unlimited**

    Allows the user to list all launch services.

    - **List Launch Services Limited**

      Allows the user to list launch services from sandboxes the user can read from.

  - **Create Launch service**

    Allows the user to create a new launch service.

    User has to have the create graph event listener permission (p. 133) to bind the launch service with a graph.

    If the launch service is to be created in the Server web interface, the user has to have the list launch services limited permission (p. 136) (or the list launch services unlimited permission (p. 136) to access the section with launch services.

  - **Delete Launch Service**

    Allows the user to delete a launch service.

    User has to have delete graph event listener permission (p. 135) to delete a launch service.

    If the launch service is to be deleted in the Server web interface, the user must have the list launch services limited permission (p. 136) to access the section with launch services.

- **Edit Launch Service**

  Allows the user to edit a launch services.

  User has to have edit graph event listener (p. 134) to edit the launch service.

  If the launch service is to be edited in the Server web interface, the user must have the list launch services limited permission (p. 136) to choose the launch service in the server interface.

- **Data service**

  Allows the user to access the **Data service** section, see Chapter 40, Data Services (p. 256).

  - **List data services**

    Allows the user to list data services.

  - **Manage data services**

    Allows the user to manage data services.

  - **Delete data services**

    Allows the user to delete data services.

  - **Execute and access documentation**

    Allows the user to execute and access documentation.

  - **Manage HTTPS connectors**

    Allows the user to manage HTTPS connectors.

- **Tasks history**

  Allows the user to access the **Tasks history** section, see Chapter 28, Tasks (p. 171).

- **Monitoring**

  Monitoring permission grants user all its subpermissions.

  - **Monitoring section**

    Allows the user to access the monitoring section.

    See Chapter 18, Monitoring (p. 107).

- **Suspend**

  Allows the user to suspend the server, a cluster node, or a sandbox.

  The user must have the monitoring section permission (p. 137) to access the Monitoring section.

  - **Suspend server**

    Allows the user to suspend or resume the server.

    The user must have the monitoring section permission (p. 137) to access the monitoring section.

  - **Suspend cluster node**

    Allows the user to suspend or resume a cluster node.

The user must have the <u>monitoring section permission</u> (p. 137) to access the monitoring section.

- **Suspend sandbox**

  Allows the user to suspend a sandbox. The user must have <u>list sandbox permission</u> (p. 131) to view the sandboxes to suspend them.

  See also Chapter 22, <u>Sandboxes - Server Side Job Files</u> (p. 144).

- **Reset caches**

  Deprecated.

- **Running jobs unlimited**

  If the graph is to be run from server web interface, the user must have the <u>list sandbox permission</u> (p. 131) to list the graphs.

  - **Running jobs limited**

    If the graph is to be run from server web interface, the user must have the <u>list sandbox permission</u> (p. 131) to list the graphs.

- **Configuration**

Allows the user to access the configuration section.

- **Users**

  This permission allow user to access the **Users** section and configure user accounts.

  - **List user**

    Allows the user to list users and access to the **Users** administration section (**Configuration** →**Users**)

  - **Change passwords**

    Allows the user to change his password and to change password of another user.

    To see list of users, the user needs the <u>list user permission</u> (p. 138).

  - **Edit user**

    Allows the user to change group assignment.

    To see the list of users, the user must have the <u>list user permission</u> (p. 138).

    - **Edit own profile and password**

      Allows the user to change his profile (first name, last name, email, and password).

      The user can access her profile in main web console view under username, in upper right corner of the page.

  - **Delete user**

    Allows the user to disable a user.

    The user must have the <u>list user permission</u> (p. 138) to list available users.

  - **Create user**

Allows the user to create a new user.

If the user is to be created in the Server web interface, the creating user must have the [list user permission](#) (p. 138) to list users to access this option.

- **Groups assignment**

Allows the user to assign users to groups.

The user must have the [edit user permission](#) (p. 138) to successfully finish the assignment of users to groups.

If the user is to be created in the Server web interface, the creating user must have the [list user permission](#) (p. 138) to list users to access this option.

- **Groups**

Allows the user to manage groups: user can list groups, create groups, delete groups, edit the group, assign users to the group, and change permissions of the group.

- **List groups**

Allows the user to list groups. This permission is necessary for use of other options from the **Groups** group.

- **Create group**

Allows the user to create a new user group.

If the user group is to be created in the Server web interface, the user must have the [list groups permission](#) (p. 139) to view a list of groups and to access this option.

- **Delete group**

Allows the user to delete a user group.

Only empty groups can be deleted. You need to have the [list groups permission](#) (p. 139) to view list of groups and to access this option.

- **Edit group**

This permission allow user to edit user groups.

This permission does not include **User assignment** and **Permission assignment**.

If the user group is to be edited from server web interface, the user must have the [list groups permission](#) (p. 139).

- **Users assignment**

Allows the user to assign users to groups.

The user needs [Edit group permission](#) (p. 139) to commit the changes in the assignment.

If the assignment is to be edited in the Server web interface, the user must have the [list groups permission](#) (p. 139) to list the groups.

- **Permission assignment**

Allows the user to configure group **Permissions**.

The user needs have the [Edit group permission](#) (p. 139) to commit the changes.

If the permissions are to be edited in the Server web interface, the user must have the [list groups permission](#) (p. 139) to list the groups.

- **Secure parameters administration**

  - **Secure params**

    Allows the user to change the value of a secure parameter.

    The user can use secure parameters in graphs even without this permission.

- **CloverDX/System info sections**

  Allows the user to view **System Info** and **CloverDX Info** sections.

- **CloverDX Server properties**

  Allows the user to view **Server Properties** tab and **Data Profiler properties** tab in **CloverDX Info** section.

  The user must have the [CloverDX/System info sections permission](#) (p. 140) to access **CloverDX Info** section.

- **Reload license**

  Allows the user to reload and view the server license.

  The user must have the [CloverDX/System info sections permission](#) (p. 140) to access the **Configuration** section.

- **Upload license**

  Allows the user to update the server license.

  The user must have the [CloverDX/System info sections permission](#) (p. 140) to access the **Configuration** section.

  See [Activation](#) (p. 32).

- **Server Configuration Management**

  Allows the user to import and export the server configuration.

  See Chapter 23, [Server Configuration Migration](#) (p. 155).

  - **Export Server Configuration**

    Allows the user to export the server configuration.

    See [Server Configuration Export](#) (p. 156).

  - **Import Server Configuration**

    Allows the user to import the server configuration.

    See [Server Configuration Import](#) (p. 157).

- **Temp Space Management**

  Allows the user to access **Temp Space Management** section.

  See Chapter 19, [Temp Space Management](#) (p. 116).

- **Server Setup**

  Allows the user to access the server setup.

  See Chapter 13, Setup (p. 54).

- **Heap Memory Dump**

  Allows the user to create a **Thread dump** and a **Heap Memory Dump**.

  See Chapter 25, Diagnostics (p. 162).

- **Groovy Code API**

  Allows the user to run Groovy scripts.

- **Open Profiler Reporting Console**

  Allows the user to login to the **Profiler reporting console**.

  The permission is necessary to view the results of **CloverDX** Profiling Jobs in Designer.

  Even without this permission, a user can create and run `.cpj` jobs from Designer.

User Lockout

CloverDX can lock out a user access after a set number of unsuccessful login attempts as a way of protecting against brute force attacks on users' credentials.

The lockout occurs only in **CloverDX**. For example, it will not affect LDAP in the case of LDAP user authentication. By default, the feature is disabled.

Information regarding user lockout is stored in the **USER_ACTION** server log. Notifications can be sent via email; however, it is necessary to set up a connection to an SMTP server in the E-mail (p. 59) tab of the **Setup GUI**.

The feature has several parameters which can be set by modifying the following lines in the configuration file; either directly or in the **Configuration File** tab of the Setup GUI (p. 54):

Table 21.4. User lockout parameters

Parameter	Description
security.lockout.login.attempts	Limits the number of login attempts of the user. The next failed login attempt will lock the user's access. When setting the value, keep in mind that **CloverDX Designer** with several server projects can attempt to log in multiple times. The recommended value is 50. Change the value to 0 to disable the feature.
security.lockout.reset.period	Represents the period (in seconds) during which failed login attempts are counted. If no such attempt occurs during this period, the counter of failed login attempts is reset to 0. This way, the user does not have to worry about accidentally locking himself out of the system after a certain number of failed login attempts over an extended period of time. The default value is 60 (1 minute). Change the value to 0 to set the period to infinity.
security.lockout.unlock.period	Represents the period (in seconds) after which a successful login attempt will unlock the previously locked user. After this period, the user is able to login using his credentials again without the need to have his account unlocked by the administrator. The parameter protects the system against denial of service (DoS) attacks and should be set to a reasonable value so you are not locked out of the system for too long in case the administrator's account is affected by the attack. The default value is 300 (5 minutes). Change the value to 0 to set the period to infinity.
security.lockout.notify.email	This parameter represents a comma separated list of emails of persons who should be notified when a user lockout occurs. Note that the locked out user receives the notification email automatically (if the server's SMTP is configured and they have provided their email address). This parameter should therefore be set, for example, to an administrators' mail group so they are aware of the situation.

The recommended, default values are set in such a way as to efficiently protect the system against brute force attacks, prevent complete lockout of the administrator access and not limit users in standard usage of **CloverDX Server**.

The properties can be set in the following section of the properties file:

```
## Uncomment lines bellow to enable user lockout after number of failed logins
## Number of failed login attempts after which a next failed login attempt will lock the user
## 0 means feature is switched off
## default suggested value is 50
#security.lockout.login.attempts=50
## Periods are specified in seconds
## Period of time during which the failed login attempts are counted
## Default is 60s (1 min)
```

#security.lockout.reset.period=60
Period of time after which a successful login attempt will unlock previously locked user
Default is 300s (5 min)
#security.lockout.unlock.period=300
Comma separated list of emails which will be notified when user is locked out.
#security.lockout.notification.email=

Unlocking User

Once the user's access is locked, you can see the status in the **Users** tab of the **Configuration** section.

To unlock the user, click on the **...** button in the respective row of the **Action** column and choose **Unlock**.

Chapter 22. Sandboxes - Server Side Job Files

A sandbox is a place where you store all your project's transformation graph files, jobflows, data, and other resources. It's a server side analogy to a Designer project. The Server adds additional features to sandboxes, like user permissions management and global per-sandbox configuration options.

The Server and the Designer are integrated so that you are able to connect to a Server sandbox using a **Server Project** in your Designer workspace. Such a project works like a remote file system – all data is stored on the Server and accessed remotely. Nonetheless, you can do everything with Server Projects the same way as with local projects – copy and paste files, create, edit, and debug graphs, etcetera. See the **CloverDX Designer** manual for details on configuring a connection to the Server.

Technically, a sandbox is a dedicated directory on the Server host file system and its contents are managed by the Server. Advanced types of sandboxes, like "partitioned sandbox" have multiple locations to allow distributed parallel processing (more about that in Chapter 41, Sandboxes in Cluster(p. 276)). A sandbox cannot contain another sandbox within – it's a single root path for a project.

It is recommended to put all sandboxes in a folder outside the **CloverDX Server** installation (by default the sandboxes would be stored in the `${user.data.home}/CloverDX/sandboxes`, where the `user.data.home` is automatically detected user home directory). However, each sandbox can be located on the file system independently of the others if needed. The containing folder and all its contents must have read/write permission for the user under which the **CloverDX Server** is running.
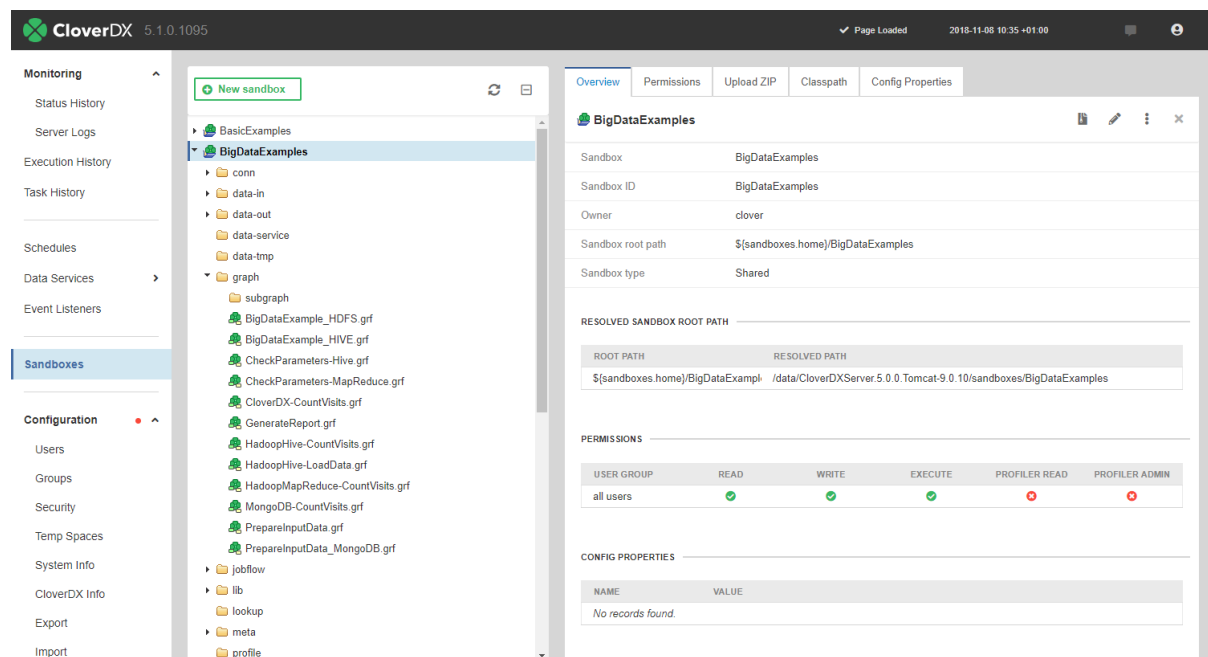
Figure 22.1. Sandboxes Section in CloverDX Server Web GUI

Each sandbox in non-cluster environment is defined by the following attributes:

Table 22.1. Sandbox attributes

Sandbox	A sandbox name used just for display. It is specified by the user during sandbox creation and it can be modified later.
Sandbox ID	A unique name of the sandbox. It is used in server APIs to identify sandbox. It must meet common rules for identifiers. It is specified by user in during sandbox creation and it can be modified later. *Note: modifying is not recommended, because it may be already used by some APIs clients.*
Owner	It is set automatically during sandbox creation. It may be modified later.
Sandbox root path	Absolute server side file system path to sandbox root. It is specified by user during sandbox creation and it can be modified later. Instead of the absolute path, it's recommended to use `${sandboxes.home}` placeholder, which may be configurable in the **CloverDX Server** configuration. So e.g. for the sandbox with ID "dataReports" the specified value of the "root path" would be `${sandboxes.home}/dataReports`. Default value of `sandboxes.home` config property is `${user.data.home}/CloverDX/sandboxes` where the `user.data.home` is configuration property specifying home directory of the user running JVM process - it's OS dependent). Thus on the unix-like OS, the fully resolved sandbox root path may be: `/home/clover/CloverDX/sandboxes/dataReports`. See Chapter 41, [Sandboxes in Cluster](#) (p. 276) for details about sandboxes root path in Cluster environment.
Sandbox type	Type of the sandbox. It can be: local (p. 276), shared (p. 276) or partitioned (p. 277).

Referencing Files from the Graph or Jobflow

In some components you can specify file URL attribute as a reference to some resource on the file system. Also external metadata, lookup or DB connection definition is specified as reference to some file on the filesystem. With **CloverDX Server** there are more ways how to specify this relation.

• Relative path

  All relative paths in your graphs are considered as relative paths to the root of the same sandbox which contains job file (graph or Jobflow).

• sandbox:// URLs

  Sandbox URL allows user to reference the resource from different sandboxes with standalone **CloverDX Server** or the cluster. In cluster environment, **CloverDX Server** transparently manages remote streaming if the resource is accessible only on some specific cluster node.

  For details about the sandbox URLs, see <u>Using a Sandbox Resource as a Component Data Source</u> .

Sandbox Content Security and Permissions

Each sandbox has its owner who is set during sandbox creation. This user has unlimited privileges to this sandbox as well as administrators. Another users may have access according to sandbox settings.

Figure 22.2. Sandbox Permissions in CloverDX Server Web GUI

Permissions to a specific sandbox are modifiable in **Permissions** tab in sandbox detail. In this tab, selected user groups may be allowed to perform particular operations.

There are the following types of operations:

Table 22.2. Sandbox permissions

Read	Users can see this sandbox in their sandboxes list.
Write	Users can modify files in the sandbox through CS APIs.
Execute	Users can execute jobs in this sandbox. **Note:** job executed by **graph event listener** and similar features is actually executed by the same user as job which is the source of the event. See details in **graph event listener**. Job executed by schedule trigger is actually executed by the schedule owner. See details in Chapter 30, Scheduling (p. 192). If the job needs any files from the sandbox (e.g. metadata), the user also must have **read** permission, otherwise the execution fails.
Profiler Read	User can view results of profiler jobs executed from the sandbox.
Profiler Admin	User can administer results of profiler jobs executed from the sandbox.

Note that these permissions modify the access to the content of specific sandboxes. In addition, it is possible to configure permissions to perform operations with sandbox configuration (e.g. create sandbox, edit sandbox, delete sandbox, etc). For details, see Chapter 21, Users and Groups (p. 123).

Sandbox Content and Options

Download sandbox as ZIP (p. 148)
Upload ZIP to sandbox (p. 148)
Download file in ZIP (p. 148)
Download file HTTP API (p. 149)
Delete Sandbox (p. 149)

A sandbox should contain jobflows, graphs, metadata, external connection and all related files. Files, especially graph or jobflow files, are identified by a relative path from sandbox root. Thus you need two values to identify a specific job file: sandbox and path in sandbox. The path to the Jobflow or graph is often referred to as **Job file**.

Although the web GUI section **sandboxes** is not a file-manager, it offers some useful features for sandbox management:
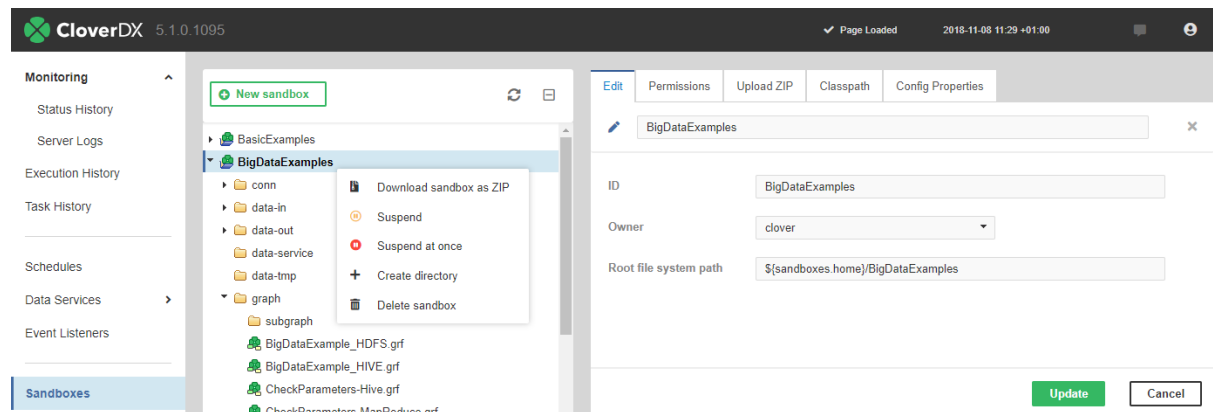
Figure 22.3. GUI - Sandboxes context menu

Download sandbox as ZIP

Select a sandbox in the left panel, then the web GUI displays the **Download sandbox as ZIP** button in the tool bar on the right side.

Created ZIP contains all readable sandbox files in the same hierarchy as on the file system. You can use this ZIP file to upload files to the same sandbox, or another sandbox on a different Server instance.

Upload ZIP to sandbox

Select a sandbox in the left panel. You must have the **write** permission for the selected sandbox. Then select the **Upload ZIP** tab in the right panel. Upload of a ZIP is parametrized by couple of switches, which are described below. Click the + **Upload ZIP** button, to open a common file browser dialog. When you choose a ZIP file, it is immediately uploaded to the Server and a result message is displayed. Each row of the result message contains a description of one single file upload. Depending on selected options, the file may be skipped, updated, created or deleted.

Table 22.3. ZIP upload parameters

Label	Description
Encoding of packed file names	File names which contain special characters (non ASCII) are encoded. In the drop-down list, you choose the right encoding, so filenames are decoded properly.
Overwrite existing files	If this checkbox is checked, the existing file is overwritten by a new one, if both of them are stored in the same path in the sandbox and both of them have the same name.
Delete folders and files missing in uploaded zip file	If this option is enabled, all files which are missing in uploaded ZIP file, but they exist in destination sandbox, will be deleted. This option might cause a loss of data, so the user must have the May delete files missing in uploaded ZIP (p. 131) permission to enable it.

Download file in ZIP

Select a file in the left pane, then the web GUI displays the **Download file as ZIP** button in the tool bar on the right side.

Created ZIP contains just the selected file. This feature is useful for large files (i.e. input or output file) which cannot be displayed directly in the web GUI, so the user can download it.

Download file HTTP API

It is possible to download/view the sandbox file accessing "download servlet" by simple HTTP GET request:

```
http://[host]:[port]/[Clover Context]/downloadFile?[Parameters]
```

The Server requires BASIC HTTP Authentication. Thus with Linux command line HTTP client "wget" it would look like this:

```
wget --user=clover --password=clover
    http://localhost:8080/clover/downloadFile?sandbox=default\&file=data-out/data.dat
```

Please note, that ampersand character is escaped by back-slash. Otherwise it would be interpreted as command-line system operator, which forks processes.

URL Parameters

- sandbox - Sandbox code. Mandatory parameter.
- file - Path to the file relative from sandbox root. Mandatory parameter.
- zip - If set to `true`, the file is returned as ZIP and the response content type is `application/x-zip-compressed`. By default it is `false`, so the response is the content of the file.

Delete Sandbox

You can delete a sandbox by selecting the sandbox and clicking the **Delete sandbox** button on the top of the right pane, or by right-clicking the sandbox in the tree pane on the left and selecting the **Delete sandbox** option.

After that, a confirmation dialog opens where you can choose to delete sandbox files on disk, as well.

Job Config Properties

Each graph or Jobflow may have a set of configuration properties, which are applied during the execution. Properties are editable in the web GUI section **Sandboxes**. Select the job file and go to the **Config properties** tab.

The same configuration properties are editable even for each sandbox. Values specified for a sandbox are applied for each job in the sandbox, but with a lower priority than configuration properties specified for the job.

If neither the sandbox nor the job have configuration properties specified, defaults from main Server configuration are applied. Global configuration properties related to Job configuration properties have the `executor.` prefix. For example, the server property `executor.classpath` is default for the Job configuration property `classpath`. (See Part III, "Configuration" (p. 47) for details)

In addition, it is possible to specify additional job parameters, which can be used as placeholders in job XML. Please keep in mind, that these placeholders are resolved during loading and parsing of the XML file, thus such a job cannot be pooled.

If you use a relative path, the path is relative to `${SANDBOX_ROOT}`.

In a path definition, you can use system properties - e.g. `${java.io.tmpdir}` - and some of server configuration properties: `${sandboxes.home}`, `${sandboxes.home.partitioned}` and `${sandboxes.home.local}`.

Table 22.4. Job configuration properties

Property name	Default value	Description
classloader_caching	`false`	**CloverDX** creates new classloaders when necessary to load a class in runtime. For example, a **Reformat** component with a Java transformation has to create a new classloader to load the class. It is worth noting that classloaders for JDBC drivers are not re-created. Classloader cache is used to avoid PermGen out of memory errors (some JDBC drivers automatically register themself to DriverManager, which can cause the classloader cannot be released by garbage collector). This behavior can be inconvenient for example if you want to share POJO between components. For example, a **Reformat** component creates an object (from a `jar` file on runtime classpath) and stores it into a dictionary. Another **Reformat** component recover the object from the dictionary and attempts to cast the object to the expected class. ClassCastException is thrown due different classloaders used in the **Reformat** components. Using this flag you can force CloverServer to re-use classloader when possible.
classpath		List of paths or `jar` files which contain external classes used in the job file (transformations, generators, JMS processors). All specified resources will be added to the runtime classpath of the transformation job. All **CloverDX** Engine libraries and libraries on application-server's classpath are automatically on the classpath. Separator is specified by the Engine property `DEFAULT_PATH_SEPARATOR_REGEX`. The directory path must always end with a slash character "/", otherwise ClassLoader doesn't recognize it is a directory. Server always automatically adds `trans` subdirectory of job's sandbox, so it doesn't have to be added explicitly.
compile_classpath		List of paths or `jar` files which contain external classes used in the job file (transformations, generators, JMS processors) and related libraries for their compilation. Please note, that libraries on application-server's classpath are not included automatically. Separator is specified by the Engine property `DEFAULT_PATH_SEPARATOR_REGEX`. The directory path must always end with a slash character "/", otherwise ClassLoader doesn't recognize it is a directory. Server always automatically adds a `SANDBOX_ROOT/trans/` directory and all JARs in the `SANDBOX_ROOT/lib/` directory, so they don't have to be added explicitly.

Property name	Default value	Description
debug_mode	false	If `true`, edges with debug enabled will store data into files in a debug directory. Without explicit setting, running of a graph from Designer with server integration would set the debug_mode to `true`. On the other hand, running of a graph from the server console sets the debug_mode to `false`.
delete_obsolete_temp_files	false	If `true`, the system will remove temporary files produced during previous finished runs of the respective job. This property is useful together with enabled debug mode ensuring that obsolete debug files from previous runs of a job are removed from temp space. This property is set to `true` by default when executing job using Designer-Server integration.
enqueue_executions	false	Boolean value. If `true`, executions above `max_running_concurrently` are enqueued, if `false`, executions above `max_running_concurrently` fail.
jobflow_token_tracking	true	If `false`, token tracking in jobflow executions will be disabled.
locale	`DEFAULT_LOCALE` engine property	Can be used to override the `DEFAULT_LOCALE` engine property.
log_level	INFO	Log4j log level for this graph executions. (`ALL` | `TRACE` | `DEBUG` | `INFO` | `WARN` | `ERROR` | `FATAL`). For lower levels (`ALL`, `TRACE` or `DEBUG`), root logger level must be set to lower level, as well. Root logger log level is `INFO` by default, thus a transformation run log does not contain more detail messages then `INFO` event if the job configuration parameter `log_level` is set properly. For details about log4j configuration, see Chapter 17, Logging (p. 103).
max_graph_instance_age	0	A time interval in milliseconds which specifies how long may a transformation instance last in server's cache. 0 means that the transformation is initialized and released for each execution. The transformation cannot be stored in the pool and reused in some cases (a transformation uses placeholders using dynamically specified parameters)
max_running_concurrently	unlimited	The maximum number of concurrently running instances of this transformation. In cluster environment, the limit is per node.
password		This property is deprecated. Password for decoding of encoded DB connection passwords.
skip_check_config	default value is taken from engine property	Specifies whether check config must be performed before a transformation execution.
time_zone	`DEFAULT_TIME_ZONE` engine property	Can be used to override the `DEFAULT_TIME_ZONE` engine property.
tracking_interval	2000	Interval in milliseconds for sampling nodes status in a running transformation.
use_jmx	true	If `true`, job executor registers jmx mBean of a running transformation.
use_local_context_url	false	If `true`, the context URL of a running job will be a local `file:` URL. Otherwise, a `sandbox:` URL will be used.
verbose_mode	true	If `true`, more descriptive logs of job runs are generated.
worker_execution		Set to `false` to enforce execution in Server Core. Can be set per file or per sandbox.

Property name	Default value	Description
	`true`	

Figure 22.4. Job config properties

WebDAV Access to Sandboxes

Since 3.1

WebDAV API allows you to access and manage sandbox content using a standard WebDAV specification.

Specifically, it allows for:

- Browsing a directory structure
- Editing files
- Removing files/folders
- Renaming files/folders
- Creating files/folders
- Copying files
- Moving files

The WebDAV interface is accessible from the URL: "http://[host]:[port]/clover/webdav".

Note: Although common browsers will open this URL, most of them are not rich WebDAV clients. Thus, you will only see a list of items, but you cannot browse the directory structure.

WebDAV Clients

There are many WebDAV clients for various operating systems, some OS support WebDAV natively.

Linux like OS

Great WebDAV client working on Linux systems is Konqueror. Please use different protocol in the URL: `webdav://[host]:[port]/clover/webdav`

Another WebDAV client is Nautilus. Use different protocol in the URL `dav://[host]:[port]/clover/webdav`.

MS windows

Last distributions of MS Windows (Win XP and later) have native support for WebDAV. Unfortunately, it is more or less unreliable, so it is recommended to use some free or commercial WebDAV client.

- The best WebDAV client we've tested is BitKinex: http://www.bitkinex.com/webdavclient

- Another option is to use Total Commander (http://www.ghisler.com/index.htm) with WebDAV plugin: http://www.ghisler.com/plugins.htm#filesys

Mac OS

Mac OS supports WebDAV natively and in this case it should be without any problems. You can use "finder" application, select "Connect to the server ..." menu item and use URL with HTTP protocol: "http://[host]:[port]/clover/webdav".

WebDAV Authentication/Authorization

CloverDX Server WebDAV API uses the HTTP Basic Authentication by default. However it may be reconfigured to use HTTP Digest Authentication.

Digest Authentication may be useful, since some WebDAV clients can't work with HTTP Basic Authentication, only with Digest Authentication.

HTTP Digest Authentication is feature added to the version 3.1. If you upgraded your older **CloverDX Server** distribution, users created before the upgrade cannot use the HTTP Digest Authentication until they reset their passwords. So when they reset their passwords (or the admin does it for them), they can use Digest Authentication as well as new users.

The HTTP Digest Authentication is configured with `security.digest_authentication.*` configuration properties. To enable it, set `security.digest_authentication.features_list` to contain features that are listed in the `security.digest_authentication.features_list`. As items in `security.basic_authentication.features_list` have higher priority, you should empty it to allow HTTP Digest Authentication to be used.

See Chapter 15, for details.

For details on authentication methods see https://tools.ietf.org/html/rfc7617 and https://tools.ietf.org/html/rfc2617 .

Chapter 23. Server Configuration Migration

CloverDX Server provides means to migrate its configuration (e.g. event listeners, schedules, etc.) or parts of the configuration between separate instances of the Server. A typical use case is deployment from test environment to production - this involves not only deployment of graphs, but also copying parts of the configuration, such as file event listeners, etc.

Configuration migration is performed in 2 steps - export of the configuration from the source Server, followed by import of the configuration at the destination Server. After exporting, the configuration is stored as an XML file. The file can be modified manually before import, for example to migrate only parts of the configuration. Additionally, the configuration file can be stored in a versioning system (such as Subversion or Git) for versioning of the **CloverDX Server** configuration.

It is recommended to perform import of configuration on a suspended **CloverDX Server** and to plan for maintenance. Additionally, it is recommended to backup the **CloverDX Server** configuration database before the import.

The following items are parts of the *Server Configuration* and can be migrated between servers:

- Users & Groups (p. 123)

- Sandboxes (p. 144)

- Job Parameters (p. 167)

- Schedules (p. 192)

- Graph Event Listeners(p. 205,) Jobflow Event Listeners(p. 211,) JMS Message Listeners(p. 213,) File Event Listeners (remote and local) (p. 220)

- Temp Spaces (p. 116)

Permissions for Configuration Migration

Whether a user is entitled to perform configuration migration is determined by having *Server Configuration Management* permission; this permission has two sub-permissions: *Export Server Configuration* and *Import Server Configuration* (see Groups permissions (p. 130) for further information on permissions). These permissions are of higher priority than permissions related to a particular migrated item type - so even if the user does not have a permission e.g. to list server's schedules, with *Export Server Configuration* he will be allowed to export all of defined schedules. The same is true for adding and changing items with the *Import Server Configuration* permission.

See Server Configuration permission (p. 140).

Server Configuration Export

Export of Server configuration is performed in the **Configuration** > **Export** section. You can choose which items will be exported (see Figure 23.1 (p. 156)). Click the **Export Configuration** button to save an XML file with the configuration. The name of the XML file includes current timestamp.

If you manually edit the file, make sure that the content is valid. This can be done by validation against XSD schema. The schema for a configuration XML document can be found at `http://[host]:[port]/ [contextPath]/schemas/clover-server-config.xsd`.

The XML file contains selected items of the **CloverDX Server** instance. The file can by modified before importing (p. 157) to another Server instance - for example to import schedules only.

If you want to automate the process of configuration export, use the HTTP API Operation export_server_config (p. 243).

Figure 23.1. Server Configuration Export screen

Server Configuration Import

This function merges exported configuration into the Server. The configuration is loaded from an XML file which can be created by the Server Configuration Export (p. 156) function, or automatically generated. The **Import** function is located in **Configuration** > **Import**.

If you want to automate the process of configuration import, use the HTTP API Operation import_server_config (p. 244).
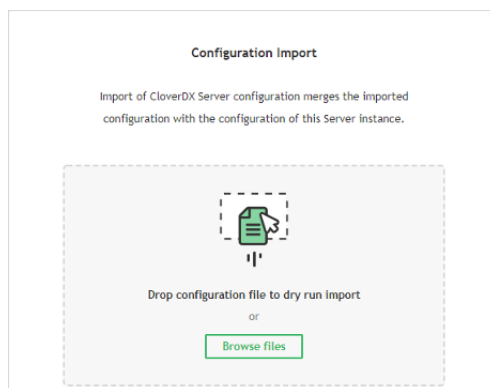
Figure 23.2. Server Configuration Import screen

The XML file defines configuration items to be imported. The items are matched against current configuration of the destination Server. Depending on result, the items are either added to the destination Server or existing item are updated. Matching of items is based on a key that depends on the item type:

Table 23.1. Codes for matching items in configuration import

Item	Code
users	user code
user groups	group code
sandboxes	sandbox code
job parameters	triplet: job parameter name, sandbox code, job file
event listeners	event listener name
schedule	schedule description
launch service	triplet: service name, server user, service user group
data service	pair: sandbox, rjob
temp spaces	pair: temp space node ID, temp space path

Configuration Import Process

Uploading Configuration

Select the XML file with exported configuration. As the first step, the Server executes a safe *dry run* import, without actually changing your Server's configuration. Once uploaded, the Server checks the validity of the configuration and displays a dry run log containing added/updated items or errors.

You will be notified if the source and target differ in at least minor version number (e.g. 4.8.1 and 4.9.0). Importing configuration from different version may generate more warnings which will require your attention.

Verifying Configuration

There are three main states that can occur in a dry run:

1. ⊘ **The configuration is valid:** no errors have occurred and the configuration can be committed.

2. ⚠ **The dry run ended with warning(s):** the configuration can be committed, but the warning(s) should be resolved after import.

3. ⊖ **The dry run ended with error(s):** the configuration cannot be committed until the error(s) are fixed. Consult the dry run log to fix the errors in the XML file and re-upload it.

Dry run log

The **Dry run log** displays changes in the configuration, warns the user about items in configuration that require their attention and notifies them about errors that must be fixed before commit. Changes in the configuration are displayed in a diff view; items before update have a red background while added items or items after update have a green background with changes highlighted:

Below is an example of a log entry indicating a change in the configuration where a user `smithj` is added to a previously empty group `configuration`. The diff view shows the change in two steps (two lines grouped together with no space between them) as 'replacing' the empty group (⊖) with the same group with the new user (⊕).

Figure 23.3. Updated User Groups - User added to a group

Committing Import

Once all errors are resolved and the configuration is valid, you can **Commit Changes**. After confirmation, **Log of committed changes** will display the results. The log can be downloaded, as well.

Some items may not be initialized properly after the import (e.g. their initialization requires presence of a cluster node that went down in the meantime or someone made changes on the Server between dry run and committing). User is notified about these problems in **Log of committed changes** with link to the affected items. You should check such items in respective sections of the Server console and change their settings to fix the issue or remove them.

Chapter 24. Upgrading Server to Newer Version

General Notes on Upgrade

- An upgrade of **CloverDX Server** requires down time; plan a maintenance window.

- A successful upgrade requires about 30 minutes; rollback requires 30 minutes.

- Perform the steps below in a development/testing environment first before moving onto a production environment.

Upgrade Prerequisites

- Having a new **CloverDX Server** web application archive (`clover.war` appropriate for the application server used) & license files available.

- Having [release notes](#) for the particular **CloverDX** version available (and all versions between current and intended version to be upgraded to).

- Having the graphs and jobs updated and tested with regards to known issues & compatibility for the particular **CloverDX** version.

- Having the **CloverDX Server** configuration properties file externalized from default location, see Chapter 12, [Configuration Sources](#) (p. 50).

- Standalone database schema where **CloverDX Server** stores configuration, see Chapter 14, [System Database Configuration](#) (p. 63).

- Having a separate sandbox with a test graph that can be run at any time to verify that **CloverDX Server** runs correctly and allows for running jobs.

Upgrade Instructions

1. Suspend all sandboxes, wait for running graphs to finish processing.

2. Shutdown the **CloverDX Server** application (or all servers, if they run in a cluster mode).

3. Backup the existing **CloverDX** database schema (if any changes to the database schema are necessary, the new server will automatically make them when you start it for the first time).

4. Backup the existing **CloverDX** web application archive (`clover.war`) & license files (on all nodes).

5. Backup the existing **CloverDX** sandboxes (on all nodes).

6. Re-deploy the CloverDX Server web application. Instructions how to do that are application server dependent - see [Production Server](#) (p. 15) for installation details on all supported application servers. After the re-deployment, your new server will be configured based on the previous version's configuration.

7. Replace old license files with the valid one (or you can later use the web GUI form to upload new license). The license file is shipped as a text containing a unique set of characters. If you:

   - received the new license as a file (`*.dat`), then simply use it as new license file.

   - have been sent the license text, e.g. inside an email, then copy the license contents (i.e. all text between `Company` and `END LICENSE`) into a new file called `clover-license.dat`. Next, overwrite the old license file with the new one or upload it in the web GUI.

   For details on license installation, see [Activation](#) (p. 32).

8. - **Standalone Server**

Start the **CloverDX Server** application.

- **Cluster**

  a. Start the **CloverDX Server** application on the first node,

  b. Wait until **CloverDX Server** is accessible (i.e. you can log in),

  c. Start **CloverDX Server** on other nodes.

9. Review that contents of all tabs in the **CloverDX Server** Console, especially scheduling and event listeners looks OK.

10. Update graphs to be compatible with the particular version of **CloverDX Server** (this should be prepared and tested in advance).

11. Resume the test sandbox and run a test graph to verify functionality.

12. Resume all sandboxes.

Upgrade from 4.8.x or earlier to 4.9.x or later

Significant architectural change in version 4.9.0

By default since 4.9.0, jobs (graphs, jobflow, data services) are executed in a standalone JVM called Worker (p. 5).

To run jobs in the Server Core (i.e. in the same way as in earlier versions of **CloverDX Server**), you can disable execution in Worker for particular jobs or sandboxes (p. 151) or disable Worker completely (p. 90).

Note that Launch Services (deprecated) and Profiler are only available in the Server Core. These features require no additional configuration.

Configuration Changes

Worker is the executor of jobs, all jobs run in the Worker by default. It runs in a separate process (JVM), so it requires configuration in addition to the Server Core.

The Worker's configuration relates to memory size, classpath, command line options and JNDI. For an overview of Worker related configuration, see Introduction to Worker configuration (p. 5). The introduction provides an overview of the new Worker specific configuration with links to details.
.

With Worker, new configuration properties (p. 90) were introduced to set up various parameters (timeouts, heap size, etc.). These properties can be added to the properties file (p. 56).

You should split the main memory used by the Server between the Server Core and Worker. Generally, Worker would require more memory than the Server Core as it runs graphs.

The command line arguments and parameters, you were used to add to the Server Core command line in earlier versions, should be added to the Worker's command line, too.

Libraries added to the Server Core classpath should be added to the classpath of Worker. The place to copy these libraries is the `${clover.home}/worker-lib` directory.

Changes to Jobs

If your jobs (DX graphs, jobflows, etc.) use JNDI for database or JMS connections, you need to configure JNDI on Worker (see JNDI in Worker (p. 93)). You might also need to update your jobs to use the new JNDI resources

---

configured in Worker - they might be available on new JNDI paths. If you do not need these JNDI resources on the Server Core anymore, consider removing them from the Server Core.

Rollback Instructions

1. Shutdown the **CloverDX Server** application.

2. Restore the **CloverDX Server** web application (`clover.war`) & license files (on all nodes).

3. Restore the **CloverDX Server** database schema.

4. Restore the **CloverDX** sandboxes (on all nodes).

5. Start the **CloverDX Server** application (on all nodes).

6. Resume the test sandbox and run a test graph to verify functionality.

7. Resume all sandboxes.

Important

Evaluation Version - a mere upgrade of your license is not sufficient. When moving from an evaluation to production server, you should not use the default configuration and database. Instead, take some time to configure **CloverDX Server** so that it best fits your production environment.

Chapter 25. Diagnostics

This chapter introduces you into diagnostic tools that help you localize performance and memory issues.

CloverDX Server allows you to create a thread dump or a heap dump. The thread and heap dumps are useful for investigation of performance and memory issues.

In server GUI, go to **Configuration** →**System Info** →**Diagnostics**.

Heap Dump

A **heap dump** is a content of the JVM process memory stored in a binary file. Creating a heap dump requires a [Heap Memory Dump permission](#) (p. 141).

Server Core

To download a **heap dump** of the Server Core, click the **Download** button under the **Heap Dump** section. Downloading the **heap dump** may take some time.

The **Dump live objects only** checkbox allows you to avoid dumping of objects awaiting garbage collection.

Worker

The heap dump of Worker can be created with jcmd command: `jcmd <pidOfWorker> GC.heap_dump <filename>` You should specify the file name with full path to avoid searching for the file as `jcmd` does not always create it in the working directory.

See [details on jcmd](#).

To generate a heap dump on Out of Memory errors, add `-XX:+HeapDumpOnOutOfMemoryError` to the worker.jvmOptions (p. 91) property. A dump file `java_pid.hprof` will be generated when an Out of Memory error occurs. The heap dump will be located in the working directory of the Worker's process (same as the Server Core's working directory). You can override the directory location with the `-XX:HeapDumpPath=/disk2/dumps` option. **Important:** the generated file can be large, its size is equal to the heap size.

The generated heap dump file can be investigated with tools like `jvisualvm` or `jhat`.

Heap Dump Analysis

You can use **jvisualvm** or **jhat** to view and analyze the heap dump.

> ## Important
>
> A **heap dump** does not work on **WebSphere**. On WebSphere, you can create heap dumps using the administration console. For the instructions, see [IBM Knowledge Center](#).

Thread Dump

A **thread dump** is a list of existing JVM threads with their callstacks and held locking objects (if supported). It can be viewed in a text editor. Creating a thread dump requires a [Heap Memory Dump permission](#) (p. 141).

Server Core

To download the thread dump, click the **Download** button under the **Thread Dump** section.

Worker

The thread dump of Worker can be created with jcmd command: `jcmd <pidOfWorker> Thread.print`

See [details on jcmd](#).

Enabling GC Logging

Some memory and performance issues can be investigated with help of garbage collection logs.

Core

To enable the logging of the Server Core, add `-verbose:gc   -XX:+PrintGCTimeStamps   -Xloggc:server_core_gc_log.txt` to `JAVA_OPTS` in `$CATALINA_HOME/bin/setenv.sh`. Restart of the Server is required to reflect the configuration change.

Worker

To enable the garbage collection logging in Worker, add the flags `-verbose:gc   -XX:+PrintGCTimeStamps -Xloggc:worker_gc_log.txt` to the Worker's JVM arguments (p. 58) field in the Setup GUI, or the worker.jvmOptions (p. 91) property.

The `-Xloggc` option sets the path for the detailed garbage collector log. For the GC logging of Worker, use a different file name than for the Server. You can analyze the log file with various tools, e.g. http://gceasy.io/.

Restart of Worker is **required**.

More Details

Another useful garbage collector flags are `-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCCause   -XX:+UseGCLogFileRotation   -XX:NumberOfGCLogFiles=10   -XX:GCLogFileSize=5M -XX:+PrintTenuringDistribution`

For details on JVM flags, see [Oracle's Java HotSpot VM Options](#).

Additional Diagnostic Tools

Below are additional useful diagnostic tools:

Investigating usage of direct memory

To investigate usage of direct memory, add `-XX:NativeMemoryTracking=summary` the worker.jvmOptions (p. 91) property. The details on native memory usage can be displayed with **jcmd <pid> VM.native_memory summary**. For more information, see [Native Memory Tracking tool](#).

Enable remote JMX monitoring

Add the following options to the worker.jvmOptions (p. 91) property:

```
-Dcom.sun.management.jmxremote=true                                        -
Dcom.sun.management.jmxremote.port=8687                                    -
Dcom.sun.management.jmxremote.authenticate=false                          -
Dcom.sun.management.jmxremote.ssl=false                                    -
Djava.rmi.server.hostname=example.com
```

With the above options, you enable remote connection to JMX monitoring, which provides a wide range of information about the running JVM, JNDI resources etc. Change the value of `java.rmi.server.hostname` to the hostname of your Server.

Chapter 26. Troubleshooting Worker

Worker Logs

When investigating issues with Worker itself or jobs running in Worker, there are several logs with useful information:

Logs found in the **Monitoring →Server Logs** section of the Server Console.

- **COMMON** log - the main Server log contains also information related to Worker. This log contains the full command line used to start Worker, this allows you to check the command line arguments. Additionally, standard output of the Worker process is redirected to this log - this is useful especially if the Worker process crashes during startup.

  The COMMON log file is located in `${java.io.tmpdir}/cloverlogs/all.log`

- **WORKER** log - the main Worker log provides information about Worker startup, initialization, executed jobs, runtime activities, etc. The initialization details contain information about Worker's JNDI resources, etc.

  The WORKER log file is located in `${java.io.tmpdir}/cloverlogs/worker_[nodeID].log`

  You can also open this log via the **Go to logs** action in the **Worker** section of the **Monitoring** page.

Worker Command Line

The full command line that was used to start the Worker process can be found in:

- Monitoring section, use the **Show command line** action on Worker. For more details, see .

- the COMMON log of the Server (found in the **Monitoring →Server Logs** page). See section above for more details.

Investigate the command line options in case Worker does not correctly start or if the configuration of the running Worker is not correct.

Worker Does Not Start

If Worker does not start, check the following:

- Server's COMMON log and the WORKER log, see . Look for errors during Worker startup and initialization.

- Worker's command line arguments, see . Look for invalid command line arguments. Additionally, check the custom JVM arguments set on Worker, in the Worker (p. 58) tab of Setup (p. 54) or via the worker.jvmOptions (p. 91) configuration property.

Restarting Worker

In case Worker gets into an unrecoverable state (e.g. out of heap memory, etc.) and you fix the source issue, you can restart it from the Monitoring section (For more details, see .):

- restart immediately, which will abort jobs currently running in Worker;

- restart after running jobs finish, in case the currently running jobs are crucial.

Worker does not start

In case Worker does not start (i.e. remains in the `STARTING` status), check the COMMON log first.

Worker Crashes

If common log (`all.log` file) contains row similar to the following one, the worker crashed due to exhausted heap space.

```
2018-03-22 16:08:29,008[s StdOut reader] WorkerProcess    INFO    [worker0@N1:10500]: java.lang.OutOfMemoryError: Java heap
```

You can configure it to generate heap dump for further investigation. To do so, add `-XX:+HeapDumpOnOutOfMemoryError` to JVM arguments in Worker configuration (**Configuration** > **Setup** > **Worker**). The generated file can be investigated with tools like `jvisualvm` or `jhat`.

Another cause of the crash can be swapped Worker's main memory. If there is insufficient free space in the main memory and the pages of Worker process are swapped on the hard drive, Worker is slowed down, it does not receive heart beat from the Server in time and it kills itself. Lower the maximum heap size of Worker to avoid swapping. Make note java process uses also non-heap memory, e.g. metaspace or direct memory.

To investigate usage of direct memory, add `-XX:NativeMemoryTracking=summary` to Worker's JVM arguments (**Configuration** > **Setup** > **Worker**). The details on native memory usage can be displayed with **jcmd <pid> VM.native_memory summary**. See https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr007.html for details on native memory tracking.

Worker Hangs

Usually, it is caused by garbage collector. Try tweaking the garbage collection.

Another cause can be swapping of worker process pages on hard drive.

Issues with Classloading

To debug issues with classloading, add `-verbose:class` to **JVM arguments** of Worker. Loaded and unloaded classes will be printed to the output. The output can be seen in the common log.

Part V. Using Graphs

Chapter 27. Graph/Jobflow Parameters

The **CloverDX Server** passes a set of parameters to each graph or jobflow execution.

Keep in mind that $\${paramName}$ placeholders (parameters) are resolved only during the initialization (loading of XML definition file), so if you need the parameters to be resolved for each job execution, you cannot set the job to be pooled. However, current parameter values are always accessible by an inline Java code:

```
String runId = getGraph().>getGraphProperties().getProperty("RUN_ID");
```

Properties may be added or replaced:

```
getGraph().getGraphProperties().setProperty("new_property", value);
```

This is a set of parameters which are always set by **CloverDX Server** (for more information, see Job Config Properties (p. 150)):

Table 27.1. Defaults for graph execution configuration

key	description
SANDBOX_CODE	An identifier of a sandbox which contains the executed graph.
JOB_FILE	A path to the file (graph, subgraph, jobflow). The path relative to the sandbox root path.
SANDBOX_ROOT	An absolute path to sandbox root.
RUN_ID	An ID of the graph execution. In a standalone mode or cluster mode, it is always unique. It may be lower than 0 value, if the run record isn't persistent. For more information, see Chapter 39, Launch Services (p. 249).
PARENT_RUN_ID	A run ID of the graph execution which is a parent to the current one. Useful when the execution is a subgraph, child-job of some jobflow or worker for distributed transformation in a cluster. When the execution doesn't have a parent, the PARENT_RUN_ID is the same as RUN_ID.
ROOT_RUN_ID	A run ID of the graph execution which is a root execution to the current one (the one which doesn't have a parent). Useful when the execution is a subgraph, child-job of some jobflow or worker for distributed transformation in a cluster. When the execution doesn't have a parent, the ROOT_RUN_ID is the same as RUN_ID.
CLOVER_USERNAME	The username of the user who launched the graph or jobflow.
NODE_ID	An ID of the node running the graph or jobflow.

Parameters by Execution Type

Additional parameters are passed to the graph depending on how the graph is executed:

Executed from Web GUI

Graphs executed from the web GUI have no additional parameters.

Executed by Launch Service Invocation

Service parameters which have the **Pass to graph** attribute enabled are passed to the graph not only as "dictionary" input data, but also as a graph parameter.

Executed by HTTP API Run Graph Operation Invocation

Any URL parameter with a `param_` prefix is passed to an executed graph but without the `param_` prefix, i.e. `param_FILE_NAME` specified in a URL is passed to the graph as a property named `FILE_NAME`.

Executed by RunGraph Component

Since 3.0, only parameters specified by the **paramsToPass** attribute are passed from the parent graph to the executed graph. However common properties (RUN_ID, PROJECT_DIR, etc.) are overwritten with new values.

Executed by WS API Method executeGraph Invocation

Parameters with values may be passed to the graph with a request for execution.

Executed by Task Graph Execution by Scheduler

Table 27.2. passed parameters

key	description
EVENT_SCHEDULE_EVENT_TYPE	The type of a schedule: SCHEDULE_PERIODIC | SCHEDULE_ONETIME
EVENT_SCHEDULE_LAST_EVENT	Date/time of a previous event
EVENT_SCHEDULE_DESCRIPTION	A schedule description, which is displayed in the web GUI
EVENT_USERNAME	The owner of the event. For schedule it is the user who created the schedule.
EVENT_SCHEDULE_ID	An ID of the schedule which triggered the graph

Executed from JMS Listener

There are many graph parameters and dictionary entries passed, depending on the type of incoming message. See details in JMS Message Listeners (p. 213).

Executed by Task Start a Graph by Graph/Jobflow Event Listener

Since 3.0, only *specified* properties from a "source" job are passed to the executed job, by default. This behavior can be changed by the `graph.pass_event_params_to_graph_in_old_style` Server config property so that *all* parameters from a "source" job are passed to the executed job. This switch is implemented for backwards compatibility. With the default behavior, in the editor of graph event listener, you can specify a list of parameters to pass. For more information, see .

The following parameters with current values are always passed to the target job

Table 27.3. passed parameters

key	description
EVENT_RUN_SANDBOX	A sandbox with the graph which is the source of the event
EVENT_JOB_EVENT_TYPE	GRAPH_STARTED	GRAPH_FINISHED	GRAPH_ERROR	GRAPH_ABORTED	GRAPH_TIMEOUT	GRAPH_STATUS_UNKNOWN, analogically JOBFLOW_* for jobflow event listeners.
EVENT_RUN_JOB_FILE	A jobFile of the job which is the source of the event
EVENT_RUN_ID	An ID of the graph execution which is the source of the event.
EVENT_TIMEOUT	A number of milliseconds which specifies an interval of timeout. Useful only for "timeout" graph event.
EVENT_RUN_RESULT	A result (or current status) of the execution which is the source of the event.
EVENT_USERNAME	The owner of the event. For graph events it is the user who created the graph event listener

Executed by Task Graph Execution by File Event Listener

Table 27.4. passed parameters

key	description
EVENT_FILE_PATH	A path to the file which is the source of the event. Does not contain a file name. Does not end with a file separator. Is passed only for the local file event listener.
EVENT_FILE_NAME	A filename of the file which is the source of the event. Is passed only when the "grouping" mode is disabled. Otherwise there are more than one file event.
EVENT_FILE_URLS	Contains string, which may be used "as is" in the **file URL** attribute of various **CloverDX** components. It may contain a URL to one or more (if grouping is enabled) files. It may contain local path(s) or remote URL(s) where credentials are replaced by placeholders (due to security reasons).
EVENT_FILE_AUTH_USERNAME	A username/ID to the remote location.
EVENT_FILE_AUTH_USERNAME_URL_ENCODED	The same as EVENT_FILE_AUTH_USERNAME, but the value is also URL encoded, so it may be used in the URL.
EVENT_FILE_AUTH_PASSWORD	Password/key to the remote location. It's encrypted by the master password. It is passed only when the file listener uses user+password authentication.
EVENT_FILE_AUTH_PASSWORD_URL_ENCODED	

key	description
	The same as EVENT_FILE_AUTH_PASSWORD, but the value is also URL encoded, so it may be used in the URL (EVENT_FILE_URLS parameter).
EVENT_FILE_EVENT_TYPE	SIZE | CHANGE_TIME | APPEARANCE | DISAPPEARANCE
EVENT_FILE_PATTERN	A pattern specified in a file event listener
EVENT_FILE_LISTENER_ID	
EVENT_USERNAME	Owner of the event. For file events, it is the user who created the file event listener.

Adding Another Graph Parameters

Additional "Graph Config Parameters"

It is possible to add so-called additional parameters in the web GUI (Chapter 22, Sandboxes - Server Side Job Files (p. 144)) for the selected graph or for all graphs in the selected sandbox. See details in Job Config Properties (p. 150).

Task "execute_graph" Parameters

The **execute graph** task may be triggered by a schedule, graph event listener, or file event listener. The task editor allows you to specify key=value pairs which are passed to the executed graph.

Chapter 28. Tasks

A task is a graph, jobflow, Groovy script, etc. that can be started manually, started on scheduled time, or triggered by some event. A task specifies WHAT to do.

There are several tasks implemented for a schedule and graph event listener as follows:

Tasks in Cluster Environment

In Cluster environment, you can specify a node where the task runs. The task can run on **Any node** or on **one of selected nodes**. If there is no node ID specified, the task may be processed on any cluster node; so in most cases, it will be processed on the same node where the event was triggered. If there are some nodeIDs specified, task will be processed on the first node in the list which is connected in cluster and ready.

Tasks are used in

Send an Email

The **send e-mail** task is useful for notifications about a result of graph execution. For example, you can create a listener with this task type to be notified about each failure in the specified sandbox or a failure of the particular graph.

This task is very useful, but for now only as a response for graph events. This feature is very powerful for monitoring. (for description of this task type, see Graph Event Listeners (p. 205)).

Note: It seems useless to send emails periodically, but it may send current server status or daily summary. These features will be implemented in further versions.

Table 28.1. Attributes of "Send e-mail" task

Task type	"Send an email"
To	The recipient's email address. It is possible to specify more addresses separated by a comma. It is also possible to use placeholders. For more information, see Placeholders (p. 173).
Cc	Cc stands for 'carbon copy'. A copy of the email will be delivered to these addresses. It is possible to specify more addresses separated by a comma. It is also possible to use placeholders. For more information, see Placeholders (p. 173).
Bcc	Bcc: stands for 'Blind carbon copy'. It is similar as Cc, but the others recipients aren't aware, that these recipients received a copy of the email.
Reply-to (Sender)	Email address of sender. It must be a valid address according to the SMTP server. It is also possible to use placeholders. For more information, see Placeholders (p. 173).
Subject	An email subject. It is also possible to use placeholders. For more information, see Placeholders (p. 173).
HTML	A body of the email in HTML. The email is created as multipart, so the HTML body should have a precedence. A plain text body is only for email clients which do not display HTML. It is also possible to use placeholders. For more information, see Placeholders (p. 173).
Text	A body of the email in plain text. The email is created as multipart, so the HTML body should have a precedence. A plain text body is only for email clients which do not display HTML. It is also possible to use placeholders. For more information, see Placeholders (p. 173).
Log file as attachment	If this switch is checked, the email will have an attachment with a packed log file of the related graph execution.
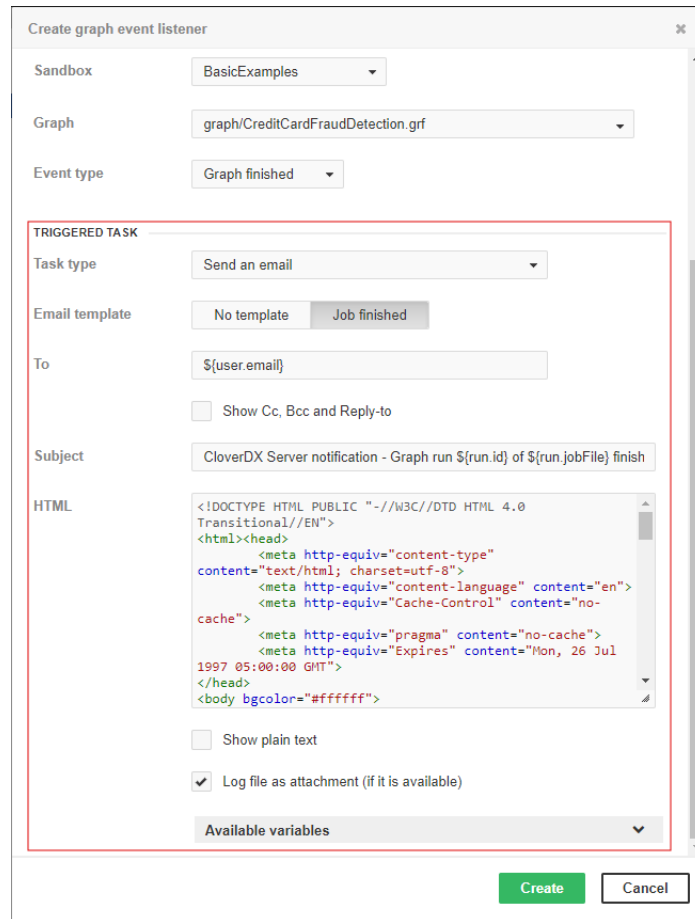
Figure 28.1. Web GUI - send email

Note: Do not forget to configure a connection to an SMTP server (see Part III, "Configuration" (p. 47)).

Placeholders

Placeholder may be used in some fields of tasks. They are especially useful for email tasks, where you can generate the content of email according to context variables.

Note: In most cases, you can avoid this by using email templates (See E-mail task for details)

These fields are preprocessed by Apache Velocity templating engine. See the Velocity project URL for syntax description http://velocity.apache.org/.

There are several context variables, which you can use in placeholders and even for creating loops and conditions.

- *event*
- *now*
- *user*
- *run*
- *sandbox*

Some of them may be empty depending on the type of the event. For example, if a task is processed because of a graph event, then *run* and *sandbox* variables contain related data, otherwise they are empty.

Table 28.2. Placeholders useful in email templates

Variable name	Contains
now	Current date-time
user	The user, who caused this event. It may be an owner of a schedule, or someone who executed a graph. It contains sub-properties which are accessible using dot notation (i.e. ${user.email}) email: • user.email • user.username • user.firstName • user.lastName • user.groups (list of values)
run	A data structure describing one single graph execution. It contains sub-properties which are accessible using dot notation (i.e. ${run.jobFile}) • job.jobFile • job.status • job.startTime • job.stopTime • job.errNode • job.errMessage • job.errException • job.logLocation

tracking	A data structure describing a status of components in a graph execution. It contains sub-properties which are accessible using the Velocity syntax for loops and conditions.
	
	<pre>#if (${tracking})
	<table border="1" cellpadding="2" cellspacing="0">
	#foreach ($phase in $tracking.trackingPhases)
	<tr><td>phase: ${phase.phaseNum}</td>
	    <td>${phase.executionTime} ms</td>
	   <td></td><td></td><td></td></tr>
	  #foreach ($node in $phase.trackingNodes)
	    <tr><td>${node.nodeName}</td>
	        <td>${node.result}</td>
	        <td></td><td></td><td></td></tr>
	    #foreach ($port in $node.trackingPorts)
	      <tr><td></td><td></td>
	          <td>${port.type}:${port.index}</td>
	          <td>${port.totalBytes} B</td>
	          <td>${port.totalRows} rows</td></tr>
	    #end
	  #end
	#end
	</table>
	#end
	}</pre>

sandbox	A data structure describing a sandbox containing an executed graph. It contains sub-properties which are accessible using dot notation (i.e. ${sandbox.name}) • sandbox.name • sandbox.code • sandbox.rootPath
schedule	A data structure describing a schedule which triggered this task. It contains sub-properties which are accessible using dot notation (i.e. ${schedule.description}) • schedule.description • schedule.startTime • schedule.endTime • schedule.lastEvent • schedule.nextEvent • schedule.fireMisfired

Execute Shell Command

Execute Shell Command executes a system command or shell script.

This task is used in Chapter 30, Scheduling (p. 192), Chapter 32, Listeners (p. 203) and Chapter 29, Manual Task Execution (p. 191).

Table 28.3. Attributes of "Execute shell command" task

Task type	"Execute shell command"
Start on	Node IDs to process the task. This attribute is accessible only in Cluster environment. If there are nodes specified, the task will be processed on the first node which is online and ready.
Shell script	Command line for execution of external process.
Working directory	A working directory for the process. If not set, the working directory of the application server process is used.
Timeout	Timeout in milliseconds. After a period of time specified by this number, the external process is terminated and all results are logged.

Figure 28.2. Web GUI - shell command

Execute Shell Command Parameters

Some parameters are available only in particular context: scheduling, event listeners, or manual task execution.

Table 28.4. Parameters of "Execute shell command" task

event	An event that has triggered the task
now	Current date-time
task	The triggered task
user	The object representing the user who executed the graph/jobflow. It contains sub-properties that are accessible using dot notation (i.e. ${user.email}) • user.email • user.username • user.firstName • user.lastName • user.groups (list of values)

Table 28.5. Parameters of "Execute shell command" task - available in scheduling

schedule	The object representing the schedule that triggered this task. It contains sub-properties that are accessible using dot notation (i.e. ${schedule.description}) • schedule.description • schedule.startTime • schedule.endTime • schedule.lastEvent • schedule.nextEvent • schedule.fireMisfired
EVENT_USERNAME	The name of the user who caused the event.
EVENT_USER_ID	The numeric ID of the user who caused the event.
EVENT_SCHEDULE_DESCRIPTION	A description of the schedule.
EVENT_SCHEDULE_EVENT_TYPE	The type of the schedule - SCHEDULE_ONETIME or SCHEDULE_PERIODIC.
EVENT_SCHEDULE_ID	The numeric ID of the schedule
EVENT_SCHEDULE_LAST_EVENT	Date-time of the latest schedule triggering (in java.util.Date.toString() format).

Table 28.6. Parameters of "Execute shell command" task - available in listeners

run	The object representing a single graph/jobflow execution. It contains sub-properties that are accessible using dot notation (i.e. ${run.jobFile}). • run.jobFile • run.status • run.startTime • run.stopTime • run.errNode • run.errMessage • run.errException

sandbox	The object representing a sandbox containing the executed graph/jobflow. It contains sub-properties that are accessible using dot notation (i.e. ${sandbox.name}) • sandbox.name • sandbox.code • sandbox.rootPath
tracking	An object represeting a status of components in a graph execution. It contains sub-properties that are accessible using Velocity syntax for loops and conditions.
EVENT_USERNAME	The name of the user who caused the event.
EVENT_USER_ID	A numeric ID of the user who caused the event.
EVENT_RUN_SANDBOX	A code of the sandbox containing the graph/jobflow.
EVENT_RUN_JOB_FILE	A sandbox-relative path to the graph/jobflow file.
EVENT_RUN_RESULT	The current status of the graph/jobflow execution • N_A • READY • RUNNING • WAITING • FINISHED_OK • ERROR • ABORTED • TIMEOUT • UNKNOWN
EVENT_RUN_ID	A numeric ID of the run record representing graph/jobflow execution
EVENT_TIMEOUT	A specified timeout (in milliseconds) for the TIMEOUT event to occur.
EVENT_JOB_EVENT_TYPE	Graph event that triggered the task • GRAPH_STARTED • GRAPH_PHASE_FINISHED • GRAPH_FINISHED • GRAPH_ERROR • GRAPH_ABORTED • GRAPH_TIMEOUT • GRAPH_STATUS_UNKNOWN

Table 28.7. Parameters of "Execute shell command" task - available in manual task execution

parameters	Task parameters - container for String-String key-value pairs passed to this task.

Start a Graph

Start a Graph starts a specified graph from a specified sandbox.

Table 28.8. Attributes of "Graph execution" task

Task type	"Start a graph"
Start on	Node(s) to process the task. This attribute is accessible only in Cluster environment. If there are nodes specified, the task will be processed on the first node which is online and ready.
Sandbox	This select box contains sandboxes which are readable for the logged user. Select the sandbox which contains the graph to execute.
Graph	The graph to be executed. This select box is filled with all graphs files accessible in the selected sandbox. Type a graph name or path to filter available items.
Save run record	Saves run record to database. If the task runs too often (once in several seconds), you can increase the database performance by disabling this attribute.
Parameters	A list of parameters passed to the graph. Event parameters like `EVENT_RUN_RESULT`, `EVENT_RUN_ID`, etc. are passed to the executed job without limitations. The `EVENT_RUN_RESULT` and `EVENT_RUN_ID` parameters are used in context of event listeners. They are not used in context of scheduling.

Figure 28.3. Web GUI - Graph execution task

Please note that the behavior of this task type is almost the same as Start a Jobflow (p. 181).

Parameters

You can start a graph with parameters.

To start a graph with a parameter, choose an existing parameter from the list, set its value, and click the **plus sign** button at the end of line.

If the graph is started by an event listener, it receives additional parameters from the triggering job.

Parameters passed to graph by Event Listeners

Table 28.9. Additional parameters available in Event Listeners

EVENT_USERNAME	The name of the user who caused the event
EVENT_USER_ID	A numeric ID of the user who caused the event.
EVENT_RUN_SANDBOX	A code of the sandbox containing the graph/jobflow
EVENT_RUN_JOB_FILE	A sandbox-relative path to the graph/jobflow file.
EVENT_RUN_RESULT	The current status of the graph/jobflow execution (N_A, READY, RUNNING, WAITING, FINISHED_OK, ERROR, ABORTED, TIMEOUT or UNKNOWN).
EVENT_RUN_ID	A numeric ID of the run record representing graph/jobflow execution.
EVENT_JOB_EVENT_TYPE	A graph/jobflow event type that triggered the task.
EVENT_TIMEOUT	A specified timeout (in milliseconds) for the TIMEOUT event to occur

Start a Jobflow

Start a jobflow starts a specified jobflow from a specified sandbox.

Table 28.10. Attributes of "Jobflow execution" task

Task type	"Start a jobflow"
Start on	Node(s) to process the task. This attribute is accessible only in Cluster environment. If there are nodes specified, the task will be processed on the first node which is online and ready.
Sandbox	This select box contains sandboxes which are readable for the logged user. Select sandbox which contains jobflow to execute.
Jobflow	This select box is filled with all jobflow files accessible in the selected sandbox. Type the jobflow name or path to filter available items.
Save run record	Saves run record to database. If the task runs too often (once in several seconds), you can increase the database performance by disabling this attribute.
Parameters	Key-value pairs which are passed to the executed job as parameters. Event parameters like EVENT_RUN_RESULT, EVENT_RUN_ID, etc. are passed to the executed job without limitations. The EVENT_RUN_RESULT and EVENT_RUN_ID parameters are used in context of event listeners. They are not used in context of scheduling.

Figure 28.4. Web GUI - Jobflow execution task

Please note that the behavior of this task type is almost the same as <u>Start a Graph</u> (p. 178).

If the jobflow start is triggered by an event, the same set of parameters as in a graph event listener is passed to the jobflow. <u>Parameters</u> (p. 180).

Start a Profiler Job

Start a profiler job starts a specified profiler job from a specified directory.

You can pass parameters to the profiler job in the same way as in case of starting a graph (p. 178) or jobflow (p. 181).

In case of triggering the profiler job by an event listener, the same set of additional parameters, as in case of execution of graph, is passed to the profiler job. See .

Figure 28.5. Web GUI - Profiler job execution task

Abort job

This task kills/aborts a specified job (graph or jobflow), if it is currently running.

Table 28.11. Attributes of "Abort job" task

Task type	"Abort job"
Start on	Node(s) to process the task. This attribute is accessible only in Cluster environment. If there are nodes specified, the task will be processed on the first node which is online and ready.
Kill source of event	If this switch is on, the task will kill the job which is the source of the event, which activated this task. Attributes **sandbox** and **job** are ignored. This checkbox is useful only if **Abort job** is activated by some event.
Sandbox	Select a sandbox which contains the job to kill. This attribute works only when the **Kill source of event** switch is off.
Job	This select box is filled with all jobs accessible in the selected sandbox. All instances of the selected job that are currently running and will be killed. This attribute works only when **Kill source of event** switch is off.

Figure 28.6. Web GUI - "Abort job"

Archive Records

This task can archive (or delete) obsolete records from the database.

Table 28.12. Attributes of "Archivator" task

Task type	"Archivator"
Start on	This attribute specifies a cluster node on which the task may process. This attribute is accessible only in Cluster environment. If it is empty, it may be any node; if there are nodes specified, the task will be processed on the first node which is online and ready.
Archivator type	There are two possible values: `archive` or `delete`. `Delete` removes records without any possibility of recovery. `Archive` option removes records from the database, but creates a ZIP package with CSV files containing the deleted data.
Older than	Time period (in minutes) - specifies which records are evaluated as obsolete. Records older than the specified interval are stored in archives.
Output path for archives	This attribute is useful only for the `archive` option.
Include executions history	
Include temp files	If checked, the archivator removes all graph temporary files older than the given timestamp defined in **Older than** attribute. The temporary files are files with graph debug data, dictionary files and files created by graph components.
Include tasks history	If checked, the archivator will include run records. Log files of graph runs are included as well.
Include profiler runs	If checked, the archivator will include profiler job results.
Include server instance history	

Figure 28.7. Web GUI - archive records

Send a JMS Message

This type of task is useful for notifications about result of a graph execution. For example, you can create a graph event listener with this task type to be notified about each failure in a specific sandbox or failure of a particular graph.

JMS messaging requires JMS API (jms.jar) and third-party libraries. All these libraries must be available on the application server classpath. Some application servers contain these libraries by default, some do not, thus the libraries must be added explicitly.

Table 28.13. Attributes of JMS message task

Task type	"JMS message"
Initial context	Choose between the **default** and **custom** initial context.
Initial context class name	A full class name of `javax.naming.InitialContext` implementation. Each JMS provider has its own implementation. For example, in case of Apache MQ, it is `org.apache.activemq.jndi.ActiveMQInitialContextFactory`. If it is empty, the Server uses the default initial context.
Broker URL	
Connection factory JNDI name	The JNDI name of a connection factory. It depends on a JMS provider.
Destination JNDI name	The JNDI name of a message queue/topic on the server
Username	A username for connection to a JMS message broker
Password	A password for connection to a JMS message broker
URL	A URL of a JMS message broker
JMS pattern	This select box is available only when the user is creating a new record. It contains all predefined JMS message patterns. If the user chooses any of them, the text field below is automatically filled with a value from the pattern.
Text	The body of a JMS message. It is also possible to use placeholders. *See* Placeholders *(p. 173) of send email task for details.*

Figure 28.8. Web GUI - Task JMS message editor

Table 28.14. Parameters of "Send a JMS Message"

event	The event that triggered the task.
now	Current date-time
task	The triggered task.
user	The object representing the owner of the schedule. It contains sub-properties that are accessible using dot notation (i.e. ${user.email}) email, username, firstName, lastName, groups (list of values).
schedule	The object representing the schedule that triggered this task. It contains sub-properties that are accessible using dot notation (i.e. ${schedule.description}) description, startTime, endTime, lastEvent, nextEvent, fireMisfired.
EVENT_USERNAME	The username of the user who caused the event
EVENT_USER_ID	A numeric ID of the user who caused the event.
EVENT_SCHEDULE_DESCRIPTION	A description of the schedule
EVENT_SCHEDULE_EVENT_TYPE	The type of the schedule - SCHEDULE_ONETIME or SCHEDULE_PERIODIC.
EVENT_SCHEDULE_ID	A numeric ID of the schedule.
EVENT_SCHEDULE_LAST_EVENT	Date-time of the latest schedule triggering (in java.util.Date.toString() format).

Execute Groovy Code

This type of task allows to execute a code written in the Groovy script language. The script can be defined in place or using a path to external `.groovy` file. It is possible to use some variables.

The basic attribute of this task is a source code of written in Groovy.

If the source codes are provided from both a file and through the input form, only the code from the input form will be executed.

In Cluster environment, there is also one additional attribute **Node IDs to process the task**. If it is empty, it may be any node; if there are nodes specified, the task will be processed on the first node which is online and ready.

CloverDX Server contains Groovy version 2.0.0

Table 28.15. List of variables available in Groovy code

variable	class	description	availability
event	com.cloveretl.server.events.AbstractServerEvent		every time
task	com.cloveretl.server.persistent.Task		every time
now	java.util.Date	current time	every time
parameters	java.util.Properties	Properties of a task	every time
user	com.cloveretl.server. persistent.User	Same as event.getUser()	every time
run	com.cloveretl.server. persistent.RunRecord		When the event is an instance of GraphServerEvent
tracking	com.cloveretl.server. worker.commons.persistent.TrackingGraph	same as run.getTrackingGraph()	When the event is an instance of GraphServerEvent
sandbox	com.cloveretl.server. persistent.Sandbox	same as run.getSandbox()	When the event is an instance of GraphServerEvent
schedule	com.cloveretl.server. persistent.Schedule	same as ((ScheduleServerEvent)event). getSchedule()	When the event is an instance of ScheduleServerEvent
servletContext	javax.servlet.ServletContext		every time
cloverConfiguration	com.cloveretl.server.spring.CloverConfiguration	Configuration values for **CloverDX Server**	every time
serverFacade	com.cloveretl.server.facade. api.ServerFacade	The reference to the facade interface. Useful for calling **CloverDX Server** core. WAR file contains JavaDoc of facade API and it is accessible on URL: http://host:port/ clover/javadoc/index.html	every time
sessionToken	String	A valid session token of the user who owns the event. It is useful for authorization to the facade interface.	every time

Variables run, tracking and sandbox are available only if the event is an instance of GraphServerEvent class. A variable schedule is only available for ScheduleServerEvent as an event variable class.

Example of use Groovy script

This example shows a script which writes a text file describing the finished graph. It shows use of the 'run' variable.

```
import com.cloveretl.server.persistent.RunRecord;
String dir = "/tmp/";
RunRecord rr = (RunRecord)run        ;

String fileName = "report"+rr.getId()+"_finished.txt";

FileWriter fw = new FileWriter(new File(dir+fileName));
fw.write("Run ID       :"+rr.getId()+"\n");
fw.write("Graph ID     :"+rr.getGraphId()+"\n");
fw.write("Sandbox      :"+rr.getSandbox().getName()+"\n");
fw.write("\n");
fw.write("Start time   :"+rr.getStartTime()+"\n");
fw.write("Stop time    :"+rr.getStopTime()+"\n");
fw.write("Duration     :"+rr.getDurationString()+"\n");
fw.write("Status           :"+rr.getStatus()+"\n");
fw.close();
```

Chapter 29. Manual Task Execution

Since 3.1

A manual task execution allows you to invoke a task directly with an immediate effect, without defining and triggering an event.

There are a number of task types that are usually associated with a triggering event, such as a file listener or a graph/jobflow listener. You can execute any of these tasks manually.

Additionally, you can specify task parameters to simulate a source event that would normally trigger the task. The following is a figure displaying how a 'file event' could be simulated. The parameters for various event sources are listed in the Chapter 27, Graph/Jobflow Parameters (p. 167).

Figure 29.1. Web GUI - "Manual task execution" form

Using Manual Task Execution

In the Server GUI, switch to the **Event Listeners** tab. In the **New Listener** drop-down menu, select the **Manual Task Execution** option.

Choose the task type you would like to use. See documenation on chosen tasks:

Send an Email (p. 172)
Execute Shell Command (p. 175)
Start a Graph (p. 178)
Start a Jobflow (p. 181)
Start a Profiler Job (p. 183)
Abort job (p. 184)
Archive Records (p. 185)
Send a JMS Message (p. 187)
Execute Groovy Code (p. 189)

To access the **Manual Task Execution** form, you need Manual task execution permission (p. 136).

Chapter 30. Scheduling

The scheduling module allows you to create a time schedule for operations you need to trigger in a repetitive or timely manner.

Similar to cron from Unix systems, each schedule represents a separate time schedule definition and a task to perform.

In **Cluster**, you can explicitly specify which node should execute the scheduled task using the Nodes to process (p. 196) parameter. However, if not set, the node will be selected automatically from all available nodes (but always just one).

Figure 30.1. Web GUI - Scheduling

Tasks you can schedule are described in Chapter 28, Tasks (p. 171).

Send an Email (p. 172)
Execute Shell Command (p. 175)
Start a Graph (p. 178)
Start a Jobflow (p. 181)
Start a Profiler Job (p. 183)
Abort job (p. 184)
Archive Records (p. 185)
Send a JMS Message (p. 187)
Execute Groovy Code (p. 189)

Timetable Setting

This section describes specification of triggering schedules. Note that exact trigger times are not guaranteed. There may be couple of seconds delay. Schedule itself can be specified in different ways.

Onetime Schedule

This schedule is triggered just once.

Table 30.1. Onetime schedule attributes

Periodicity	Onetime
Start time	Date and time, specified in the `yyyy-mm-dd hh:mm:ss` format.
Start skipped executions as soon as possible	If checked and execution is skipped for any reason (e.g. server restart), it will be triggered immediately when it is possible. Otherwise it is ignored and it will be triggered at the next scheduled time.

Figure 30.2. Web GUI - onetime schedule form

Periodical schedule by Interval

This type of schedule is the simplest periodical type. Trigger times are specified by these attributes:

Table 30.2. Periodical schedule attributes

Periodicity	Interval
Run every	Specifies interval between two trigger times (in minutes). The next task is triggered even if the previous task is still running.
Active from/to	Date and time, specified in the `yyyy-mm-dd hh:mm:ss` format.
Start skipped executions as soon as possible	If checked and execution is skipped for any reason (e.g. server restart), it will be triggered immediately when it is possible. Otherwise it is ignored and it will be triggered at the next scheduled time.

Figure 30.3. Web GUI - periodical schedule form

Periodical schedule by timetable (cron Expression)

Timetable is specified by a cron expression.

Table 30.3. Cron periodical schedule attributes

Periodicity	Timetable
Cron expression	Cron is a job scheduler which uses its own format for scheduling. i.e. `0 0/2 4-23 * * ?` means "every 2 minutes between 4:00 AM and 11:59 PM".
Active from/to	Date and time, specified in the `yyyy-mm-dd hh:mm:ss` format.
Start skipped executions as soon as possible	If checked and execution is skipped for any reason (e.g. server restart), it will be triggered immediately when it is possible. Otherwise it is ignored and it will be triggered at the next scheduled time.

Figure 30.4. Cron periodical schedule form

When setting up a cron expression, a hint displays it in a human readable format. Furthermore, when you click on each of the field in the expression, the hint expands, indicating which part of the expression you are editing and listing symbols, their meaning and values that can be used in the expression.

Figure 30.5. Editing the cron expression - hours field

![Note icon] **Note**

Server cron expression for *Days of Week* differs from *nix cron expression. Days in cron expression in Server start from 1 which corresponds to Sunday. *nix cron expression uses 0 or 7 for Sunday.

Allocations of Scheduled Task on Nodes

In **Cluster** environment, you can set the node on which the scheduled task will be launched.

If you choose:

- **Any node** - one of the available nodes will be selected automatically.

- **One of selected nodes** - you can select which node will run the task.
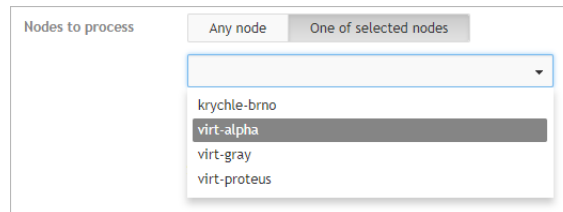
Figure 30.6. Schedule allocation - One ore more specific nodes

Scheduling the Tasks - Examples

Start a graph at specific time

This example shows scheduling the start of a graph at specific time.

1. In the **Scheduling** section of the Server GUI, click on the **New schedule** button.

2. Enter a **Schedule name**.

3. In the **Periodicity** drop-down list, choose **Onetime** to start the graph just once.

4. Enter **Start time**. Use the calendar ▣ to enter the required date and time in a correct format.

5. In the **Task type** field, choose **Start a graph**.

6. Select a **Sandbox** and a **Graph** within the sandbox.

Start a Jobflow once an hour

This example shows scheduling of a periodic task.

Create a new schedule that runs the **UserStats.jbf** jobflow from the **reports** sandbox once an hour.

1. Enter a **Schedule name**.

2. In the **Periodicity** drop-down list, choose **Interval**.

3. Enter the interval between two jobflow starts in the **Run every** field. In the drop-down list, you can choose between seconds, minutes and hours units.

4. The task will be started once an hour within a specified time period. Enter the beginning and end of this period in the **Active from to** fields.

5. Select the **Sandbox** and **Graph**.

Complex Scheduling

This example shows a complex scheduling using a cron expression.

Start a graph **WeekendGraph.grf** every Saturday and Sunday at 22:15.

1. Enter a **Schedule name**.

2. In the **Periodicity** drop-down list, choose **Timetable**.

3. Edit the **Cron expression** field. The expression for every Saturday and Sunday at 22:15 is `0 15 22 ? * 1,7`.

4. In the **Task type** field, choose **Start a graph**.

5. Select the **Sandbox** and **Graph**.

Chapter 31. Viewing Job Runs - Execution History

Execution History shows the history of all jobs that the Server has executed – transformation graphs, jobflows, and Data Profiler jobs. You can use it to find out why a job failed, see the parameters that were used for a specific run, and much more.

The table shows basic information about the job: Run ID, Node, Job file, Executed by, Status, and time of execution. After clicking on a row in the list, you can see additional details of the respective job, such as associated log files, parameter values, tracking and more.

Note

Some jobs might not appear in the Execution History list. These are jobs that have disabled persistency for increased performance (for example, some Launch Services disable storing the run information in order to increase service responsiveness).

Filtering and ordering

Use the Filter panel to filter the view. By default, only parent tasks are shown (Show executions children) – e.g. master nodes in a cluster and their workers are hidden by default.

Use the up and down arrows in the table header to sort the list. By default, the latest job is listed first.

Figure 31.1. Execution History - executions table

When some job execution is selected in the table, the detail info is shown on the right side.

Table 31.1. Persistent run record attributes

Attribute	Description
Run ID	A unique number identifying the run of the job. Server APIs usually return this number as a simple response to the execution request. It is useful as a parameter of subsequent calls for specification of the job execution.
Execution type	A type of a job as recognized by the Server. STANDALONE for graph, JOBFLOW for Jobflow, PROFILER_JOB for profiler, MASTER for the main record of partitioned execution in a cluster, PARTITIONED_WORKER for the worker record of partitioned execution in a cluster
Parent run ID	A run ID of the parent job. Typically the jobflow which executed this job, or master execution which encapsulates this worker execution.
Root run ID	A run ID of the root parent job. Job execution which wasn't executed by another parent job.
Execution group	Jobflow components may group sub-jobs using this attribute. See the description of Jobflow components for details.
Nested jobs	Indication that this job execution has or has not any child execution.
Node	In cluster mode, it shows the ID of the cluster node which this execution was running on.
Executor	If it runs on worker, it contains the text "worker".
Executed by	The user who executed the job. Either directly using some API/GUI or indirectly using the scheduling or event listeners.
Sandbox	The sandbox containing a job file. For jobs which are sent together with an execution request, so the job file doesn't exist on the Server site, it is set to the "default" sandbox.
Job file	A path to a job file, relative to the sandbox root. For jobs which are sent together with an execution request, so the job file doesn't exist on the Server site, it is set to generated string.
Job version	The revision of the job file. A string generated by **CloverDX Designer** and stored in the job file.
Status	Status of the job execution. READY - waiting for execution start, RUNNING - processing the job, FINISHED OK - the job finished without any error, ABORTED - the job was aborted directly using some API/GUI or by the parent jobflow, ERROR - the job failed, N/A (not available) - the server process died suddenly, so it couldn't properly abort the jobs. After restart, the jobs with unknown status are set as N/A
Started	Server date-time (and timezone) of the execution start.
Finished	Server date-time (and timezone) of the execution finish.
Duration	Execution duration
Error in component ID	If the job failed due the error in a component, this field contains the ID of the component.
Error in component type	If the job failed due the error in a component, this field contains type of the component.
Error message	If the job failed, this field contains the error description.
Exception	If the job failed, this field contains error stack trace.
Input parameters	A list of input parameters passed to the job. A job file can't be cached, since the parameters are applied during loading from the job file. The job file isn't cached, by default.
Input dictionary	A list of dictionary elements passed to the job. A dictionary is used independently of job file caching.
Output dictionary	A list of dictionary elements at the moment the job ends.

For jobs which have some children executions, e.g. partitioned or jobflows also an executions hierarchy tree is shown.

Figure 31.2. Execution History - overall perspective

Tip

Since the detail panel, and especially job logs, may be wide, it may be useful to hide a table on the left, so the detail panel spreads. Click on the minimize icon on the top of the list panel to hide the panel. Then to show list panel again, click to the "Expand panel" icon on the left.

Figure 31.3. Executions Hierarchy with docked list of jobs

Executions hierarchy may be rather complex, so it's possible to filter the content of the tree by a fulltext filter. However when the filter is used, the selected executions aren't hierarchically structured.

Tracking

The **Tracking** tab, contains details about the selected job:

Table 31.2. Tracking table information

Attribute	Description
Component ID	The ID of the component.
Component name	The name of the component.
Status	Status of data processing in the respective component. • FINISHED_OK: data was successfully processed • ABORTED: data processing has been aborted • N_A: status unknown • ERROR: an error occurred while data was processed by the component
CPU	CPU usage of the component.
Port	Component's ports (both input and output) that were used for data transfer.
Records	The number of records transferred through the port of the component.
kB	Amount of data transferred in kB.
Records/s	The number of records processed per second
KB/s	Data transfer speed in KB.
Records/s peak	The peak value of **Records/s**.
KB/s peak	The peak value of **KB/s**.

Figure 31.4. Execution History - Tracking

Log File

In the **Log file** tab, you can see the log of the job run with detailed information. A log with a green background indicates a successfully run job, while a red background indicates an error.

You can download the log as a plain text file by clicking **Download log** or as a `zip` archive by clicking **Download log (zipped)**.
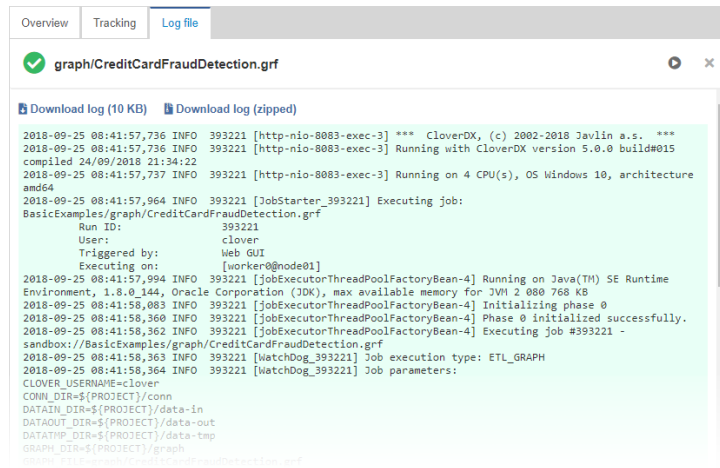
Figure 31.5. Execution History - Tracking

Chapter 32. Listeners

Listeners can be seen as 'hooks'. They wait for a specific event and take a used-defined action if the event occurs.

Created listeners are shown in the **Event Listener** list (see Figure below). The list has the following columns :

Table 32.1. Event Listeners Table Description

Column name	Description
Enabled	Indicates whether the listener is enabled or disabled. Clicking the icon enables/disables the listener.
Name	Shows the name of the listener, task type and event the listener is waiting for.
Last run	Shows the date and time of the listener's last run.
OK	Shows the number of successful runs.
FAIL	Shows the number of failed runs.
...	Click the three vertical dots to open a submenu where you can create an email notification or delete the listener.

Figure 32.1. Listeners

The event is specific to the particular listener

The available actions taken by the listeners are common for all listeners. The actions, that can be taken are:

Graph Event Listeners

Graph Event Listeners allow you to define a task that the Server will execute as a reaction to the success, failure or other event of a specific job (a transformation graph).

Each listener is bound to a specific graph and is evaluated no matter whether the graph was executed manually, scheduled, or via an API call, etc.

You can use listeners to chain multiple jobs (creating a success listener that starts the next job in a row). However, we recommend using Jobflows to automate complex processes because of its better development and monitoring capabilities.

Graph Event Listeners are similar to Jobflow Event Listeners (Jobflow Event Listeners (p. 211)) – for **CloverDX Server** both are simply 'jobs'.

In the Cluster, the event and the associated task are executed on the same node the job was executed on, by default. If the graph is distributed, the task will be executed on the master worker node. However, you can override where the task will be executed by explicitly specifying a Node IDs in the task definition.

Graph Events

Each event carries properties of a graph, which is the source of the event. If there is an event listener specified, the task may use these properties. For example the next graphs in a chain may use "EVENT_FILE_NAME" placeholder which was activated by the first graph in the chain. Graph properties, which are set specifically for each graph run (e.g. RUN_ID), are overridden by the last graph.

Types of graph events

Graph started

The **Graph started** event is created, when an graph execution successfully started.

Graph phase finished

The **Graph phase finished** event is created, everytime a graph phase is finished and all its nodes are finished with status FINISHED_OK.

Graph finished

The **Graph finished** event is created, when all phases and nodes of a graph are finished with FINISHED_OK status.

Graph error

The **Graph error** event is created, when a graph cannot be executed for some reason, or when any node of graph fails.

Graph aborted

The **Graph aborted** event is created, when a graph is explicitly aborted.

Graph timeout

The **Graph timeout** event is created, when a graph runs longer than a specified interval. Thus you should specify the **Job timeout interval** for each listener of a graph timeout event. You can specify the interval in seconds, minutes or hours.

Graph unknown status

The **Graph unknown status** event is created when the Server, during the startup, detects run records with undefined status in the Execution History. Undefined status means, that the Server has been killed during the graph run. The Server automatically changes the state of the graph to *Not Available* and sends a *graph unknown status* event.

Please note that this works just for executions, which have a persistent record in the Execution History. It is possible to execute a transformation without a persistent record in the Execution History, typically for better performance of fast running transformations (e.g. using Launch Services).

Listener

User may create a listener for a specific event type and graph (or all graphs in sandbox). The listener is actually a connection between a graph event and a task, where the graph event specifies *when* and the task specifies *what* to do.

Event handling consists of the following course of actions:

- the event is created
- listeners for this event are notified
- each listener performs the related task

Tasks

Task types are described in Chapter 28, [Tasks](#) (p. 171).

In the Cluster environment, all tasks have an additional attribute **Node IDs** to process the task. If there is no node ID specified, the task may be processed on any cluster node. In most cases, it will be processed on the same node where the event was triggered. If there are some nodeIDs specified, the task will be processed on the first node in the list which is connected in cluster and ready.

[Send an Email](#) (p. 172)
[Execute Shell Command](#) (p. 175)
[Start a Graph](#) (p. 178)
[Start a Jobflow](#) (p. 181)
[Start a Profiler Job](#) (p. 183)
[Abort job](#) (p. 184)
[Archive Records](#) (p. 185)
[Send a JMS Message](#) (p. 187)
[Execute Groovy Code](#) (p. 189)

Use Cases

Possible use cases are:

- [Execute graphs in chain](#) (p. 206)
- [Email notification about graph failure](#) (p. 208)
- [Email notification about graph success](#) (p. 209)
- [Backup of data processed by graph](#) (p. 210)

Execute graphs in chain

For example, we have to execute graph B, only if another graph A finished without any error. So there is a relation between these graphs. We can achieve this behavior by creating a graph event listener. We create a listener for

`graph finished OK` event of graph A and choose an `execute graph` task type with graph B specified for execution. If we create another listener for graph B with the `execute graph` task with graph C specified, it will work as a chain of graphs.

Figure 32.2. The event source graph isn't specified, thus the listener works for all graphs in the specified sandbox

Email notification about graph failure

Figure 32.3. Web GUI - email notification about graph failure

Email notification about graph success

Figure 32.4. Web GUI - email notification about graph success

Backup of data processed by graph

Figure 32.5. Web GUI - backup of data processed by graph

Jobflow Event Listeners

Jobflow Event Listeners allow you to define a task that the Server will execute as a reaction to the success or failure of executing a specific job (a jobflow).

Each listener is bound to a specific jobflow and is evaluated every time the jobflow is executed (no matter whether manually, through another jobflow, via a schedule, API call, etc.).

Jobflow Event Listeners work very similarly to Graph Event Listeners (p. 205) in many ways, since Graphs and Jobflows are both 'jobs' from the point of view of the **CloverDX Server**.

In the Cluster, the event and the associated task are executed on the same node the job was executed on. If the jobflow is distributed, the task will be executed on the master worker node. However, you can override the default setting by explicitly specifying a Node ID in the task definition.

Jobflow Events

Each event carries properties of the event source job. If there is an event listener specified, a task may use these properties. For example, the next job in the chain may use "EVENT_FILE_NAME" placeholder which activated the first job in the chain. Job properties, which are set specifically for each run (e.g. RUN_ID), are overridden by the last job.

Types of jobflow events

Jobflow started

A **Jobflow started** event is created, when jobflow execution successfully started.

Jobflow phase finished

The **Jobflow phase finished** event is created everytime a jobflow phase is finished and all its nodes are finished with the FINISHED_OK status.

Jobflow finished

The **Jobflow finished** event is created, when all phases and nodes of a jobflow are finished with the FINISHED_OK status.

Jobflow error

The **Jobflow error** event is created, when a jobflow cannot be executed for some reason, or when any node of the jobflow fails.

Jobflow aborted

The **Jobflow aborted** event is created, when a jobflow is explicitly aborted.

Jobflow timeout

The **Jobflow timeout** event is created when a jobflow runs longer then the specified interval. Thus you have to specify the **Job timeout interval** for each listener of the jobflow timeout event. You can specify this interval in seconds, minutes or hours.

Jobflow unknown status

The **Jobflow unknown status** event is created, when the Server, during the startup, detects run records with undefined status in the Execution History. Undefined status means, that the Server has been killed during the jobflow run. The server automatically changes the state of the jobflow to *Not Available* and sends the *jobflow status unknown* event.

Please note, that this works just for executions, which have a persistent record in the Execution History. It is possible to execute a transformation without a persistent record in the Execution History, typically for better performance of fast running transformations (e.g. using Launch Services).

Listener

The user may create a listener for the specified event type and jobflow (or all jobflows in sandbox). The listener is actually a connection between the jobflow event and task, where the jobflow event specifies *when* and the task specifies *what* to do.

Event handling consists of the following course of actions:

- event is created
- listeners for this event are notified
- each listener performs the related task

Tasks

A task specifies an operation which should be performed as a reaction to a triggered event.

Task types are described in Chapter 28, Tasks (p. 171).

Note: You can use a task of any type for a jobflow event listener. The description of task types is divided into two sections just to show the most evident use cases.

Send an Email (p. 172)
Execute Shell Command (p. 175)
Start a Graph (p. 178)
Start a Jobflow (p. 181)
Start a Profiler Job (p. 183)
Abort job (p. 184)
Archive Records (p. 185)
Send a JMS Message (p. 187)
Execute Groovy Code (p. 189)

JMS Message Listeners

JMS Message Listeners allow you to listen for incoming JMS messages. You specify the source of the messages (JMS Topic or JMS Queue) and a task that will be executed for each incoming message.

JMS messaging requires a JMS API (jms.jar) and specific third-party libraries. Every one of these libraries must be available on a classpath of an application server. Some application servers contain these libraries by default; however, some do not. In such a case, libraries must be added explicitly before starting the **CloverDX Server**.

JMS Message Listeners on Worker

JMS Message Listeners can be used with Worker as well. In such a case, make sure that the required `.jar` file and libraries are available on the Worker's classpath as well. For more information, see Adding Libraries to the Worker's Classpath (p. 41).

JMS is a complex topic that goes beyond the scope of this document. For more detailed information about JMS, refer to the Oracle website: https://docs.oracle.com/javaee/7/tutorial/jms-concepts.htm#BNCDQ

Note that the JMS implementation is dependent on the application server that the **CloverDX Server** is running in.

In Cluster, you can either explicitly specify which node will listen to JMS or not. If unspecified, all nodes will register as listeners. In the case of JMS Topic, all nodes will get the message and trigger the task (multiple instances) or, in the case of JMS Queue, a random node will consume the message and run the task (just one instance).

Table 32.2. Attributes of JMS message task

Attribute	Description
Initialize by	This attribute is useful only in a cluster environment. It is a node ID where the listener should be initialized. If it is not set, the listener is initialized on all nodes in the cluster. In the Cluster environment, each JMS event listener has a **Node IDs** attribute which may be used to specify the cluster node which will consume messages from the queue/topic. There are the following possibilities: • **No failover:** Just one node ID specified - Only the specified node may consume messages, however the node status must be "ready". When the node isn't ready, messages aren't consumed by any cluster node. • **Failover with node concurrency:** No node ID specified (empty input) - All cluster nodes with status "ready" consume messages concurrently. • **Failover with node reservation:** More node IDs specified (separated by a comma) - Just one of the specified nodes consumes messages at a time. If the node fails for any reason (or its status isn't "ready"), any other "ready" node from the list continues with consuming messages. In a standalone environment, the **Node IDs** attribute is ignored.
JNDI Access	
Initial context	Default or custom
Initial context factory class	A full class name of the `javax.naming.InitialContext` implementation. Each JMS provider has its own implementation. For example, Apache MQ has `org.apache.activemq.jndi.ActiveMQInitialContextFactory`. If it is empty, the Server uses a default initial context. The specified class must be on the web-app classpath or application-server classpath. It is usually included in one library with a JMS API implementation for each specific JMS broker provider.

Attribute	Description
Broker URL	A URL of a JMS message broker
Listen To	
Connection factory	A JNDI name of a connection factory. It depends on a JMS provider.
Username	A username for a connection to a JMS message broker
Password	A password for a connection to JMS message broker
Queue/Topic	A JNDI name of a message queue/topic on the Server
Durable subscriber	If false, the message consumer is connected to the broker as 'non-durable', so it receives only messages which are sent while the connection is active. Other messages are lost.
	If the attribute is true, the consumer is subscribed as 'durable' so it receives even messages which are sent while the connection is inactive. The broker stores such messages until they can be delivered or until the expiration is reached.
	This switch is useful *only for Topics* destinations, because Queue destinations always store messages until they can be delivered or the expiration is reached.
	Please note that consumer is inactive e.g. during server restart and during short moment when the user updates the "JMS message listener" and it must be re-initialized. So during these intervals, the message in the Topic may get lost if the consumer does not have the durable subscription.
	If the subscription is durable, client must have **ClientId** specified. This attribute can be set in different ways in dependence on JMS provider. E.g. for ActiveMQ, it is set as a URL parameter `tcp://localhost:1244?jms.clientID=TestClientID`.
Subscriber name	Available only when **Durable subscriber** is `true`. By default, a durable subscriber name is generated automatically in the `subscr_[clusterNodeId]_[listenerId]` format; therefore, a subscriber has a different name on each cluster node. Using this attribute, you can specify a custom subscriber name that will be identical on all cluster nodes.
Message selector	This query string can be used as a specification of conditions for filtering incoming messages. Syntax is well described on [Java EE API](#) web site. It has different behavior depending on the type of consumer (queue/topic):
	Queue: Messages that are filtered out remain in the queue.
	Topic: Messages filtered out by a Topic subscriber's message selector will never be delivered to the subscriber. From the subscriber's perspective, they do not exist.
Message Processing	
Number of consumers	E.g. 1
Groovy code	A Groovy code may be used for additional message processing and/or for refusing a message. Both features are described below.

Optional Groovy code

Groovy code may be used for additional message processing or for refusing a message.

- **Additional message processing** Groovy code may modify/add/remove values stored in the containers "properties" and "data".

- **Refuse/acknowledge the message** If the Groovy code returns `Boolean.FALSE`, the message is refused. Otherwise, the message is acknowledged. A refused message may be redelivered, however the JMS broker

should configure a limit for redelivering messages. If the Groovy code throws an exception, it's considered a coding error and the JMS message is NOT refused because of it. So, if the message refusal is to be directed by some exception, it must be handled in Groovy.

Table 32.3. Variables accessible in groovy code

type	key	description
javax.jms.Message	msg	instance of a JMS message
java.util.Properties	properties	See below for details. It contains values (String or converted to String) read from a message and it is passed to the task which may then use them. For example, the **execute graph** task passes these parameters to the executed graph.
java.util.Map<String, Object>	data	See below for details. Contains values (Object, Stream, etc.) read or proxied from the message instance and it is passed to the task which may then use them. For example, the **execute graph** task passes it to the executed graph as dictionary entries.
javax.servlet.ServletContext	servletContext	An instance of ServletContext.
com.cloveretl.server.api.ServerFacade	serverFacade	An instance of serverFacade usable for calling CloverDX Server core features.
java.lang.String	sessionToken	SessionToken needed for calling serverFacade methods

Message data available for further processing

A JMS message is processed and the data it contains is stored into two data structures: Properties and Data.

Table 32.4. Properties Elements

key	description
JMS_PROP_[property key]	For each message property, one entry is created where "key" is made of the `JMS_PROP_` prefix and property key.
JMS_MAP_[map entry key]	If the message is an instance of MapMessage, for each map entry, one entry is created where "key" is made of the `JMS_MAP_` prefix and map entry key. Values are converted to String.
JMS_TEXT	If the message is an instance of TextMessage, this property contains content of the message.
JMS_MSG_CLASS	A class name of a message implementation
JMS_MSG_CORRELATIONID	Correlation ID is either a provider-specific message ID or an application-specific String value
JMS_MSG_DESTINATION	The JMSDestination header field contains the destination to which the message is being sent.
JMS_MSG_MESSAGEID	JMSMessageID is a String value that should function as a unique key for identifying messages in a historical repository. The exact scope of uniqueness is provider-defined. It should at least cover all messages for a specific installation of a provider, where an installation is some connected set of message routers.
JMS_MSG_REPLYTO	A destination to which a reply to this message should be sent.
JMS_MSG_TYPE	A message type identifier supplied by the client when the message was sent.
JMS_MSG_DELIVERYMODE	The DeliveryMode value specified for this message.
JMS_MSG_EXPIRATION	The time the message expires, which is the sum of the time-to-live value specified by the client and the GMT at the time of the send.
JMS_MSG_PRIORITY	The JMS API defines ten levels of priority value (0 = lowest, 9 = highest). In addition, clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority.
JMS_MSG_REDELIVERED	If this message is being redelivered.
JMS_MSG_TIMESTAMP	The time a message was handed off to a provider to be sent. It is not the time the message was actually transmitted, because the actual send may occur later due to transactions or other client-side queueing of messages.

Note that all values in the "Properties" structure are stored as a String type – however they are numbers or text.

For backwards compatibility, all listed properties can also be accessed using lower-case keys; however, it is a deprecated approach.

Table 32.5. "Data" elements

key	description
JMS_DATA_MSG	An instance of javax.jms.Message.
JMS_DATA_STREAM	An instance of java.io.InputStream. Accessible only for TextMessage, BytesMessage, StreamMessage, ObjectMessage (only if a payload object is an instance of String). Strings are encoded in UTF-8.
JMS_DATA_TEXT	An instance of String. Only for TextMessage and ObjectMessage, where a payload object is an instance of String.
JMS_DATA_OBJECT	An instance of java.lang.Object - message payload. Only for ObjectMessage.

The **Data** container is passed to a task that can use it, depending on its implementation. For example, the task **execute graph** passes it to the executed graph as dictionary entries.

In a Cluster environment, you can explicitly specify node IDs, which can execute the task. However, if the data payload is not serializable and the receiving and executing node differ, an error will be thrown as the Cluster cannot pass the data to the executing node.

Inside a graph or a jobflow, data passed as dictionary entries can be used in some component attributes. For example, the **File URL** attribute would look like: `"dict:JMS_DATA_STREAM:discrete"` for reading the data directly from the incoming JMS message using a proxy stream.

Note

> If the graph is **executed on Worker**, the dictionary entries must be **serialized**; otherwise, they cannot be passed to the graph.

For backwards compatibility, all listed dictionary entries can also be accessed using lower-case keys; however, it is a deprecated approach.

Universal Event Listeners

Since 2.10

Universal Event Listeners allow you to write a piece of Groovy code that controls when an event is triggered, subsequently executing a predefined task. The Groovy code is periodically executed and when it returns TRUE, the task is executed.

Table 32.6. Attributes of Universal message task

Attribute	Description
Node IDs to handle the event	In a Cluster environment, each universal event listener has a **Node IDs** attribute which may be used to specify which cluster node performs the Groovy code. There are following possibilities: • **No failover:** Just one node ID specified - Only the specified node performs the Groovy code, however node status must be "ready". When the node isn't ready, the code isn't performed at all. • **Failover with node concurrency:** No node ID specified (empty input) - All cluster nodes with the status "ready" concurrently perform the Groovy code. So the code is executed on each node in the specified interval. • **Failover with node reservation:** More node IDs specified (separated by a comma) - Just one of the specified nodes performs the Groovy code. If the node fails for any reason (or its status isn't "ready"), any other "ready" node from the list continues with periodical Groovy code processing. In a standalone environment, the **Node IDs** attribute is ignored.
Interval of check in seconds	Periodicity of Groovy code execution.
Groovy code	Groovy code that evaluates either to TRUE (execute the task) or FALSE (no action). See below for more details.

Groovy code

A piece of Groovy is repeatedly executed and evaluated; based on the result, the event is either triggered and the task executed or no action is taken.

For example, you can continually check for essential data sources before starting a graph. Or, you can do complex checks of a running graph and, for example, decide to kill it if necessary. You can even call the **CloverDX Server** Core functions using the ServerFacade interface, see Javadoc: http://host:port/clover/javadoc/index.html

Evaluation Criteria

If the Groovy code returns Boolean.TRUE, the event is triggered and the associated task is executed. Otherwise, nothing happens.

If the Groovy code throws an exception, it is considered a coding error and the event is NOT triggered. Thus, exceptions should be properly handled in the Groovy code.

Table 32.7. Variables accessible in Groovy code

type	key	description
java.util.Properties	properties	An empty container which may be filled with String-String key-value pairs in your Groovy code. It is passed to the task which may use them somehow. For example, the task **execute graph** passes these parameters to the executed graph.
java.util.Map<String, Object>	data	An empty container which may be filled with String-Object key-value pairs in your Groovy code. It is passed to the task which may use them somehow according to its implementation - e.g. the task **execute graph** passes it to the executed graph as dictionary entries. Note that it is not serializable, thus if the task is relying on it, it can be processed properly only on the same cluster node.
javax.servlet.ServletContext	servletContext	An instance of ServletContext in which **CloverDX Server** is running.
com.cloveretl.server.api.ServerFacade	serverFacade	An instance of serverFacade usable for calling **CloverDX Server** core features.
java.lang.String	sessionToken	A sessionToken needed for calling methods on the serverFacade.

File Event Listeners (remote and local)

Local file-system changes: Since 1.3

Remote file-system changes: Since 4.2

File Event Listeners allow you to monitor changes on a specific local file system path or remote URL – for example, new files appearing in a folder – and react to such an event with a predefined task.

You can either specify an exact file name or use a wildcard or regexp, then set a checking interval in seconds and define a task to process the event.

There is a global minimum check interval that you can change if necessary in the configuration (the `clover.event.fileCheckMinInterval` property). See Chapter 15, List of Configuration Properties (p. 82).

Figure 32.6. Web GUI - creating a File Event listener

Table 32.8. Parameters passed from the listener to the task

Parameter	Description
EVENT_USERNAME	The name of the user who caused the event.
EVENT_USER_ID	A numeric ID of the user who caused the event.
EVENT_FILE_NAME	The name of the file (without the path or URL) that triggered the event. Present only when **Trigger one task for multiple changed files** is disabled.
EVENT_FILE_PATH	A resolved (without placeholders) path to the observed directory on the local filesystem. Valid only for a local file listener.
EVENT_FILE_PATTERN	A filename pattern.
EVENT_FILE_EVENT_TYPE	The type of the file event. Possible values: SIZE, CHANGE_TIME, APPEARANCE, DISAPPEARANCE.
EVENT_FILE_LISTENER_ID	An ID of the listener which triggered the event.
EVENT_FILE_URLS	Full URLs to access the files, e.g. in the File URL attribute of components. If **Trigger one task for multiple changed files** is enabled and there are multiple URLs, they are separated by a separator specified by **CloverDX** Engine property DEFAULT_PATH_SEPARATOR_REGEX.
EVENT_FILE_AUTH_USERNAME	A username/ID to the remote location.
EVENT_FILE_AUTH_USERNAME_URL_ENCODED	The same as EVENT_FILE_AUTH_USERNAME, but the value is also URL encoded, so it may be used in a URL.
EVENT_FILE_AUTH_PASSWORD	a password/key to the remote location. It is encrypted by the master password.
EVENT_FILE_AUTH_PASSWORD_URL_ENCODED	The same as EVENT_FILE_AUTH_PASSWORD, but the value is also URL encoded, so it may be used in a URL.

Cluster environment

In a Cluster environment, each file event listener has a **Node IDs** attribute which may be used to specify which cluster node will perform the checks on its local file system. There are following possibilities:

- **No failover:** Just one node ID specified - Only the specified node observes the local/remote filesystem; however, the node status must be "ready". When the node isn't ready, the file system isn't checked at all.

  To create a file event listener with no failover, select **One of selected nodes** in **Initialize by** and select one node from the table below.

- **Failover with node concurrency::** No node ID specified (empty input) - All cluster nodes with the status "ready" concurrently check the local/remote filesystem according to file event listener attributes settings. In this mode, when the listener is configured to observe the local filesystem, each cluster node observes its own local file system. So it's useful only when the observed path is properly shared among the cluster nodes. It may behave unpredictably otherwise. On the other hand, when the listener is configured to observe the remote filesystem, listeners running on different cluster nodes may connect to the same remote resource. The nodes use a locking mechanism when accessing the local or remote filesystem, so no conflict between listeners running concurrently on different nodes can occur.

  To create file event listener with node cuncurrency, select **Any node** in **Initialize by**.

- **Failover with node reservation:** More node IDs specified (separated by comma) - Just one of the specified nodes checks its filesystem. If the node fails for any reason (or its status isn't "ready"), any other "ready" node

from the list continues with checking. Please note, that when file event listener is re-initialized on another cluster node, it compares the last directory content detected by the failed node with the current directory content.

To create a file event listener with node reservation, select **One of selected nodes** in **Initialize by** and select more nodes.

In a standalone environment, the **Node IDs** attribute is ignored.

Supported filesystems and protocols

Local filesystem

The user may specify a path to the directory which the listener shall observe. The listener doesn't read the directory content recursively. The directory must exist.

If the listener can run concurrently on more nodes, the directory must be shared among all these nodes and the directory must exist on all these nodes. In a cluster environment, the directory must exist on each cluster node where the listener may run.

It is recommended to use placeholders to unify the configuration on all nodes. The recommended placeholders are: **CloverDX Server** config property `${sandboxes.home}` and JVM system property `${java.io.tmpdir}`. It is possible to use any JVM system property or Environment variable.

Figure 32.7. File available on local file system

Remote filesystem

The user may specify a URL to the directory which the listener shall observe. The supported protocols are: FTP, S3, SFTP and SMB. Different protocols may use different authentication methods: none, username+password and keystore. The listener doesn't read the directory content recursively. The directory must exist.

Figure 32.8. File available on remote location

Currently the subset of the protocols allowed by file-operations is supported:

• **FTP - File Transfer Protocol** (no authentication or username+password authentication) URL example:

```
ftp://host:23/observed/path/
```

- **SFTP (SSH/FTP) - SSH File Transfer Protocol** (username+private key authentication) URL example:

```
sftp://host:23/observed/path/
```

It is recommended to use placeholders to unify the path configuration on all nodes. The recommended placeholders are: **CloverDX Server** config property `${sandboxes.home}`, JVM system property `${user.home}`. It is possible to use any JVM system property or Environment variable.

- **S3 - Amazon S3 Storage** (AWS Access Key ID + Secret Access Key authentication) URL example:

```
s3://s3.amazonaws.com/bucketname/path/
```

Please specify the AWS Access Key ID as a username and Secret Access Key as a password.

- **Microsoft SMB/CIFS Protocol** (username+password authentication) URL example:

```
smb://host/path/
```

- **Microsoft SMBv2/v3 Protocol** (username+password authentication) URL example:

```
smb2://host/path/
```

Observed file

The local observed file is specified by a directory path and file name pattern.

The remote observed file is specified by a URL, credentials and file name pattern.

The user may specify just one exact file name or file name pattern for observing more matching files in specified directory. If there are more changed files matching the pattern, separated event is triggered for each of these files.

There are three ways how to specify file name pattern of observed file(s):

- <u>Exact match</u> (p. 223)
- <u>Wildcards</u> (p. 223)
- <u>Regular expression</u> (p. 224)

Exact match

You specify the exact name of the observed file.

Wildcards

You can use wildcards common in most operating systems (*, ?, etc.)

- `*` - Matches zero or more instances of any character
- `?` - Matches one instance of any character
- `[...]` - Matches any of characters enclosed by the brackets
- `\` - Escape character

Examples

- `*.csv` - Matches all CSV files

- `input_*.csv` - Matches i.e. input_001.csv, input_9.csv

- `input_???.csv` - Matches i.e. input_001.csv, but does not match input_9.csv

Regular expression

Examples

- `(.*?)\.(jpg|jpeg|png|gif)$` - Matches image files

Notes

- It is strongly recommended to use absolute paths with placeholders. It is possible to use a relative path, but the working directory depends on an application server.
- Use forward slashes as file separators, even on MS Windows. Backslashes might be evaluated as escape sequences.

File Events

For each listener you have to specify event type, which you are interested in.

Please note that since **CloverETL 4.2**, the grouping mode may be enabled for the file listener, so all file changes detected by a single check produce just one 'grouped' file event. Otherwise each single file produces its own event.

There are four types of file events:

- File added (p. 224)
- File removed (p. 224)
- File size changed (p. 224)
- File timestamp changed (p. 224)

File added

Event of this type occurs, when the observed file is created or copied from another location between two checks. Please keep in mind that event of this type occurs immediately when a new file is detected, regardless if it is complete or not. Thus task which may need a complete file is executed when the file is still incomplete. Recommended approach is to save the file to a different location and when it is complete, rename it or move it to an observed location where **CloverDX Server** may detect it. File moving/renaming should be an atomic operation.

An event of this type does not occur when the file has been updated (change of timestamp or size) between two checks. Appearance means that the file didn't exist during the previous check and it exists now, during the current check.

File removed

Event of this type occurs, when observed file is deleted or moved to another location between two checks.

File size changed

Event of this type occurs when the size of the observed file has changed between two checks. Event of this type is never produced when the file is created or removed. The file must exist during both checks.

File timestamp changed

Event of this type occurs, when timestamp of the observed file has changed between two checks. Event of this type is never produced when the file is created or removed. The file must exist during both checks.

Check Interval, Task and Use Cases

- The user may specify the minimum time interval between two checks. Use the **Check every** field to specify the interval in seconds.

- Each listener defines a task which will be processed as a reaction to the file event. All task types and theirs attributes are described in the Scheduling (p. 192) and Graph Event Listeners (p. 205) sections.

  - Graph Execution when a file with input data is accessible.

  - Graph Execution when a file with input data is updated.

  - Graph Execution when a file with generated data is removed and must be recreated.

How to use source of event during task processing

A file(s) which caused the event (considered as a source of the event) may be used during task processing. **CloverDX** graph/jobflow components with the **File URL** attribute (e.g. reader or writer components) may directly use an event source by parameter placeholder: ${EVENT_FILE_URLS}. For another parameters, see Executed by Task Graph Execution by File Event Listener (p. 169).

Note that previous versions used lower-case placeholders. Since version 3.3, placeholders are upper-case, however lower-case still work for backward compatibility.

For **graph execution** task this works only if the graph is not pooled. Thus keep in pool interval must be set to 0 (default value).

Delayed triggering for incomplete files

It is possible to delay task execution for incomplete files. This is useful in cases when the condition to execute the listener's task has been met, but there is still some work that needs to be done on the file, e.g. the whole file needs to be uploaded.

Ignore empty files
>

If the process creating the file creates an empty file, then switch to different task for several minutes or even hours and finally writes the content, tick **Ignore empty files** checkbox. The task will be triggered only if a non-empty file appears.

Trigger task when file has not changed for n checks

If the file size slowly rises until the file is complete, tick the checkbox **Trigger task when file has not changed**. Then specify the number of additional file size checks that are to be performed on the file. The listener's task will not be triggered until the checks are performed and the file's size stays the same between these checks.

Combination

If you use **Ignore empty files** and **Trigger task when file has not changed for n checks** together, the first one filters out empty files and the latter one the files that are being changed. The task will be triggered only on files that are not empty and that have not changed for the specified number of checks.

Howtos

Create a file event listener listening to changes on local file system

This howto shows a way to create a new listener checking appearance of a file (new_invoices.txt) on a local file system (/mnt/sdb2/). The appearance will trigger a graph (graph/checkInvoices.grf) from the **InvoicesProcessing** sandbox.

In **Event Listeners →File Event Listeners**, click **New Listener**.

Enter the **Name** of the listener, e.g. **Invoices**.

Enter the **Path** to the directory where files will appear: /mnt/sdb2. You can check that **CloverDX** can access this directory (the directory exists and permissions are set up properly) with the **Validate Accesibility** button.

If the observed directory becomes inaccessible, **CloverDX Server** can send you an email. To do so, tick **Send email on check failure** and enter recipient(s).

The event should be triggered on file appearance - set **Type of check** to **File added**.

Enter the file name new_invoices.txt to **Filename pattern**.

If the file is created empty, but the content is written after some time, tick **Ignore empty files**. Doing so, the task will be executed after the file contains some data.

If it takes a long time to copy the whole file to the observed position, the **CloverDX Server** can perform several check to ensure that the file to process is not to be changed. Tick **Trigger task when file has not changed for** and enter the number of checks. If you tick **Ignore empty files**, this checks will be performed after the file is not empty.

Choose **Sandbox** with the graph (**InvoicesProcessing**) and the **graph** (**graph/checkInvoices.grf**).

To save the changes, click on the **Create** button.

Observe file from one cluster node

Create the listener in the same way as on the Server.

Switch **Initialize by** to **One of selected nodes**.

Add the particular node(s) from **Available nodes** to **Selected nodes**.

Quickly setup failure notification

To create a notification for when the file event listener fails, click on the **Create notification** button. Pressing the button opens up a popup dialog where email addresses can be entered separated by commas.

The entered email addresses are remembered and pre-filled the next time the button is pressed. If the popup is closed with invalid email addresses entered, the field is cleared.

When creating the notification, a Task Failure Listener is created with an email task listening to the selected File Event Listener. The first entered email address will be used as the Reply-to(Sender) address. The subject and body of the email is as predefined by the Task Failure template. The trigger limit is set to 5.

Editing failure notification

If there is a Task Failure Listener listening to given File Event Listener then instead of the **Create Notification** button a **Notification Detail** button is displayed. This button redirects to the Task Failure Listener page and shows

the details of the Task Failure Listener listening to the File Event Listener. If more than one Task Failure Listeners are listening to the File Event Listener, then the details of the first one is shown.

Quickly enable or disable file event listener

In **Event Listeners** →**File Event Listeners**, there is a table with event listeners. In this table, click the icon in the **Enabled** column.

Pasting URL

The whole URL including user name and password can be pasted at once. Click **Paste URL** and paste the string.

If the name or password in URL contain special characters, e.g. +, the special characters should be encoded: `ftp://anonymous:test%2B@example.com/dir/file.txt`

Note: use encoding accepted by the `java.net.URLDecoder.decode()` function.

Task Failure Listeners

Since 4.4

Task Failure Listeners allow you to detect and react to failures in your server when a task you set up in a listener fails to execute, e.g. a File Listener is set up to detect changes on an FTP server, but it fails to connect to the FTP server.

Task Failure Listeners do not detect failures of the task itself, e.g. a File Listener is set up to detect changes on an FTP server and send an email if the change is detected. If the File Listener fails to send the email for some reason, the Task Failure Listener won't detect it.

The same tasks to be executed are available as with all the other listeners, the difference is that when creating a new Task Failure Listener the pre-selected task is **Sending an email** if the email service is configured in Configuration.

Figure 32.9. Web GUI - creating a Task Failure listener

Task Choice

There are three options to choose from: Listening to any task failure, listening to failures from a group (such as File Event Listeners) or listening to a failure of a chosen listener.

Selecting an option from the **Listen to type** menu restricts the **Listen to** combobox to event sources of the chosen category. If there are no event sources in the category, the Failure Listener can still be created, it will react to failures of tasks that will be created in that category.

When selecting an event source, you can type into the text field to filter the dropdown menu. After selecting an event source, some information is presented about the listener.

When sending an email, you can use the 'task failed' template by selecting it from the **E-mail template** dropdown menu.

Task Failed E-mail Template

The default email template may be modified using placeholders described in Placeholders (p. 173) and parameters in Table 32.9, "Parameters usable in task failure email template" (p. 229). Furthermore, some additional parameters can be used if the failed listener is a File Event Listener, see Table 32.10, "File Event Listener specific parameters usable in task failure email template" (p. 229).

Table 32.9. Parameters usable in task failure email template

Parameter	Description
TASK_LISTENER_ID	The ID of the failed listener.
TASK_LISTENER_NAME	The name of the failed listener.
TASK_LISTENER_TYPE	The type of the failed listener.
TASK_LISTENER_TYPE_TEXT	The full name of the failed listener's type.
TASK_LISTENER_OWNER_USERNAME	The owner of the failed listener.

Table 32.10. File Event Listener specific parameters usable in task failure email template

Parameter	Description
FILE_EVENT_LISTENER_PATH	The path of the observed directory.
FILE_EVENT_LISTENER_REMOTE_URL	The full remote URL of the observed directory.
FILE_EVENT_LISTENER_FILENAME_PATTERN	The filename pattern of files the listener observes.
FILE_EVENT_LISTENER_CHECK_TYPE	The type of check the listener performs.
FILE_EVENT_LISTENER_MATCH_TYPE	The listener's filename match type.
FILE_EVENT_LISTENER_NODES	The list of nodes the listener can be intitalized by.

Chapter 33. Recommendations for Transformations Developers

Add external libraries to app-server classpath

Connections (JDBC/JMS) may require third-party libraries. We strongly recommend adding these libraries to the app-server classpath.

CloverDX allows you to specify these libraries directly in a graph definition so that **CloverDX** can load these libraries dynamically. However, external libraries may cause memory leak, resulting in `java.lang.OutOfMemoryError: PermGen space`, in this case.

In addition, app-servers should have the JMS API on their classpath – and the third-party libraries often bundle this API as well. So it may result in classloading conflicts if these libraries are not loaded by the same classloader.

Another graphs executed by RunGraph component may be executed only in the same JVM instance

In the server environment, all graphs are executed in the same VM instance. The attribute **same instance** of the **RunGraph** component cannot be set to false.

Chapter 34. Extensibility - CloverDX Engine Plugins

Since 3.1.2

The **CloverDX Server** can use external engine plugins loaded from a specified source. The source is specified by `engine.plugins.additional.src` config property.

See details about the possibilities with **CloverDX** configuration in

This property must be the absolute path to the directory or zip file with additional **CloverDX** engine plugins. Both the directory and zip file must contain a subdirectory for each plugin. These plugins are not a substitute for plugins packed in a WAR file. Changes in the directory or the ZIP file apply only when the Server is restarted.

Each plugin has its own class-loader that uses a parent-first strategy by default. The parent of plugins' classloaders is web-app classloader (content of [WAR]/WEB-INF/lib). If the plugin uses any third-party libraries, there may be some conflict with libraries on the parent-classloaders classpath. These are common exceptions/errors suggesting that there is something wrong with classloading:

- java.lang.ClassCastException
- java.lang.ClassNotFoundException
- java.lang.NoClassDefFoundError
- java.lang.LinkageError

There are sevral ways you can get rid of such conflicts:

- Remove your conflicting third-party libraries and use libraries on parent classloaders (web-app or app-server classloaders)

- Use a different class-loading strategy for your plugin.

  - In the plugin descriptor `plugin.xml`, set attribute `greedyClassLoader="true"` in the element `plugin`

  - It means that the plugin classloader will use a self-first strategy

- Set an inverse class-loading strategy for selected Java packages.

  - In the plugin descriptor `plugin.xml`, set attribute `excludedPackages` in the element `plugin`.

  - It is a comma-separated list of package prefixes – for example: `excludedPackages="some.java.package,some.another.package"`

  - In the previous example, all classes from `some.java.package`, `some.another.package` and all their sub-packages would be loaded with the inverse loading strategy, then the rest of classes on the plugins classpath.

The suggestions above may be combined. Finding the best solution for these conflicts may depend on the libraries on app-server classpath.

For more convenient debugging, it is useful to set a TRACE log level for related class-loaders.

```
<logger name="org.jetel.util.classloader.GreedyURLClassLoader">
        <level value="trace"/>
</logger>
<logger name="org.jetel.plugin.PluginClassLoader">
        <level value="trace"/>
</logger>
```

See Chapter 17, Logging (p. 103) for details about overriding a server log4j configuration.

Chapter 35. Troubleshooting

Graph hangs and is un-killable

A graph can sometimes hang and be un-killable if some network connection in it hangs. Set a shorter `tcp-keepalive` so that the connection times out earlier. The default value on Linux is 2 hours (7,200 seconds). You can set it to 10 minutes (600 seconds).

See [Using TCP keepalive under Linux](#).

The file descriptor can be closed manually using `gdb`. See [How to close file descriptor via Linux shell command](#).

SSL/TLS Issues

SSL-related Failures on WebLogic 12

Certain graphs using SSL-encrypted connections may fail on WebLogic 12 due to damaged library distributed with this application server. The issue can be identified by a SHA-1 digest error in the graph execution stacktrace:

```
...
Caused by: java.io.IOException: Could not convert socket to TLS
    at com.sun.mail.pop3.Protocol.stls(Protocol.java:659)
    at com.sun.mail.pop3.POP3Store.getPort(POP3Store.java:269)
    at com.sun.mail.pop3.POP3Store.protocolConnect(POP3Store.java:207)
Caused by: javax.net.ssl.SSLException: java.lang.SecurityException:
    SHA1 digest error for org/bouncycastle/jce/provider/JCEECPublicKey.class
...
```

To fix the issue, replace the library `[MW_HOME]/oracle_common/modules/bcprov-jdk16-1.45.jar` with the one downloaded directly from Bouncy Castle home page. Restart the application server to load the new library.

Graph run in Worker is Slow

It may be caused by slow data storage. Use `vmstat`, e.g. `vmstat 1 30`. If you see high values under `io/bi` or `io/bo` columns, it might be that case. Another tool to confirm or disconfirm slow data storage as possible cause is `iotop`.

Part VI. API

Chapter 36. Simple HTTP API

The Simple HTTP API is a basic Server automation tool that lets you control the Server from external applications using simple HTTP calls.

Most of operations is accessible using the HTTP GET method and return plain text. Thus, both `request` and `response` can be conveniently sent and parsed using very simple tools (wget, grep, etc.).

If global security is on (on by default), the Basic HTTP authentication is used. Authenticated operations will require valid user credentials with corresponding permissions.

Note that the Graph-related operations `graph_run`, `graph_status` and `graph_kill` also work for jobflows and Data Profiler jobs.

The generic pattern for a request URL:

```
http://[domain]:[port]/[context]/[servlet]/[operation]?[param1]=[value1]&[param2]=[value2]...
```

example: `http://localhost:8080/clover/simpleHttpApi/help`

> **Note**
>
> For backward compatibility, you can also use `http://localhost:8080/clover/`**`request_processor`**`/help`.

CSRF Protection

The Simple HTTP API provides protection against Cross-Site Request Forgery (CSRF) attacks. An example of such an attack is a case where the user is logged into the Server Console, and an attacker sends him a link to the Simple HTTP API such that it runs a graph. Clicking on such a link would call the Simple HTTP API and re-use the session of the logged-in user. There are also more complex variants of the attack that are harder to detect by the user.

The protection against such an attack is that the Simple HTTP API requires the presence of the `X-Requested-By` header in the HTTP request. Value of the header can be arbitrary (it is not checked). Such a header cannot be set by CSRF attack vectors, i.e. by clicking on a link in an email.

Examples of calling the API with the `X-Requested-By` header:

```
curl --header "X-Requested-By: arbitrary_value"  http://user:password@hostname:port/clover/simpleHttpApi/graph_run
```

```
wget --header "X-Requested-By: arbitrary_value" --user=$USER --password=$PASS -O ./$OUTPUT_FILE $REQUEST_URL
```

The CSRF protection of Simple HTTP API can be disabled via the security.csrf.protection.enabled (p. 85) configuration property. It is enabled by default. If the protection is disabled, it is not necessary to set the `X-Requested-By` header.

The Server Console's page for testing the Simple HTTP API uses a different CSRF protection mechanism. The requests contain a `csrftoken` parameter. This is intended for usage only in the testing page.

List of Operations

- Operation help (p. 235)

The HTTP API is enabled by default. You can disable it with the configuration property `http.api.enabled`. In the Server GUI, switch to **Configuration** →**Setup** and add the following line

```
http.api.enabled=false
```

to the properties file.

Operation help

parameters

no

returns

a list of possible operations and parameters with its descriptions

example

```
http://localhost:8080/clover/simpleHttpApi/help
```

Operation graph_run

Call this operation to start an execution of the specified job. The operation is called graph_run for backward compatibility, however it may execute a graph, jobflow or profiler job.

parameters

Table 36.1. Parameters of graph_run

parameter name	mandatory	default	description
graphID	yes	-	A file path to the job file, relative to the sandbox root.
sandbox	yes	-	Text ID of sandbox.
additional job parameters	no		Any URL parameter with the `param_` prefix is passed to the executed job and may be used in transformation XML as a placeholder, but without the `param_` prefix. e.g. `param_FILE_NAME` specified in URL may be used in the XML as `${FILE_NAME}`. These parameters are resolved only during loading of XML, so it cannot be pooled.
additional config parameters	no		URL parameters prefixed with `config_` can set some of the execution parameters. For graphs, the following parameters are supported: • `config_skipCheckConfig` - when set to `false`, graph configuration will be checked before the execution. • `config_logLevel` - log level of the executed graph, one of OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, ALL. • `config_clearObsoleteTempFiles` - when set to `true`, temp files of previous runs of this graph will be deleted before the execution. • `config_debugMode` - when set to `true`, debug mode for a given graph will be enabled. For more information, see Job Config Properties (p. 150).
nodeID	no	-	In cluster mode, it is the ID of a node which should execute the job. However it is not final. If the graph is distributed or the node is disconnected, the graph may be executed on another node.
verbose	no	MESSAGE	MESSAGE | FULL - how verbose should possible error message be.

returns

run ID: incremental number, which identifies each execution request

example

```
http://localhost:8080/clover/simpleHttpApi/graph_run?graphID=graph/graphDBExecute.grf&sandbox=mva
```

Operation graph_status

Call this operation to obtain a status of a specified job execution. The operation is called graph_status for backward compatibility; however, it may return status of a graph or jobflow.

parameters

Table 36.2. Parameters of graph_status

parameter name	mandatory	default	description
runID	yes	-	Id of each graph execution
returnType	no	STATUS	STATUS | STATUS_TEXT | DESCRIPTION | DESCRIPTION_XML
waitForStatus	no	-	Status code which we want to wait for. If it is specified, this operation will wait until the graph is in the required status.
waitTimeout	no	0	If `waitForStatus` is specified, it will wait only for the specified amount of milliseconds. Default 0 means forever, but it depends on an application server configuration. When the specified timeout expires and graph run still isn't in a required status, the server returns code 408 (Request Timeout). 408 code may be also returned by an application server if its HTTP request timeout expires before.
verbose	no	MESSAGE	MESSAGE | FULL - how verbose should possible error message be.

returns

Status of a specified graph. It may be a number code, text code or a complex description in dependence on the optional parameter `returnType`. Description is returned as a plain text with a pipe as a separator, or as XML. A schema describing XML format of the XML response is accessible on **CloverDX Server** URL: `http://[host]:[port]/clover/schemas/executions.xsd` Depending on the `waitForStatus` parameter, it may return a result immediately or wait for a specified status.

example

```
http://localhost:8080/clover/simpleHttpApi/graph_status ->
          -> ?runID=123456&returnType=DESCRIPTION&waitForStatus=FINISHED&waitTimeout=60000
```

Operation graph_kill

Call this operation to abort/kill a job execution. The operation is called graph_kill for backward compatibility, however it may abort/kill a graph, jobflow or profiler job.

parameters

Table 36.3. Parameters of graph_kill

parameter name	mandatory	default	description
runID	yes	-	The ID of each graph execution
returnType	no	STATUS	STATUS | STATUS_TEXT | DESCRIPTION
verbose	no	MESSAGE	MESSAGE | FULL - how verbose should possible error message be.

returns

The status of the specified graph after an attempt to kill it. It may be a number code, text code or a complex description in dependence on optional parameter.

example

```
http://localhost:8080/clover/simpleHttpApi/graph_kill?runID=123456&returnType=DESCRIPTION
```

Operation server_jobs

parameters

no

returns

a list of runIDs of currently running jobs.

example

```
http://localhost:8080/clover/simpleHttpApi/server_jobs
```

Operation sandbox_list

parameters

no

returns

List of all sandbox text IDs. In the next versions, it will return only accessible ones.

example

```
http://localhost:8080/clover/simpleHttpApi/sandbox_list
```

Operation sandbox_content

parameters

Table 36.4. Parameters of sandbox_content

parameter name	mandatory	default	description
sandbox	yes	-	text ID of sandbox
verbose	no	MESSAGE	MESSAGE | FULL - how verbose should possible error message be.

returns

A list of all elements in the specified sandbox. Each element may be specified as a file path relative to the sandbox root.

example

```
http://localhost:8080/clover/simpleHttpApi/sandbox_content?sandbox=mva
```

Operation executions_history

parameters

Table 36.5. Parameters of executions_history

parameter name	mandatory	default	description
sandbox	yes	-	The text ID of a sandbox.
from	no		Lower datetime limit of start of execution. The operation will return only records after (and equal to) this datetime. Format: "yyyy-MM-dd HH:mm" (must be URL encoded).
to	no		The upper datetime limit of start of execution. The operation will return only records before (and equal to) this datetime. Format: "yyyy-MM-dd HH:mm" (must be URL encoded).
stopFrom	no		The lower datetime limit of stop of execution. The operation will return only records after (and equal to) this datetime. Format: "yyyy-MM-dd HH:mm" (must be URL encoded).
stopTo	no		The upper datetime limit of stop of execution. The operation will return only records before (and equal to) this datetime. Format: "yyyy-MM-dd HH:mm" (must be URL encoded).
status	no		Current execution status. The operation will return only records with specified `STATUS`. The values are `RUNNING` | `ABORTED` | `FINISHED_OK` | `ERROR`
sandbox	no		Sandbox code. The operation will return only records for graphs from a specified sandbox.
graphId	no		The text Id, which is unique in a specified sandbox. The file path is relative to the sandbox root.
orderBy	no		An attribute for list ordering. Possible values: `id` | `graphId` | `status` | `startTime` | `stopTime`. By default, there is no ordering.
orderDescend	no	true	A switch which specifies ascending or descending ordering. If true (default), ordering is descending.
returnType	no	IDs	Possible values are: `IDs` | `DESCRIPTION` | `DESCRIPTION_XML`
index	no	0	an index of the first returned records in a whole record set. (starting from
records	no	infinite	The maximum amount of returned records.
verbose	no	MESSAGE	MESSAGE | FULL - how verbose should possible error message be.

returns

List of executions according to filter criteria.

For `returnType==IDs` returns a simple list of runIDs (with new line delimiter).

For `returnType==DESCRIPTION` returns complex response which describes current status of selected executions, their phases, nodes and ports.

```
execution|[runID]|[status]|[username]|[sandbox]|[graphID]|[startedDatetime]|[finishedDatetime]|[clusterNode]|[grap
phase|[index]|[execTimeInMilis]
node|[nodeID]|[status]|[totalCpuTime]|[totalUserTime]|[cpuUsage]|[peakCpuUsage]|[userUsage]|[peakUserUsage]
port|[portType]|[index]|[avgBytes]|[avgRows]|[peakBytes]|[peakRows]|[totalBytes]|[totalRows]
```

example of request

```
http://localhost:8080/clover/simpleHttpApi/executions_history ->
```

```
         -> ?from=&to=2008-09-16+16%3A40&status=&sandbox=def&graphID=&index=&records=&returnType=DESCRIPTION
```

example of DESCRIPTION (plain text) response

```
execution|13108|FINISHED_OK|clover|def|test.grf|2008-09-16 11:11:19|2008-09-16 11:11:58|nodeA|2.4
phase|0|38733
node|DATA_GENERATOR1|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Output|0|0|0|0|0|130|10
node|TRASH0|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Input|0|0|0|5|0|130|10
node|SPEED_LIMITER0|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Input|0|0|0|0|0|130|10
port|Output|0|0|0|5|0|130|10
execution|13107|ABORTED|clover|def|test.grf|2008-09-16 11:11:19|2008-09-16 11:11:30
phase|0|11133
node|DATA_GENERATOR1|FINISHED_OK|0|0|0.0|0.0|0.0|0.0
port|Output|0|0|0|0|0|130|10
node|TRASH0|RUNNING|0|0|0.0|0.0|0.0|0.0
port|Input|0|5|0|5|0|52|4
node|SPEED_LIMITER0|RUNNING|0|0|0.0|0.0|0.0|0.0
port|Input|0|0|0|0|0|130|10
port|Output|0|5|0|5|0|52|4
```

For `returnType==DESCRIPTION_XML` returns a complex data structure describing one or more selected executions in XML format. A schema describing XML format of the XML response is accessible on **CloverDX Server** URL: http://[host]:[port]/clover/schemas/executions.xsd

Operation suspend

Suspends the Server or sandbox (if specified). No graphs may be executed on suspended Server/sandbox.

parameters

Table 36.6. Parameters of suspend

parameter name	mandatory	default	description
sandbox	no	-	The text ID of a sandbox to suspend. If not specified, it suspends the whole Server.
atonce	no		If this param is set to true, running graphs from suspended Server (or just from sandbox) are aborted. Otherwise it can run until it is finished in standard way.

returns

Result message

Operation resume

parameters

Table 36.7. Parameters of resume

parameter name	mandatory	default	description
sandbox	no	-	The text Id of a sandbox to resume. If not specified, the Server will be resumed.
verbose	no	MESSAGE	MESSAGE	FULL - how verbose should the possible error message be.

returns

Result message

Operation sandbox_create

This operation creates a specified sandbox. If it is a sandbox of "partitioned" or "local" type, it also creates locations by "sandbox_add_location" operation.

parameters

Table 36.8. Parameters of sandbox create

parameter name	mandatory	default	description
sandbox	yes	-	The text ID of a sandbox to be created.
path	no	-	A path to the sandbox root if the Server is running in a standalone mode.
type	no	shared	Sandbox type: shared | partitioned | local. For a standalone Server may be left empty, since the default "shared" is used.
createDirs	no	true	Switch whether to create a directory structure of the sandbox (only for a standalone Server or "shared" sandboxes in a cluster environment).
verbose	no	MESSAGE	MESSAGE | FULL - how verbose should possible error message be.

returns

Result message

Operation sandbox_add_location

This operation adds a location to the specified sandbox. Can be only used with partitioned or local sandboxes.

parameters

Table 36.9. Parameters of sandbox add location

parameter name	mandatory	default	description
sandbox	yes	-	A sandbox which we want to add a location to.
nodeId	yes	-	A location attribute - a node which has direct access to the location.
path	yes	-	A location attribute - a path to the location root on the specified node.
location	no	-	A location attribute - a location storage ID. If not specified, a new one will be generated.
verbose	no	MESSAGE	MESSAGE | FULL - how verbose should possible error message be.

returns

Result message

Operation sandbox_remove_location

This operation removes a location from the specified sandbox. Only sandboxes of the partitioned or local type can have locations asociated.

parameters

Table 36.10. Parameters of sandbox add location

parameter name	mandatory	default	description
sandbox	yes	-	Removes a specified location from its sandbox.
location	yes	-	A location storage ID. If the specified location isn't attached to the specified sandbox, the sandbox won't be changed.
verbose	no	MESSAGE	MESSAGE | FULL - how verbose should possible error message be.

returns

Result message

Operation download_sandbox_zip

This operation downloads the content of a specified sandbox as a ZIP archive.

parameters

Table 36.11. Parameters

parameter name	mandatory	default	description
sandbox	yes	-	A code of the sandbox to be downloaded.

returns

a content of a specified sandbox as a ZIP archive

example

```
wget --http-user=username --http-password=password http://localhost:8080/clover/simpleHttpApi/download_sandbox_zi
```

Operation upload_sandbox_zip

This operation uploads the content of a ZIP archive into a specified sandbox.

parameters

Table 36.12. Parameters

parameter name	mandatory	default	description
sandbox	yes	-	A code of the sandbox the ZIP file will be expanded to.
zipFile	yes	-	The ZIP archive file.
overwriteExisting	no	false	If `true`, the files already present in the sandbox will be overwritten.
deleteMissing	no	false	If `true`, the files not present in the ZIP file will be deleted from the sandbox.
fileNameEncoding	no	UTF-8	The encoding that was used to store file names in the ZIP archive.

returns

Result message

an example of request (with using curl CLI tool (http://curl.haxx.se/))

```
curl -u username:password -F "overwriteExisting=true"
    -F "zipFile=@/tmp/my-sandbox.zip"
    http://localhost:8080/clover/simpleHttpApi/upload_sandbox_zip
```

Operation cluster_status

This operation displays cluster's nodes list.

parameters

no

returns

Cluster's nodes list.

Operation export_server_config

This operation exports a current server configuration in XML format.

parameters

Table 36.13. Parameters of server configuration export

parameter name	mandatory	default	description
include	no	all	Selection of items that will be included in the exported XML file; the parameter may be specified multiple times. Possible values are: • `all` - include items of all types • `users` - include a list of users • `userGroups` - include a list of user groups • `sandboxes` - include a list of sandboxes • `jobConfigs` - include a list of job configuration parameters • `schedules` - include a list of schedules • `eventListeners` - include a list of event listeners • `launchServices` - include a list of launch services (deprecated) • `tempSpaces` - include a list of temporary spaces

returns

Current server configuration as an XML file.

example

```
wget --http-user=username --http-password=password http://localhost:8080/clover/simpleHttpApi/export_server_config
```

Operation import_server_config

This operation imports server configuration.

parameters

Table 36.14. Parameters of server configuration import

parameter name	mandatory	default	description
xmlFile	yes	-	An XML file with server's configuration.
dryRun	no	true	If `true`, a dry run is performed with no actual changes written.
verbose	no	MESSAGE	MESSAGE | FULL - how verbose should the response be: `MESSAGE` for a simple message, `FULL` for a full XML report.
newOnly	no	false	If `true` only new items will be imported to the Server; the items already present on the Server will be left untouched.
include	no	all	Selection of items that will be imported from the XML; the parameter may be specified multiple times. Possible values are: `all` - import items of all types `users` - import users `userGroups` - import user groups `sandboxes` - import sandboxes `jobConfigs` - import job configuration parameters `schedules` - import schedules `eventListeners` - import listeners `launchServices` - import launch services (deprecated) `tempSpaces` - import temporary spaces

returns

Result message or XML report

an example of request (with using curl CLI tool (http://curl.haxx.se/))

```
curl -u username:password -F "dryRun=true" -F "verbose=FULL"
    -F "xmlFile=@/tmp/clover_configuration_2013-07-10_14-03-23+0200.xml"
    http://localhost:8080/clover/simpleHttpApi/import_server_config
```

Chapter 37. JMX mBean

The **CloverDX Server** JMX mBean is an API that can be used for monitoring the internal status of the Server.

MBean is registered with the name:

```
com.cloveretl.server.api.jmx:name=cloverServerJmxMBean
```

.

JMX Configuration

Application's JMX MBeans aren't accessible outside of JVM by default. It needs some changes in an application server configuration to make JMX Beans accessible.

This section describes how to configure a JMX Connector for development and testing. Thus authentication may be disabled. For production deployment, authentication should be enabled. For more information, see for example Password Authentication

Configurations and possible problems:

- How to configure JMX on Apache Tomcat (p. 245)
- How to Configure JMX on WebSphere (p. 246)
- How to Configure JMX on Worker (p. 247)
- Possible Problems (p. 247)

How to configure JMX on Apache Tomcat

Tomcat's JVM must be executed with these parameters:

1. `-Dcom.sun.management.jmxremote=true`

2. `-Dcom.sun.management.jmxremote.port=8686`

3. `-Dcom.sun.management.jmxremote.ssl=false`

4. `-Dcom.sun.management.jmxremote.authenticate=false`

5. `-Djava.rmi.server.hostname=your.server.domain` (necessary only for remote JMX connections)

On UNIX like OS set environment variable CATALINA_OPTS i.e. like this:

```
export CATALINA_OPTS="-Dcom.sun.management.jmxremote=true
                      -Dcom.sun.management.jmxremote.port=8686
                      -Dcom.sun.management.jmxremote.ssl=false
                      -Dcom.sun.management.jmxremote.authenticate=false
                      -Djava.rmi.server.hostname=your.server.domain.com"
```

File `TOMCAT_HOME/bin/setenv.sh` (if it does not exist, you may create it) or `TOMCAT_HOME/bin/catalina.sh`

On Windows each parameter must be set separately:

```
set CATALINA_OPTS=-Dcom.sun.management.jmxremote=true
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.port=8686
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.authenticate=false
```

```
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.ssl=false
set CATALINA_OPTS=%CATALINA_OPTS% -Djava.rmi.server.hostname=your.server.domain
```

File `TOMCAT_HOME/bin/setenv.bat` (if it does not exist, you may create it) or `TOMCAT_HOME/bin/catalina.bat`

With these values, you can use the URL `service:jmx:rmi:///jndi/rmi://localhost:8686/jmxrmi` for connection to JMX server of JVM. No user/password is needed

How to Configure JMX on WebSphere

WebSphere does not require any special configuration, but the **CloverDX** MBean is registered with a name that depends on application server configuration:

```
com.cloveretl.server.api.jmx:cell=[cellName],name=cloverServerJmxMBean,node=[nodeName],
                  process=[instanceName]
```
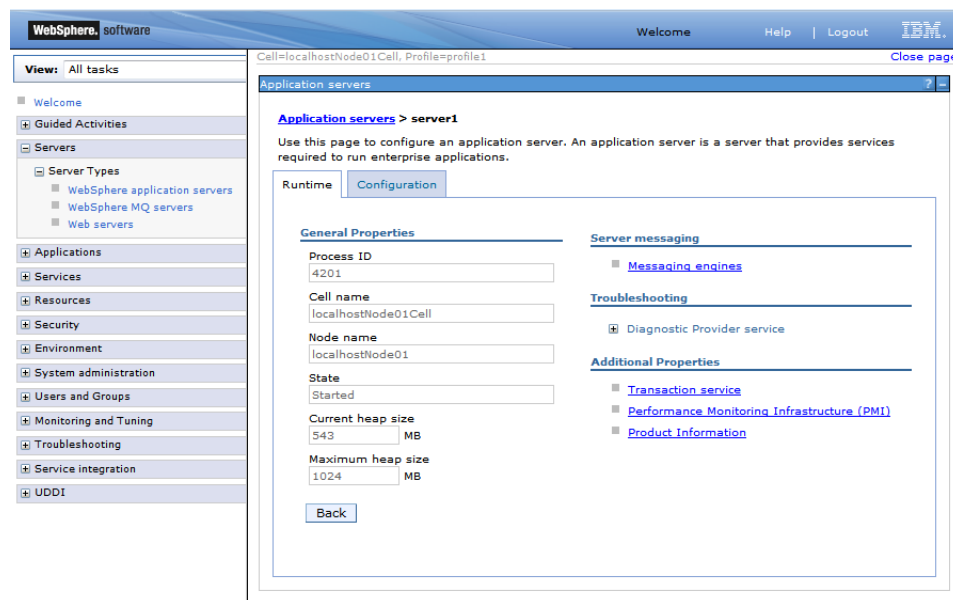

Figure 37.1. WebSphere configuration

The URL for connecting to JMX server is:

```
service:jmx:iiop://[host]:[port]/jndi/JMXConnector
```

where *host* is the host name you are connecting to and *port* is an RMI port number. If you have a default WebSphere installation, the JNDI port number will likely be 9100, depending on how many servers there are installed on one system and the specific one you want to connect to. To be sure, when starting WebSphere, check the logs for a line similar to this:

```
0000000a RMIConnectorC A   ADMC0026I: The RMI Connector is available at port 9100
```

You will also need to set on the classpath the following jar files from WebSphere home directory:

runtimes/com.ibm.ws.admin.client_8.5.0.jar
runtimes/com.ibm.ws.ejb.thinclient_8.5.0.jar
runtimes/com.ibm.ws.orb_8.5.0.jar

How to Configure JMX on Worker

See Additional Diagnostic Tools (p. 163) for details on how to enable JMX on Worker.

1. obtain `jmxremote_optional-repackaged-5.0.jar` - you can find it in the `cloverdx.server/web/WEB-INF/lib`

2. run VisualVM with `jmxremote_optional-repackaged-5.0.jar` on the classpath:

```
./visualvm --cp:a ../path/to/file/jmxremote_optional-repackaged-5.0.jar
```

3. use a URL with the JMXMP protocol, for example:
   `service:jmx:jmxmp://172.22.0.19:10501`

For more information, see JMX Monitoring and Management.

Possible Problems

• Default JMX mBean server uses RMI as a transport protocol. Sometimes RMI cannot connect remotely when one of peers uses Java version 1.6. As a solution, simply set these two system properties: `-Djava.rmi.server.hostname=[hostname or IP address] -Djava.net.preferIPv4Stack=true`

Operations

For details about operations, see the JavaDoc of the MBean interface:

JMX API MBean JavaDoc is accessible in the running **CloverDX Server** instance on URL: `http://[host]:[port]/[contextPath]/javadoc-jmx/index.html`

Chapter 38. SOAP WebService API

The **CloverDX Server** SOAP Web Service is an advanced API that provides an automation alternative to the Simple HTTP API. While most of the HTTP API operations are available in the SOAP interface too, the SOAP API provides additional operations for manipulating sandboxes, monitoring, etc.

The SOAP API service is accessible on URL:

```
http://[host]:[port]/clover/webservice
```

The SOAP API service descriptor is accessible on URL:

```
http://[host]:[port]/clover/webservice?wsdl
```

Protocol HTTP can be changed to secured HTTPS based on the web server configuration.

SOAP WS Client

Exposed service is implemented with the most common binding style "document/literal", which is widely supported by libraries in various programming languages.

To create client for this API, only WSDL document (see the URL above) is needed together with some development tools according to your programming language and development environments.

JavaDoc of the WebService interface with all related classes is accessible in a running **CloverDX Server** instance on URL `http://[host]:[port]/[contextPath]/javadoc-ws/index.html`

If the web server has an HTTPS connector configured, the client must also meet the security requirements according to web server configuration, i.e. client trust + key stores configured properly.

SOAP WS API Authentication/Authorization

Since exposed service is stateless, an authentication "sessionToken" has to be passed as a parameter to each operation. The client can obtain the authentication sessionToken by calling the `login` operation.

Chapter 39. Launch Services

> ### Important
>
> Since **4.9.0**, the **Launch Services** have been **deprecated**. We recommend to use Data Services (p. 256) instead.
>
> The Launch Services UI is hidden by default. To make it visible, use the launchservices.visible (p. 89) configuration property.

Launch Services allow you to publish a transformation graph or a jobflow as a Web Service. With Launch Services, **CloverDX** transformations can be exposed to provide a request-response based data interface (e.g. searches, complex lookups, etc.) for other application or directly to users.

Launch Services Overview

The architecture of a Launch Service is layered. It follows the basic design of multi-tiered applications utilizing a web browser.

Launch Services let you build a user-friendly form that the user fills in and sends to the **CloverDX Server** for processing.
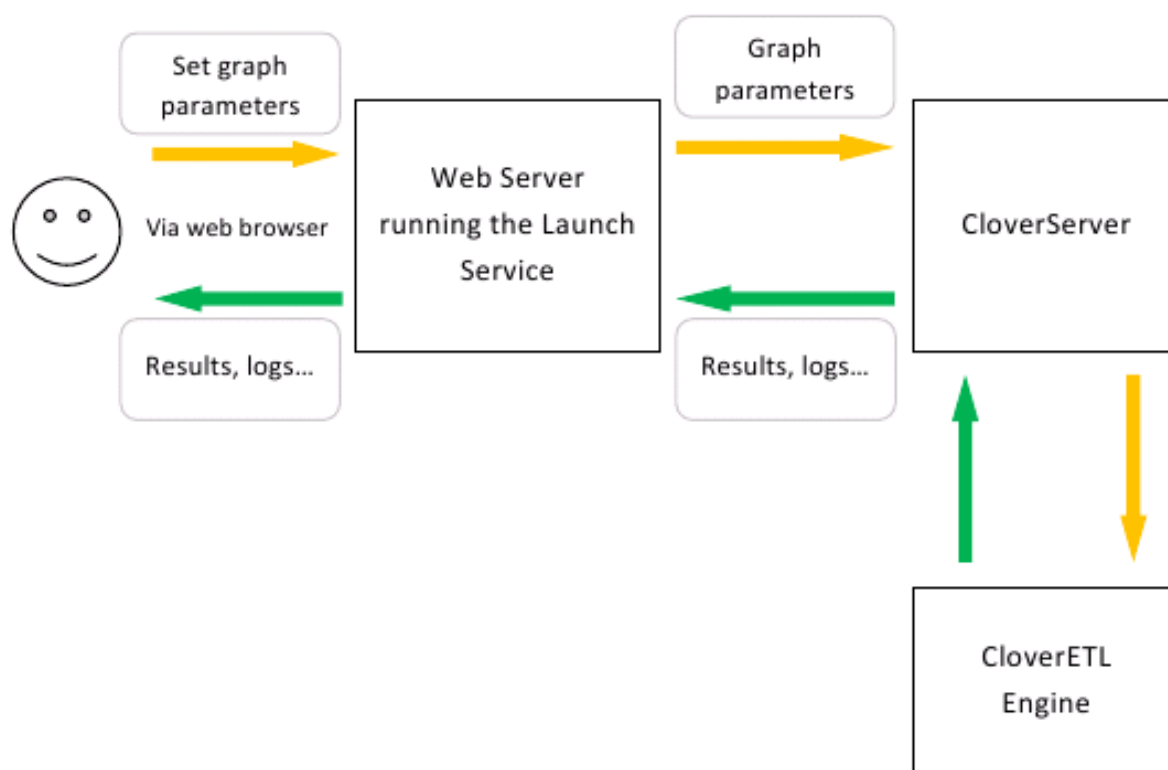
*Figure 39.1. Launch Services and **CloverDX Server** as web application back-end*

Deploying Graph in Launch Service

To prepare a graph for publishing as a Launch Service, keep this in mind during the design process:

1. You can define a graph/jobflow listeners to create parameterized calls. Parameters are passed to the graph as Dictionary entries – design the graph so that it uses the Dictionary as input/output for parameters (e.g. file names, search terms, etc.).

2. The graph will need to be published in the Launch Services section, where you provide the configuration and binding for parameters to dictionary entries.

Using Dictionary in Graph/Jobflow for Launch Services

A graph or a jobflow published as a service usually means that the caller sends request data (parameters or data) and the transformation processes it and returns back the results.

In a Launch Service definition, you can bind service's parameters to Dictionary entries. These need to be predefined in the transformation.

Dictionary is a key-value temporary data interface between the running transformation and the caller. Usually, although not restricted to, Dictionary is used to pass parameters in and out the executed transformation.

For more information about Dictionary, read the *Dictionary* section in the **CloverDX Designer** User's Guide.

Passing Files to Launch Services

If a Launch Service is designed to pass an input file to a graph or jobflow, the input dictionary entry has to be of type `readable.channel`.

Configuring the Job in CloverDX Server Web GUI

Each Launch Service configuration is identified by its name, user and group restriction. You can create several configurations with the same name, which is valid as long as they differ in their user or group restrictions.

User restrictions can then be used to launch different jobs for different users, even though they use the same launch configuration (i.e. name). For example, developers may want to use a debug version of the job, while the end customers will want to use a production job. The user restriction can also be used to prohibit certain users from executing the launch configuration completely.

Similarly, a group restriction can be used to differentiate jobs based on the user's group membership.
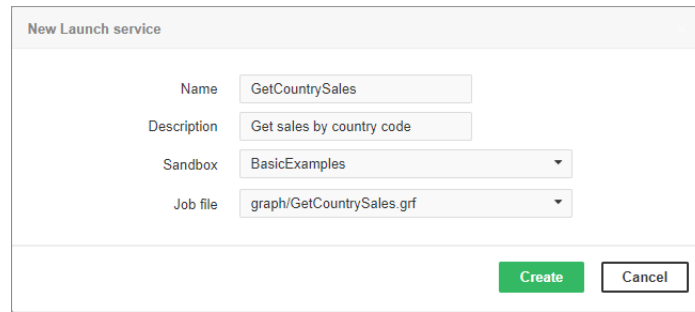
If multiple configurations match the current user/group and configuration name, the most specific one is picked. (The user name has a higher priority than the group name.)

Adding New Launch Configuration

Use the **New launch configuration** button on the Launch Services tab to create a new Launch Service.

The name is the identifier for the service and will be used in the service URL. Then, select a sandbox and either a transformation graph or a jobflow that you want to publish.
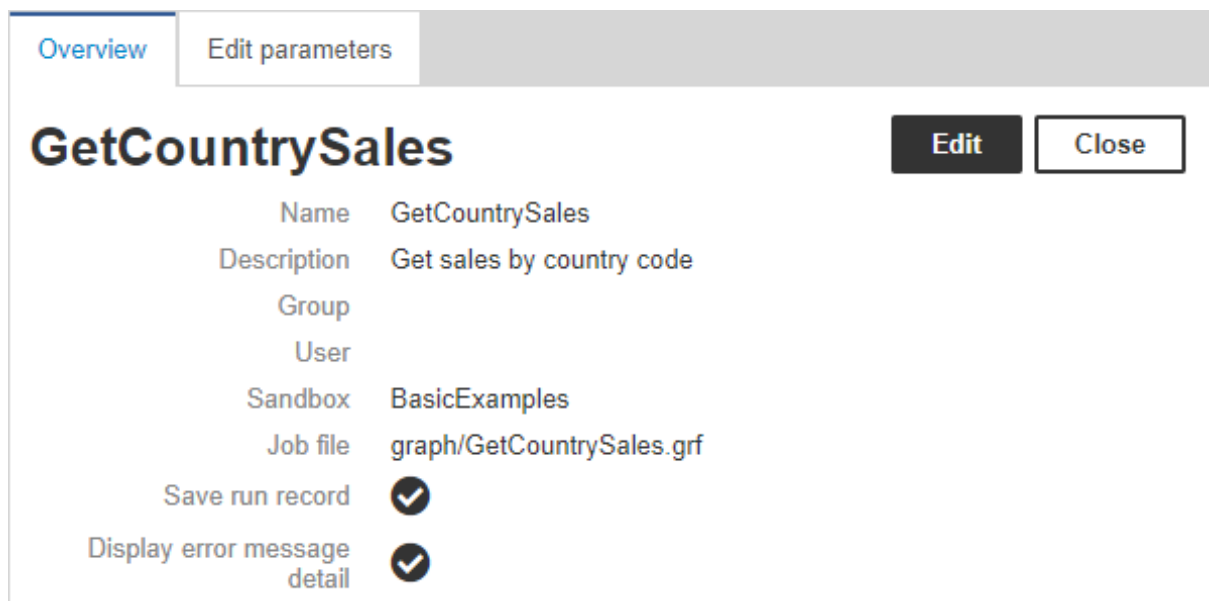
Figure 39.2. Creating a new launch configuration

Once you create the new Launch Service, you can set additional attributes like:

1. User and group access restrictions and additional configuration options (Edit Configuration)

2. Bind Launch Service parameters to Dictionary entries (Edit Parameters)

Figure 39.3. Overview tab

The Overview tab shows the basic details about the launch configuration. These can be modified in the Edit Configuration tab:

Edit Configuration

Figure 39.4. Edit Configuration tab

Editing configurations:

- *Name* - The name (identifier) under which the configuration will be accessible from the web.

- *Description* - The description of the configuration.

- *Group* - Restricts the configuration to a specific group of users.

- *User* - Restricts the configuration to a specified user.

- *Sandbox* - The **CloverDX** Sandbox where the configuration will be launched.

- *Job file* - Selects the job to run.

- *Save run record* - If checked, the details about the launch configuration will be stored in the Execution History. Uncheck this if you need to increase performance – storing a run record decreases response times for high frequency calls.

- *Display error message detail* - Check this if you want to get a verbose message in case the launch fails.

Edit Parameters

The **Edit parameters** tab can be used to configure parameter mappings for the launch configuration. The mappings are required for the Launch Service to be able to correctly assign parameters values based on the values sent in the launch request.

Figure 39.5. Creating new parameter

To add a new parameter binding, click on the **Add parameter** button. Every required graph/jobflow listener property defined by the job needs to be created here.

Figure 39.6. Edit Parameters tab

You can set the following fields for each property:

- *Dictionary entry name* - The name of the Dictionary entry defined in the graph/jobflow that you want to bind.

- *HTTP request parameter name* - The name of this property as it will be visible in the published service. This name can be different from **Name**.

- *HTTP request parameter required* - If checked, the parameter is mandatory and an error will be reported if it is omitted.

- *Pass HTTP request body* - If checked, the request body is set to dictionary entry as a readable channel.

- *Pass value as graph parameter* - If checked, the property value will be passed to the job also as a parameter (`${ParameterName}` where `ParameterName` is equal to `Name`). This lets you use the parameter anywhere in the job definition (not just places that support Dictionary). However, parameters are evaluated during job initialization. Thus, such a job cannot be pooled which decreases performance for high frequency repetitive calls to the service. In this case, consider redesigning the transformation to use Dictionary instead, allowing for pooling.

- *Default parameter value* - The default value applied in case the parameter is omitted in the launch request.

Launch Services Authentication

If you are using Launch Services, you have two ways how to be logged in: using form-based authentication of Server console or HTTP basic authentication of Launch Services.

The form-based authentication of Server console enables the user to create or modify Launch Services. If you are logged in this way, you act as an administrator of Launch Services.

To insert data into the Launch Service form, you should be logged in using the HTTP basic authentication. Follow the link to the Launch Service form and web browser will request your credentials. If you are logged in using HTTP basic authentication, you act as an user of Launch Services forms.

Sending the Data to Launch Service

A launch request can be sent via HTTP GET or POST methods. A launch request is simply a URL which contains the values of all parameters that should be passed to the job. The request URL is composed of several parts:

(You can use a Launch Services test page, accessible from the login screen, to test drive Launch Services.)

```
[Clover Context]/launch/[Configuration name]?[Parameters]
```

- `[Clover Context]` is the URL to the context in which the **CloverDX Server** is running. Usually this is the full URL to the **CloverDX Server** (for example, for **CloverDX Demo Server** this would be http://server-demo.cloverdx.com:8080/clover).

- `[Configuration name]` is the name of the launch configuration specified when the configuration was created. In our example, this would be set to "mountains" (case-sensitive).

- `[Parameters]` is the list of parameters the configuration requires as a query string. It is a URL-encoded [RFC 1738] list of name=value pairs separated by the "&" character.

Based on the above, the full URL of a launch request for our example with mountains may look like this: `http://example.com:8080/clover/launch/NewMountains?heightMin=4000`. In the request above, the value of `heightMin` property is set to 4000.

Results of the Graph Execution

After the job terminates, the results are sent back to the HTTP client as content of an HTTP response.

Output parameters are defined in the job's Dictionary. Every Dictionary entry marked as "Output" is sent back as a part of the response.

Depending on the number of output parameters, the following output is sent to the HTTP client:

- *No output parameters* - Only a summary page is returned. The page contains: when the job was started, when it finished, the user name, and so on. The format of the summary page cannot be customized.

- *One output parameter* - In this case, the output is sent to the client as in the body of the HTTP response with its MIME content type defined by the property type in Dictionary.

- *Multiple output parameters* - In this case, each output parameter is sent to the HTTP client as a part of the multipart HTTP response. The content type of the response is either multipart/related or multipart/x-mixed-replace, depending on the HTTP client (the client detection is fully automatic). The multipart/related type is used for browsers based on Microsoft Internet Explorer and the multipart/x-mixed-replace is sent to browsers based on Gecko or Webkit.

Launch requests are recorded in the log files in the directory specified by the `launch.log.dir` property in the **CloverDX Server** configuration. For each launch configuration, one log file named `[Configuration name]#[Launch ID].log` is created. For each launch request, this file will contain only one line with following tab-delimited fields:

(If the property `launch.log.dir` is not specified, log files are created in the temp directory `[java.io.tmpdir]/cloverlog/launch` where `java.io.tmpdir` is system property.)

- *Launch start time*

- *Launch end time*

- *Logged-in user name*

- *Run ID*

- *Execution status* FINISHED_OK, ERROR or ABORTED

- *IP Address* of the client

- *User agent* of the HTTP client

- *Query string* passed to the Launch Service (full list of parameters of the current launch)

If the configuration is not valid, the same launch details are saved into the `_no_launch_config.log` file in the same directory. All unauthenticated requests are saved to the same file as well.

Chapter 40. Data Services

Overview

Data Services allow you to deploy a web service. The architecture of the data service is described in the documentation on **Designer**. This section describes the server-side functionality of the Data Services.
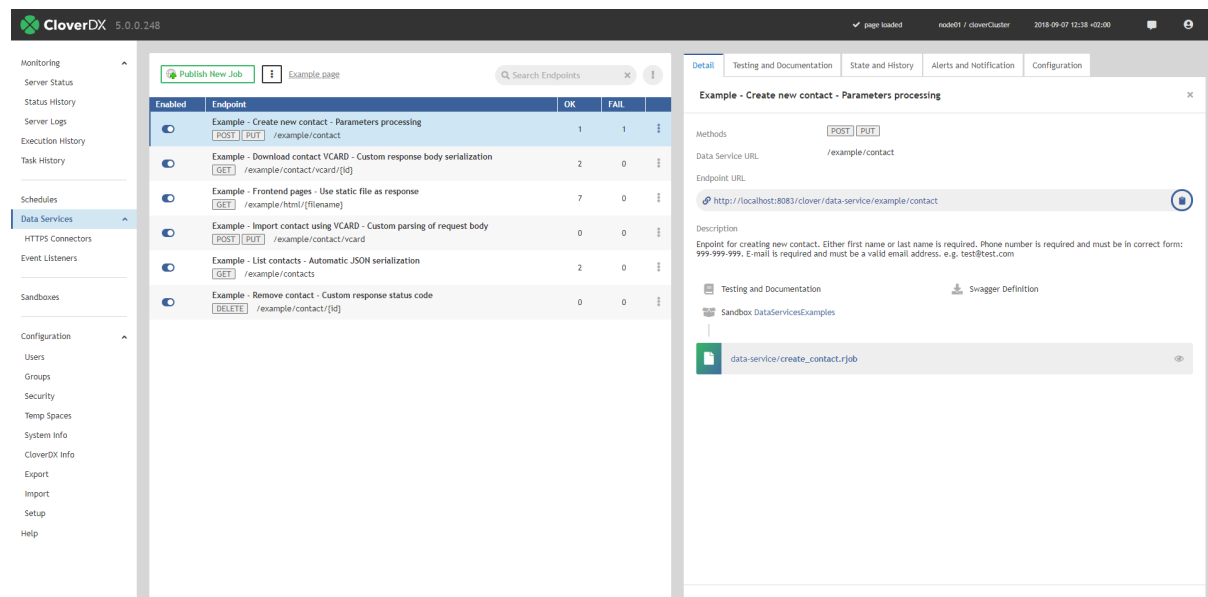
Figure 40.1. Data Services

The **Data Service** can be accessible via:

* **HTTP (default)**

    Requests are accepted via HTTP protocol on the same port as the Server. This is suitable for Data Services that do not require authentication.

* **HTTPS**

    If you need a secure connection, you should configure Data Service to listen on HTTPS: create an HTTPS Connector (p. 265) and use it in one or more Data Service endpoints. This way, you can configure a service listening on HTTPS port without restarting the Server. You can create multiple HTTPS Connectors and use it, for example, per consumer service. One Data Service endpoint can use only one HTTPS Connector.

You can publish the same data service job in multiple configurations. First, publish the job, then add an SSL Connector to it; after that, you can publish the job again with a different configuration. If a job is reconfigured to use an SSL Connector where it conflicts with another job, a failure occurs and the job has to be unpublished and republished.

Data Service can send you a notification (p. 260)in case of failure. You can set the threshold (p. 261)(number of subsequent failures or percentage) and way of notification (in the Server's UI or via email (p. 262)).
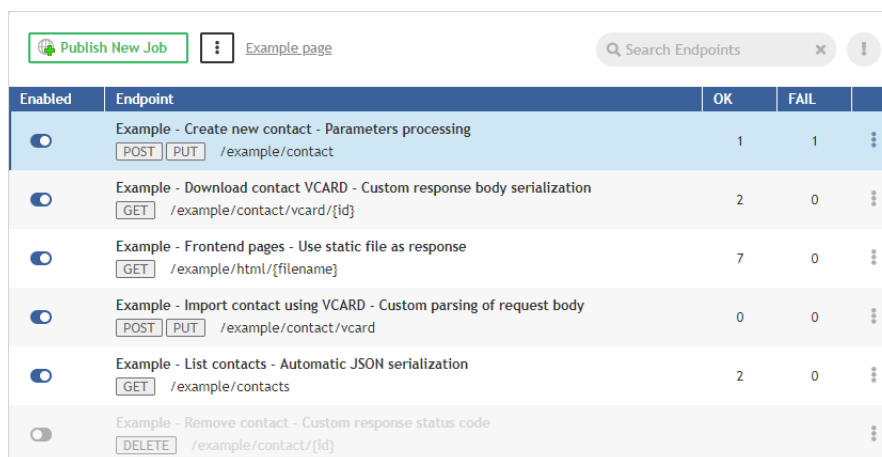
To investigate failed requests, you can use history of the particular endpoint(p. 260). Optionally, you can set the Data Service endpoint run to be recorded in Execution History (p. 263).

User Interface

Data Services user interface contains two main tabs: Endpoints (p. 257) and HTTPS Connectors (p. 265).

Endpoints

Endpoints tab consists of useful buttons (p. 257) in the top, list of data services (p. 257) and tabs with configuration of the particular data service.

Figure 40.2. Endpoints

Buttons

In the top of the **Endpoints** tab, there are five control elements.
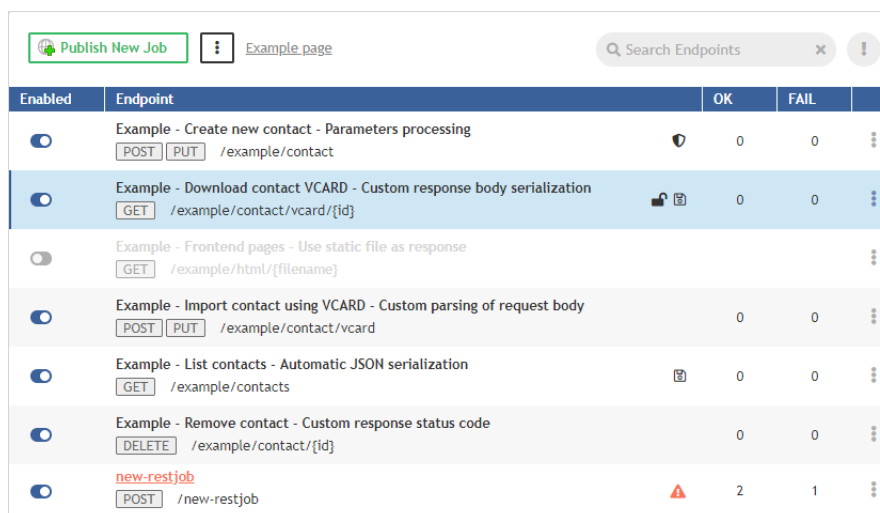
Figure 40.3. Data Service Endpoints

1. Creates a new Data Service job.

2. The three-dot-button has the following menu.

   Open Endpoint Documentation Catalog displays list of data services.

   Unpublish Data Services Examples.

   Import Data Services Configuration.

   Export Data Services Configuration.

3. Link to Data Services example.

4. Search function.

5. Filter to show all, or only failing and invalid Data Services.

List of Data Services

The **Data Services** tab contains list of all data services on the Server.

Figure 40.4. List of Data Services

- The button in the left column serves to enable (⬤) or disable (⬤) the service (e.g. temporary disable due to maintenance). A disabled Data Service returns the HTTP status code 503.

- In the second column, there are **Endpoint title**, **method(s)** and a part of **endpoint URL**.

  Icon decorators indicate these endpoint states:

  🔓 - The Data Service does not require authentication.

  📄 - The Data Service saves the job execution record in Execution History.

  ⚠ - The Data Service is marked as failing.

  🛡 - The Data Service is available on HTTPS.

- The third and fourth columns contain query statistics.

- The last column contains a menu with Data Service actions **Unpublish** and **Reset Endpoint State**.

Detail

The **Detail** tab contains overview of the particular endpoint. To display the **Detail** tab, click the particular line in the list of endpoints.
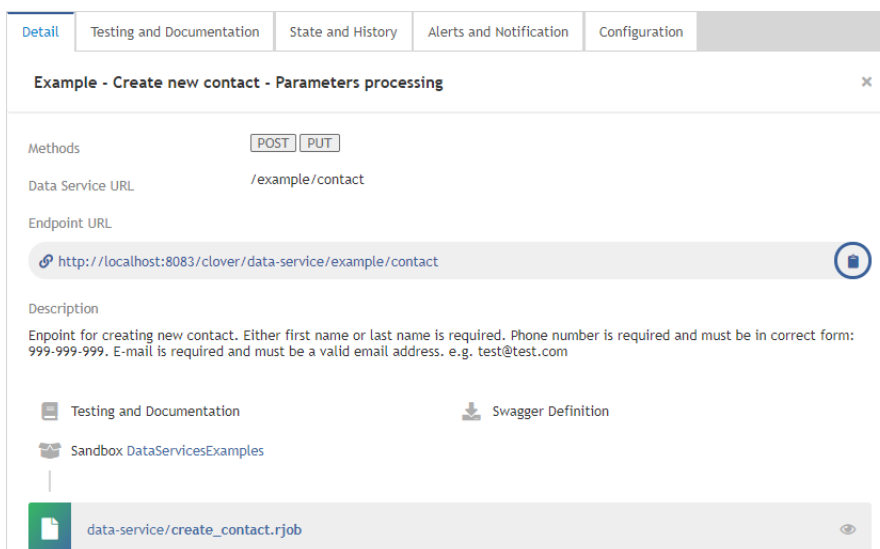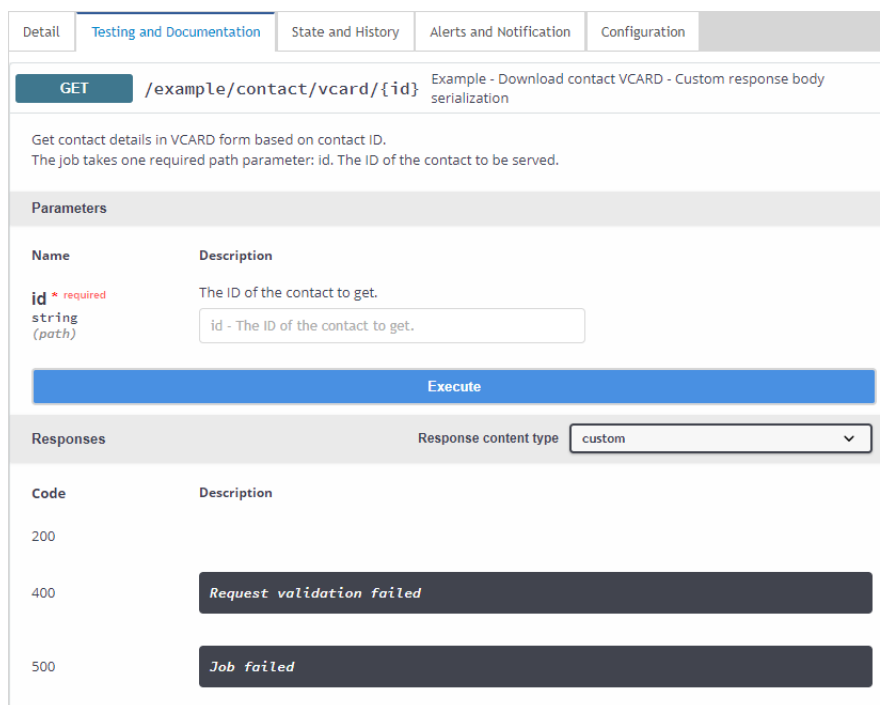
Figure 40.5. Data Service Detail tab

- **Endpoint title** is in the top of the tab's pane. It is the endpoint title specified in Designer on the **Endpoint Configuration** tab.

- **Methods** - indicates endpoint methods.

- **Data Service URL** - the configurable part of **Endpoint URL**. It can be set from Designer

- **Endpoint URL** - URL of the endpoint. This URL serves the requests.

  The **Copy link** (⚲) button copies the link to the clipboard.

- **Description** - contain a user-defined description of the Data Service. It can be set in Designer.

- **Testing and Documentation page** - links documentation of the endpoint. You can test the service there. This URL can be passed down to consumer of the service. The consumer can use the information from this URL to implement the client system.

- **Swagger file** - allows you to download a swagger file with the definition of the Data Service.

- **Sandbox** - sandbox containing the data service `.rjob` file.

- **REST job file** is a file name and path relative to the sandbox.

Testing and Documentation

The **Testing and Documentation** tab displays a user-defined documentation to the data service. The testing of the service is accessible under the **Execute** button.

Figure 40.6. Data Service - Testing and Documentation tab
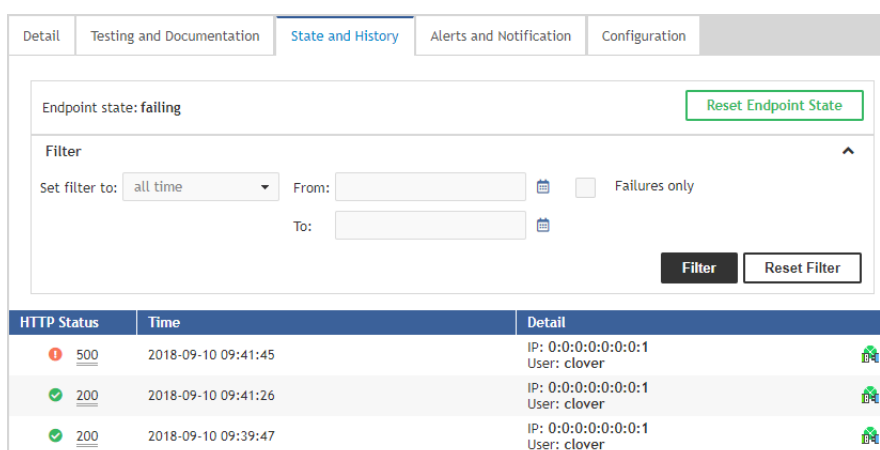
If you call the service from the Server UI, the Data Service will save its run record to the Execution History (p. 198).

State and History

The **State and History** shows invocation history of the particular Data Service. It contains a summary of the endpoint state in the top and a list of query details in the bottom. If the job is configured to save a record in execution history, the list also contains link to the **Execution History**.

You can filter records based on the time interval or you can list only the failures.

Here you can reset the state of the data service. For example, the data service endpoint was failing, you fixed it and you would like to be notified if it fails again.

Figure 40.7. Data Service - State and History

Alerts and Notification

No failure notification (p. 261)

The **Alerts and Notification** tab serves to set the threshold meaning the failure of the Data Service endpoint and way to notify you about it. You can set when the endpoint is marked as failing or disable this notification completely.

No failure notification

The **Never mark endpoint as failing** disables the failure notification in the Server UI. If you set this option and the request to the endpoint fails, there will be no red circle notification. Only the number of failures in activity column in the list of Data Service endpoints will be increased.

Failure

In this context, any response state from 4xx and 5xx range is considered as a failure.

Threshold specification

You can set threshold to

- any failure
- percentage of unsuccessful queries within interval
- fixed number of failures in a row

Any failure

The **Any failure will mark endpoint as failing** option considers the endpoint as failing even if a single failure occurs.

This choice is suitable for infrequently called Data Service endpoints.
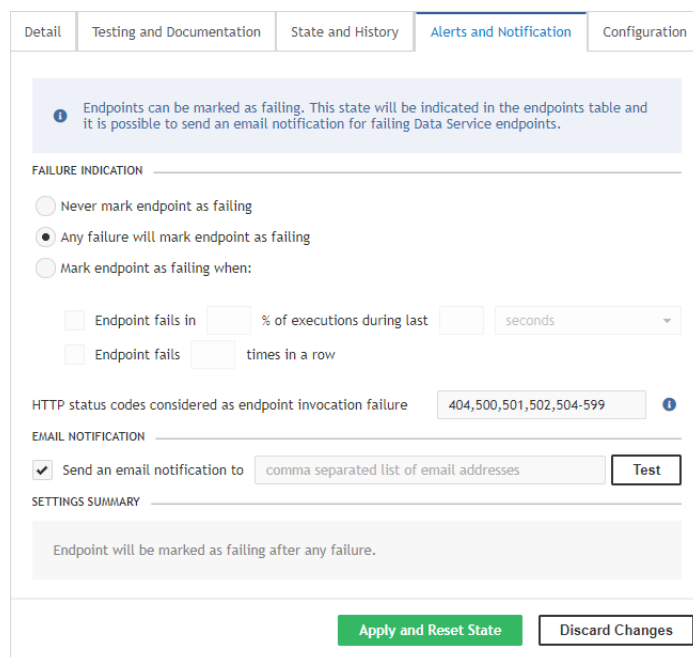
Figure 40.8. Data Service - Alerts and Notification

Threshold

The **Mark endpoint as failing when** option sets the endpoint as failing when a threshold is reached. The threshold can be specified as a *percentage of jobs is failing* or as a *number of jobs in row is failing*.

This choice is suitable for frequently called Data Service endpoints.

Figure 40.9. Data Service - Alerts and Notification

Failure Filtering

You can also select HTTP status codes which should be considered by **CloverDX Server** as an endpoint invocation failure.

Select the HTTP status codes by entering individual codes or ranges of codes separated by commas. By default, codes 404,500,501,502,504-599 are considered as an endpoint invocation failure. Leaving the field blank means that **CloverDX Server** considers all HTTP status codes from the range 400-599 as invocation failure.

> **Note**
>
> By default, **CloverDX Server** does **not** consider HTTP status code 503 as a failure, because the code is returned in the case of invocation of a manually disabled endpoint.

Failure Notification

If the Data Service endpoint fails, it is shown in the list of Data Services. Additionally, the number of failing Data Service endpoints is shown in the main menu.
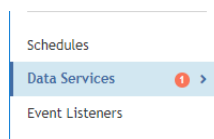
Figure 40.10. Some Data Service is failing

Email Notification

You can also set an email notification. This email notification works additionally to the notification in the Server UI. It sends an email when the endpoint's state changes to *failing*. An email is also sent if the endpoint was failing and you manually reset the endpoint state.

Figure 40.11. E-mail Notification

With the **Test** button, you can send a testing email to the addresses of the recipients.

Email notifications requires a working connection to an SMTP server.

Configuration

The **Configuration** tab allows you to disable the endpoint authentication or to enable saving records in Execution History (p. 198).

Figure 40.12. Data Service

The Data Service can be configured to require credentials or not. If the Data Service does not require credentials, the user to run it should be set in its configuration with the **Run As** option.

You can configure the data service to **save job execution record in Execution History**. The saving job execution record has a performance impact. Use this option only for:

• infrequently called endpoints
• endpoints that are not in production environment
• endpoints to be debugged

Catalog of Services

The *Catalog of services* is a list of data services allowing the user to view the documentation and test the service.

Figure 40.13. Global Catalog of Services

The details can be accessed by clicking the header. The first click displays the details, the second one fold the details back.

Figure 40.14. Global Catalog of Services

In the *Catalog of Services*, the end points can be grouped by sandbox or ordered by URL.

Built-in Data Service Examples

CloverDX Server contains built-in set of Data Service examples. The Data Service examples can be published from the **Data Services** tab.
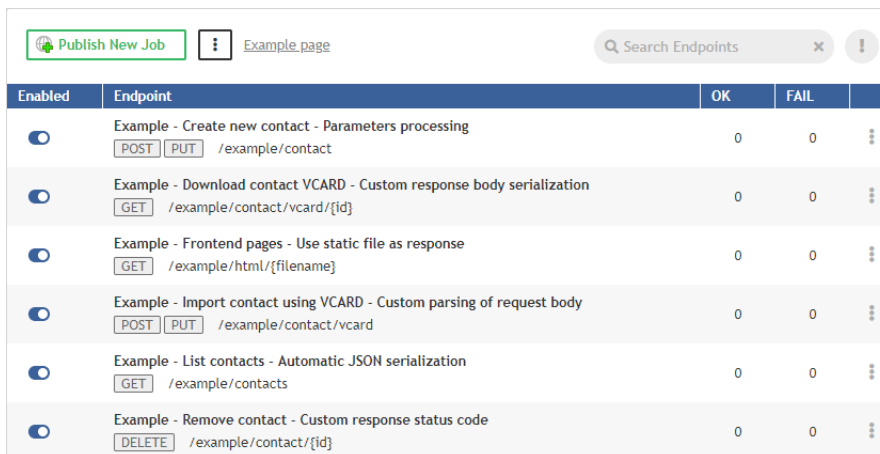
Figure 40.15. Data Services - Publishing the examples

The published examples are displayed among the others in the list of Data Services.

Figure 40.16. Data Services - Published the examples

HTTPS Connectors

The **Data Service** can be accessible via HTTPS. The configuration of HTTPS is in **Data Services →HTTPS Connectors**.

Figure 40.17. HTTPS Connectors

As a key store, we support the Java key store (`.jks`) and PKCS 12 key store (`.p12` or `.pfx`) formats.

As a trust store, we support the Java key store (`.jks`) format.

On the left hand side, there is a list of available HTTPS Connectors. On the right, there are details of the connector selected from the list. The **New HTTPS Connector** button creates a new HTTPS Connector.

List of HTTPS Connectors

The list of HTTPS Connectors shows available connectors. You can change the order by clicking on **Name** or **Enabled** in the header.

The button in the first column enables or disables the connector. Disabling the connector that is being used by Data Service makes the Data Service invalid.

The middle column shows the connector's name, port, path to the key store and path to the trust store.

The **...** button offers an option to delete the connector.

New HTTPS Connector

The **New HTTPS Connector** tab serves to create a new HTTPS Connector that can be used by one or more Data Services. One Data Service can use only one HTTPS Connector.

Figure 40.18. HTTPS Connectors

Table 40.1. Fields in Create HTTPS Connector dialog

Field	Description
Name	A name of the HTTPS Connector. The name should be unique. It is displayed in the list of HTTPS Connectors on the Endpoint's Configuration tab.
HTTPS Connector enabled	The checkbox enables or disables the HTTPS Connector to listen on the specified port. Stopping an HTTPS Connector that is being used by a Data Service makes the Data Service invalid.
Port	A TCP port used by the HTTPS Connector. The port must not be occupied by another HTTPS Connector or any other program. If the Data Service is Deployed on **CloverDX Cluster**, it listens on this port on all cluster nodes. If you use a firewall, set it to allow incoming connections to this port. If you use SELinux, it must be configured to allow **CloverDX Server** to use this TCP port.
Key store is located in sandbox	The checkbox switches between absolute paths to key store and paths relative to the Server sandbox. If selected, **Sandbox** and **Key store** items are displayed. Otherwise, you will see **Key store path**. The recommended way is to store the key stores out of the sandbox.
Sandbox	A sandbox with the key store.
Key store	The key store within the sandbox.
Key store path	An absolute path to the Java key store. You can use environment variables, system properties of JVM and configuration parameters of the Server as a part of the path. Usually, you will use `${sandboxes.home}` here.
Key store password	The password to the Java key store.
Key password	The password to the key in the key store.
Trust store is located in sandbox	The checkbox switches between absolute paths to trust store and paths relative to the Server sandbox. If selected, you can enter **Sandbox** and **Trust store** options. Otherwise, you will see **Trust store path**.
Sandbox	The sandbox containing the trust store.
Trust store	The trust store within the sandbox.
Trust store path	An absolute path to the trust store. You can use environment variables, system properties of JVM and configuration parameters of the Server as a part of the path. Usually, you will use `${sandboxes.home}` here.
Password	The password to the trust store.

To create a data service listening on HTTPS, you need a keystore with a server certificate. You can create one with the following command.

```
keytool -keystore service.jks -genkey -keyalg rsa -keysize 3072 -alias serverName
```

As a key store, we support the Java key store (`.jks`) and PKCS 12 key store (`.p12` or `.pfx`) formats.

As a trust store, we support the Java key store (`.jks`) format.

For security reasons, we recommend you to put the keystore outside the Server sandbox.

Using Data Services

Deploying Data Service

To deploy **Data Service** from the Server, go to **Data Services** tab, click **Publish Data Service job** and choose a sandbox and `.rjob` file.
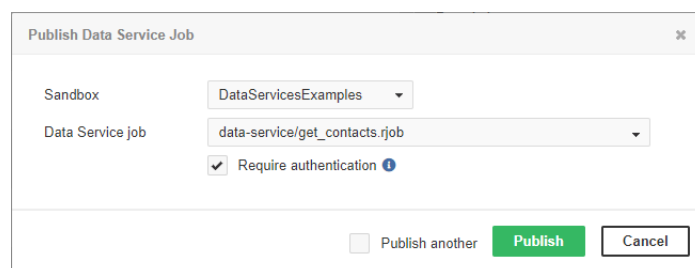
Figure 40.19. Publishing Data Service job

You can choose between Data Service with or without required authentication. In the latter case, the Data Service will run under the specified account.
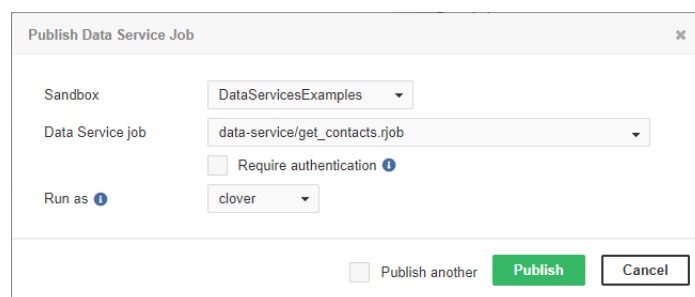
Figure 40.20. Publishing Data Service job that does not require authentication

Publishing Multiple Jobs

To deploy multiple jobs, tick the **Publish another** checkbox. After deploying one job, the dialog for publishing Data Service is displayed again to let you enter the next one.

Publishing and Unpublishing Data Service from Sandbox

You can deploy Data Service directly from a sandbox. To do so, you need read access to the sandbox and **List Data Services** and **Manage Data Services** privileges.

In the **Sandboxes** section of the Server GUI, select a data service to be published/unpublished. In the top right corner of the overview, there are options for publishing 🌐 or unpublishing 🌐 the data service, as well as downloading ⬇, downloading as ZIP 📦, showing data service in Execution History 🗒 and deleting 🗑 the data service.

Publishing Data Service Examples

CloverDX Server contains a built-in set of Data Service examples. These examples are not deployed by default.

The Data Service examples can be deployed directly from the **Data Services** tab. If you do not have any Data Service deployed, click the **Publish Data service Examples** link.
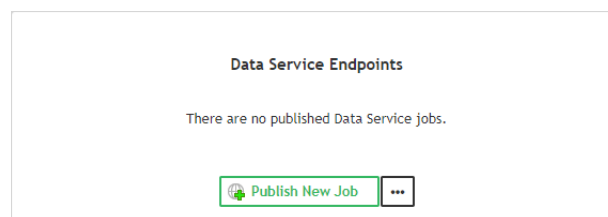
Figure 40.21. Publishing Data Service examples

If there is an existing Data Service, the button to publish examples is in the menu accessible under the three-dot-button.
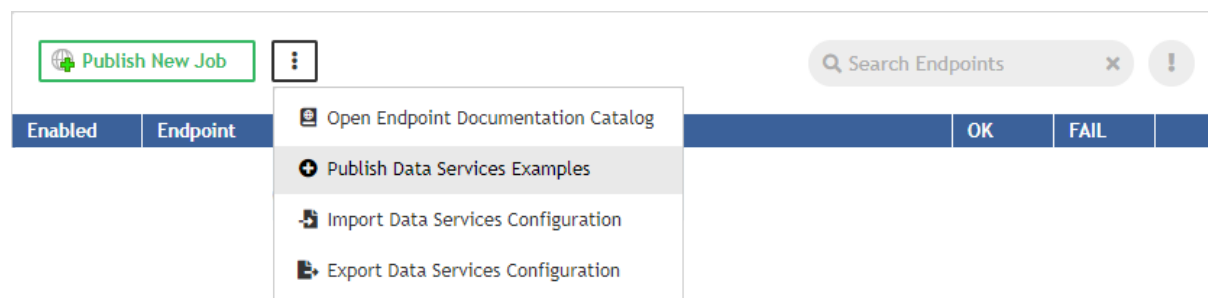
Figure 40.22. Publishing Data Service examples - II

See [Built-in Data Service Examples](#) (p. 264).

Changing Data Service to Anonymous

By default, the Data Service requires a client to send the credentials. To create the Data Service that does not require authentication, switch to the **Configuration** tab in the Data Service **Detail** pane and change **Authentication method** to *Do not require authentication*. The Data service runs with privileges of an existing user; therefore, you should set the *Run as* field to the suitable user. This user should have permissions necessary to run the Data Service.

Figure 40.23. Configuring Anonymous Data Service

In the list of Data Services, the Data Service that does not require credentials is indicated by unlocked padlock icon.
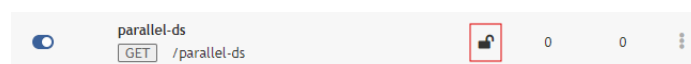
Figure 40.24. Data Service without authentication

Running Data Service on HTTPS

By default, the Data Service runs on HTTP and you can configure it to run on HTTPS.

To run Data Service on HTTPS, create a new **HTTPS Connector**.
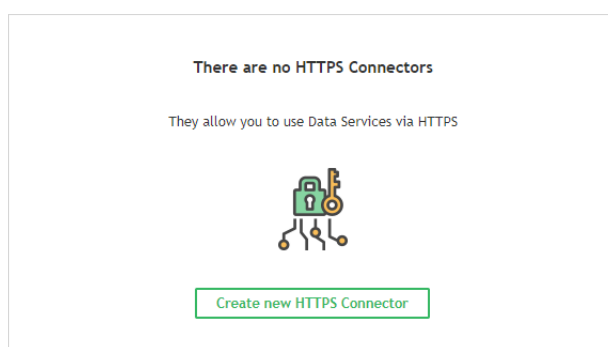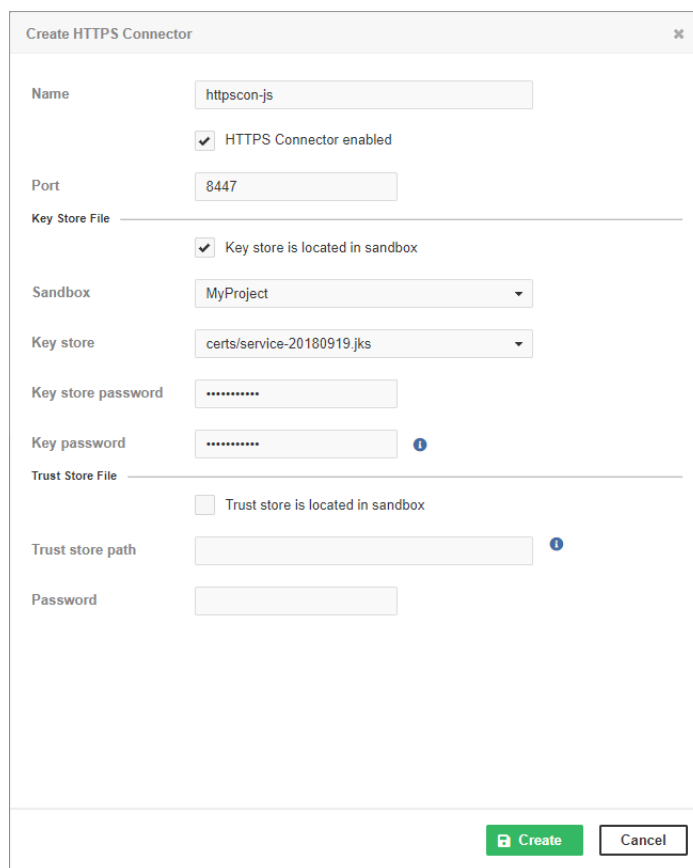
Figure 40.25. Creating a new Data Service Connector

Enter a name, port, keystore path, and keystore and key passwords.

Figure 40.26. Creating a new Data Service Connector II

In **Data Services** →**Endpoints**, select the Data Service to be running on HTTPS and switch to the **Configuration** tab.

Select the HTTPS connector from the combo box and click the **Apply** button. Now, the Data Service runs on HTTPS.

Figure 40.27. Using the HTTPS Connector in Data Service endpoint

You can have more independent HTTPS contexts running on one Server. There can be multiple Data Services running on the same HTTPS context.

Running Data Service on HTTPS on Cluster

This case extends the case of Running Data Service on HTTPS(p. 270). Different cluster nodes have different domain names, but the Java key store has to have one certificate. There are two way to solve the problem with certificates.

- Use a wildcard certificate. The key store file should be placed on the shared file system.

- Use different certificates for each cluster node. The keystores with the certificates must be on the same path on all cluster nodes.

Monitoring Data Service

To see the activity of Data Service, use the list of Data Services. There you can see the main overview of data services.

The state of a particular Data Service is on the *State and History* tab.

Testing Data Service

To test the Data Service, select the Data Service in the list, switch to the **Testing and Documentation** tab and click the **Execute** button. Results appears in the **Responses** part of the tab.

Performance Tuning

To improve performance, do not save job execution records in Execution History (p. 198). To do so, do not tick *Save job execution records in Execution History* on *Configuration* tab.

Exporting Data Service Configuration

You can export the Data Service configuration from **Data Services** →**Endpoints** tab. Click the three-dot-button and select **Export Data Services Configuration** from context menu.
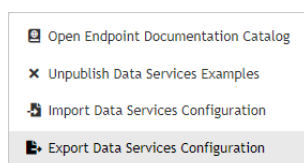
Figure 40.28. Data Services - Export

The Data Services configuration will be exported.

You can also export Data Service configuration in **Configuration** →**Export** See Server Configuration Export (p. 156).

Importing Data Service Configuration

You can import the Data Service configuration from **Endpoints**. Click the three-dot-button and select **Import Data Services Configuration** from the context menu.
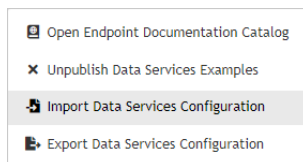
Figure 40.29. Data Services - Import

You can also import the Data Service configuration directly in **Configuration** →**Import** See Server Configuration Import (p. 157).

Avoiding Premature Marking of Data Service as Failing

Data Service might prematurely switch to a failing state if the failure indication is set up to switch to a failing state after a given percentage of executions fails in a given time window. E.g. First execution fails.

To avoid this, you can set the minimum number of events necessary to be taken into account when calculating the change of Data Service state. It can be set with the `dataservice.failure.ratio.min.record.count` configuration property . The default value is 10 executions.

It can be set in **Configuration** →**Setup** →**Configuration File**. Add a line containing

```
dataservice.failure.ratio.min.record.count=10
```

to the configuration file.

You can set it to any reasonable positive integer. This configuration is valid for all Data Services available on the Server.

See also Chapter 15, List of Configuration Properties (p. 82).

Looking up Particular Data Service

If you have multiple Data Services available, you can search for a specific Data Service:

If you know the endpoint name, you can look it up. Enter the text into the **Search endpoints** field and click the **Refresh** button. The Data Services will be filtered.

The entered text will be searched in the title of the Data Service, in the name of the request method, in the name of `.rjob` file and in the path that the Data Service uses.

If you would like to see invalid endpoints only, click the **failing and invalid only** icon. The both filters can be combined.

To switch off the filters, click the **Show All** button.

Resetting State of Failing Data Service Endpoint

If the Data Service endpoint is in the failing state and the problem has been fixed, you can reset the endpoint state manually.

To reset the state, open the details of the endpoint, switch to the **Alerts and Notification** tab and click the **Apply and Reset State** button.

If the endpoint has an email address set, a notification email will be sent to this address.

Custom HTTP Headers

Data Services accept a custom HTTP header `X-Clover-Save-Run-Record`. The possible values of the header are `TRUE` and `FALSE`. **CloverDX Server** accepts them case insensitively.

This header overrides the endpoint's configuration to save the run record or not.

Testing and Documentation page now automatically sends the header with value set to true. This means that all invocations from Testing and Documentation page are saved to the Execution History.

Testing Data Service from Designer creates a record in Execution History regardless of it being published or not.

Data Services on Cluster

Data Service jobs can run on Cluster in the same way as they run on **CloverDX Server**. Parallel run of one Data Service job on multiple cluster nodes in not supported.

Part VII. Cluster

Chapter 41. Sandboxes in Cluster

There are three sandbox types in total - shared sandboxes, and partitioned and local sandboxes (introduced in 3.0) which are vital for parallel data processing..

Shared Sandbox

This type of sandbox must be used for all data which is supposed to be accessible on all cluster nodes. This includes all graphs, jobflows, metadata, connections, classes and input/output data for graphs which should support HA. All shared sandboxes reside in the directory, which must be properly shared among all cluster nodes. You can use a suitable sharing/replicating tool according to the operating system and filesystem.

Figure 41.1. Dialog form for creating a new shared sandbox

As you can see in the screenshot above, you can specify the root path on the filesystem and you can use placeholders or absolute path. Placeholders available are environment variables, system properties or **CloverDX Server** configuration property intended for this use: sandboxes.home. Default path is set as [user.data.home]/CloverDX/sandboxes/[sandboxID] where the sandboxID is an ID specified by the user. The user.data.home placeholder refers to the home directory of the user running the JVM process (/home subdirectory on Unix-like OS); it is determined as the first writable directory selected from the following values:

- USERPROFILE environment variable on Windows OS

- user.home system property (user home directory)

- user.dir system property (JVM process working directory)

- java.io.tmpdir system property (JVM process temporary directory)

Note that the path must be valid on all cluster nodes. Not just nodes currently connected to the cluster, but also on nodes that may be connected later. Thus when the placeholders are resolved on a node, the path must exist on the node and it must be readable/writable for the JVM process.

Local Sandbox

This sandbox type is intended for data, which is accessible only by certain cluster nodes. It may include massive input/output files. The purpose being, that any cluster node may access content of this type of sandbox, but only one has local (fast) access and this node must be up and running to provide data. The graph may use resources from multiple sandboxes which are physically stored on different nodes since cluster nodes are able to create network streams transparently as if the resources were a local file. For details, see Using a Sandbox Resource as a Component Data Source (p. 278).

Do not use a local sandbox for common project data (graphs, metadata, connections, lookups, properties files, etc.). It would cause odd behavior. Use shared sandboxes instead.

Figure 41.2. Dialog form for creating a new local sandbox

The sandbox location path is pre-filled with the `sandboxes.home.local` placeholder which, by default, points to `[user.data.home]/CloverDX/sandboxes-local`. The placeholder can be configured as any other CloverDX configuration property.

Partitioned Sandbox

This type of sandbox is an abstract wrapper for physical locations existing typically on different cluster nodes. However, there may be multiple locations on the same node. A partitioned sandbox has two purposes related to parallel data processing:

1. **node allocation specification**

   Locations of a partitioned sandbox define the workers which will run the graph or its parts. Each physical location causes a single worker to run without the need to store any data on its location. In other words, it tells the **CloverDX Server**: to execute this part of the graph in parallel on these nodes.

2. **storage for part of the data**

   During parallel data processing, each physical location contains only part of the data. Typically, input data is split in more input files, so each file is put into a different location and each worker processes its own file.

Figure 41.3. Dialog form for creating a new partitioned sandbox

As you can see on the screenshot above, for a partitioned sandbox, you can specify one or more physical locations on different cluster nodes.

The sandbox location path is pre-filled with the `sandboxes.home.partitioned` placeholder which, by default, points to `[user.data.home]/CloverDX/sandboxes-paritioned`. The `sandboxes.home.partitioned` config property may be configured as any other **CloverDX Server** configuration property. Note that the directory must be readable/writable for the user running JVM process.

Do not use a partitioned sandbox for common project data (graphs, metadata, connections, lookups, properties files, etc.). It would cause odd behavior. Use shared sandboxes instead.

Using a Sandbox Resource as a Component Data Source

A sandbox resource, whether it is a shared, local or partitioned sandbox (or ordinary sandbox on standalone server), is specified in the graph under the **fileURL** attributes as a so called sandbox URL like this:

```
sandbox://data/path/to/file/file.dat
```

where `data` is a code for the sandbox and `path/to/file/file.dat` is the path to the resource from the sandbox root. The URL is evaluated by **CloverDX Server** during job execution and a component (reader or writer) obtains the opened stream from the Server. This may be a stream to a local file or to some other remote resource. Thus, a job does not have to run on the node which has local access to the resource. There may be more sandbox resources used in the job and each of them may be on a different node.

The sandbox URL has a specific use for parallel data processing. When the sandbox URL with the resource in a *partitioned sandbox* is used, that part of the graph/phase runs in parallel, according to the node allocation specified by the list of partitioned sandbox locations. Thus, each worker has its own local sandbox resource. **CloverDX Server** evaluates the sandbox URL on each worker and provides an open stream to a local resource to the component.

The sandbox URL may be used on the standalone Server as well. It is an excellent choice when graph references some resources from different sandboxes. It may be metadata, lookup definition or input/output data. A referenced sandbox must be accessible for the user who executes the graph.

Remote Edges

Data transfer between graphs running on different nodes is performed by a special type of edge - remote edge. The edge utilizes buffers for sending data in fixed-sized chunks. Each chunk has a unique number; therefore, in case of an I/O error, the last chunk sent can be re-requested.

You can set up values for various remote edge parameters via configuration properties. For list of properties, their meaning and default values, see Optional Remote Edge Properties (p. 283).

The following figure shows how nodes in a cluster communicate and transfer data - the client (graph running on Node 2) issues an HTTP request to Node 1 where a servlet accepts the request and checks the status of the source buffer. The source buffer is the buffer filled by the component writing to the left side of the remote edge. If the buffer is full, its content is transmitted to the Node 2, otherwise the servlet waits for configurable time interval for the buffer to become full. If the interval has elapsed without data being ready for download, the servlet finishes the request and Node 2 will re-issue the request at later time. Once the data chunk is downloaded, it is made available via the target buffer for the component reading from the right side of the remote edge. When the target buffer is emptied by the reading component, Node 2 issues new HTTP request to fetch the next data chunk.

This communication protocol and its implementation have consequences for the memory consumption of remote edges. A single remote edge will consume 3 x chunk size (1.5MB by default) of memory on the node that is the source side of the edge and 1 x chunk size (512KB by default) on the node that is the target of the edge. A smaller chunk size will save memory; however, more HTTP requests will be needed to transfer the data and the network latency will lower the throughput. Large data chunks will improve the edge throughput at the cost of higher memory consumption.
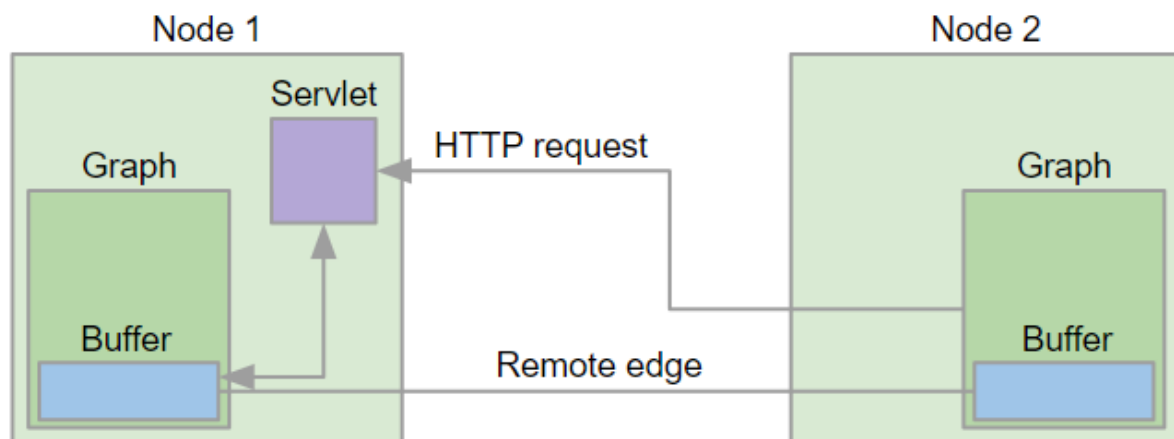
Figure 41.4. Remote Edge Implementation

Chapter 42. Cluster Configuration

Cluster can work properly only if each node is properly configured. Clustering must be enabled, nodeID must be unique on each node, all nodes must have access to shared DB (direct connection or proxied by another Cluster node) and shared sandboxes, and all properties for inter-node cooperation must be set according to network environment.

Properties and possible configuration are the following:

Mandatory Cluster Properties

Besides mandatory Cluster properties, you need to set other necessary properties which are not specifically related to the Cluster environment. Database connection must be also configured; however, besides direct connection, it is alternatively possible to configure proxying using another Cluster node/nodes. For details, see the property cluster.datasource.type (p. 283).

Mandatory properties

These properties must be properly set on each node of the Cluster.

Table 42.1. Mandatory Cluster properties

property	type	description	default
cluster.enabled	boolean	Switch whether the Server should start in the standalone or Cluster node mode. The property isn't set at all (empty value) by default, which means that the mode is chosen according to the loaded license. It is strongly recommended to set the property to `true` if the other Cluster properties are configured, as well. Thus the Cluster node will be initialized regardless of the license.	
cluster.node.id	String	Each Cluster node must have a unique ID.	node01
cluster.jgroups.bind_address	String, IP address	An IP address of the ethernet interface which is used for communication with another Cluster nodes. Necessary for inter-node messaging.	127.0.0.1
cluster.jgroups.start_port	int, port	Port where a jGroups server listens for inter-node messages.	7800
cluster.http.url	String, URL	A URL of the CloverDX Cluster node. It must be an HTTP/HTTPS URL to the root of a web application. Typically it would be `http://[hostname]:[port]/clover`. Primarily, it is used for synchronous inter-node communication from other Cluster nodes. It is recommended to use a fully qualified hostname or IP address, so it is accessible from a client browser or CloverDX Designer.	http://localhost:8080/clover
cluster.jgroups.tcpping.initial_hosts	String, in format: "IPaddress1[port1],IPaddress2[port2]..."	List of IP addresses (with ports) where we expect running and listening nodes. It is related to other nodes "bind_address and start_port" properties. Necessary for inter-node messaging.	127.0.0.1[7800]

Optional Cluster Properties

Optional General Properties

These properties are not vital for Cluster configuration - default values are sufficient.

Table 42.2. Optional general properties

property	type	description	default
cluster.jgroups.external_address	String, IP address	An IP address of the Cluster node. Configure this only if the Cluster nodes are on different sub-nets, so the IP address of the network interface isn't directly accessible from the other Cluster nodes.	
cluster.jgroups.external_port	int, port	A port for asynchronous messaging. Configure this only if the Cluster nodes are on different sub-nets and the port opened on the IP address is different than the port opened on the node's network interface IP address.	
cluster.jgroups.protocol.NAKACK.gc_lag	int	A number of delivered messages kept in the **sent messages buffer** of each jGroups view member. Messages are kept in a sender cache even though they were reported as delivered by existing view members, because there may be some other member temporarily not in the view. The higher the number, the higher the chance of reliable messages delivery in an unreliable network environment. However the messages consume memory: approximately 4kB for each message.	10000
cluster.jgroups.protocol.NAKACK.xmit_table_obsolete_member_timeout	long	How long (in milliseconds) we keep obsolete member in the xmit-table. It is necessary for recognition of member temporarily unaccessible and removed from the view. With previous NAKACK implementation, the member removed from the view was also automatically removed from xmit-table, so it appeared as a new member when it re-joined the view. With current modified implementation the member is kept in the xmit-table for a configured interval longer, so when it re-joins the view, it is a known member and undelivered messages may be re-delivered to it. A member in the xmit-table isn't consuming memory.	3600000
cluster.jgroups.protocol.AUTH.value	String	String used by a jgroups member to authenticate to the group. Must be the same on all Cluster nodes. It is a protection against fake messages.	
sandboxes.home.partitioned	String	Intended as a placeholder in the location path. So the sandbox path is specified with the placeholder and it is resolved to the real path just before it is used. For backward compatibility, the default value uses the [clover.home](#) (p. 82) configuration property.	${clover.home}/sandboxes-partitioned
sandboxes.home.local	String	Intended as a placeholder in the location path. So the sandbox path is specified with the placeholder and it is resolved to the real path just before it is used. For backward compatibility, the default value uses the [clover.home](#) (p. 82) configuration property.	${clover.home}/sandboxes-local
cluster.shared_sandboxes_path	String	**This property is deprecated.** This property still works but is used only when a shared sandbox doesn't have its own	

property	type	description	default
		path specified. It is just for backward compatibility and it is not recommended for new deployments. Since 3.5, we recommend to specify the sandbox path explicitly and use the `sandboxes.home` property/placeholder.	
cluster.node.sendinfo.interval	int	A time interval in milliseconds. Each node sends a heart-beat with information about itself to another nodes. This interval specifies how often the information is sent under common circumstances.	2000
cluster.node.sendinfo.cluster.node.sendinfo.min_interval	int	A time interval in milliseconds. A specified minimum interval between two heart-beats. A heart-beat may be send more often than specified by `cluster.node.sendinfo.interval`, e.g. when jobs start or finish. However the interval will never be shorter then this minimum.	500
cluster.node.sendinfo.history_interval	int	A time interval in milliseconds, for which each node stores a heart-beat in the memory. It is used for rendering figures in the web GUI-monitoring section.	240000
cluster.node.remove.interval	int	A time interval in milliseconds. If no node info comes in this interval, the node is considered as lost and it is removed from the Cluster.	50000
cluster.max_allowed_time_shift_between_nodes	int	The maximum allowed time shift between nodes. All nodes must have system time synchronized, otherwise the Cluster may not work properly. So if this threshold is exceeded, the node will be set as invalid.	2000
cluster.group.name	String	Each Cluster has its unique group name. If you need 2 Clusters in the same network environment, each of them would have its own group name.	cloverCluster
cluster.jgroups.protocol.AUTH.value	String	An authentication string/password used for verification Cluster nodes accessing the group. If this property is not specified, the Cluster should be protected by firewall settings.	
cluster.datasource.type	String	Change this property to `remote` if the node doesn't have a direct connection to the **CloverDX Server** database, so it has to use some other Cluster node as proxy to handle persistent operations. In such a case, the `cluster.datasource.delegate.nodeIds` property must be properly configured, as well. Properties `jdbc.*` will be ignored. Note that scheduler is active only on nodes with a direct connection.	local
cluster.datasource.delegate.nodeIds	String	A list of Cluster node IDs (separated by a comma) which this node may use as a proxy to handle persistent operations. At least one of the listed node IDs must be running, otherwise this node will fail. All listed node IDs must have a direct connection to **CloverDX Server** database properly configured. Property `cluster.datasource.delegate.nodeIds` is ignored by default. Property `cluster.datasource.type` must be set to remote to enable the feature.	

Optional Remote Edge Properties

Below is a list of names and default values of properties used to configure remote edges in a Clustered environment.

Table 42.3. Optional remote edge properties

property	description	default
cluster.edge.chunkSize	Specifies the size of a chunk created by the right side of a remote edge (in bytes).	524288
cluster.edge.chunkWaitTimeout	Specifies how long should the servlet wait for a next chunk to become available (in milliseconds).	60000
cluster.edge.connectTimeout	Specifies a socket connection timeout when fetching a chunk (in milliseconds).	30000
cluster.edge.readTimeout	Specifies a socket read timeout when fetching a chunk (in milliseconds).	90000
cluster.edge.handshakeTimeout	Specifies how long should the client wait until a remote edge is registered by a data producing job (in milliseconds).	120000
cluster.edge.chunkReadRetries	Specifies how many times should be a chunk fetch re-attempted before reporting an error to the consumer.	2
cluster.edge.disableChunkProtocol	Disables the chunked data transfer protocol, switching to the old implementation.	false
cluster.ssl.disableCertificateValidation	Disables validation of certificates in HTTPS connections of remote edges. Disabling the validation affects jobs run on both Worker and Server Core.	false

Example of 2 Node Cluster Configuration

This section contains examples of **CloverDX** Cluster nodes configuration. We assume that the user "clover" is running the JVM process and the license will be uploaded manually in the web GUI. In addition it is necessary to configure:

- sharing or replication of file system directory which the property "sandboxes.home" is pointing to. E.g. on Unix-like systems it would be typically `/home/[username]/CloverDX/sandboxes`.

- connection to the same database from both nodes

Basic 2-nodes Cluster Configuration

This example describes a simple Cluster: each node has a direct connection to a database.
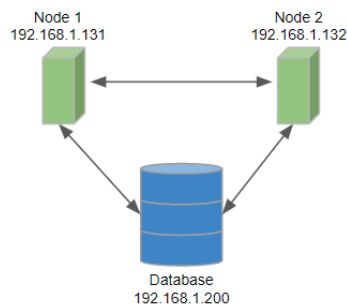
Figure 42.1. Configuration of 2-nodes Cluster, each node has access to a database

Configuration of Node 1 on 192.168.1.131

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://192.168.1.200/clover_db?charSet=UTF-8
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
jdbc.username=clover
jdbc.password=clover

cluster.enabled=true
cluster.node.id=node01
cluster.http.url=http://192.168.1.131:8080/clover
cluster.jgroups.bind_address=192.168.1.131
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.132[7800]

cluster.group.name=TheCloverCluster1

sandboxes.home=/home/clover/shared_sandboxes
```

Configuration of Node 2 on 192.168.1.132

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://192.168.1.200/clover_db?charSet=UTF-8
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
jdbc.username=clover
jdbc.password=clover
```

```
cluster.enabled=true
cluster.node.id=node02
cluster.http.url=http://192.168.1.132:8080/clover
cluster.jgroups.bind_address=192.168.1.132
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.131[7800]

cluster.group.name=TheCloverCluster1

sandboxes.home=/home/clover/shared_sandboxes
```

The configuration is done in a **properties file**. The file can be placed either on a default (p. 51) or specified (p. 50) location.

2-nodes Cluster with Proxied Access to Database

This Cluster configuration is similar to the previous one, but only one node has direct access to a database. The node2 has to use node1 as a proxy.
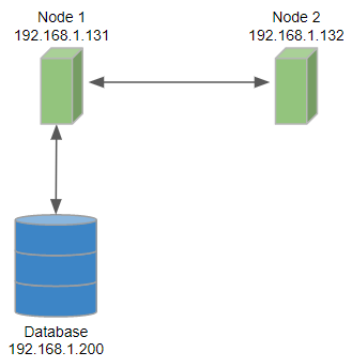
Figure 42.2. Configuration of 2-nodes Cluster, one node without direct access to database

Configuration of Node 1 on 192.168.1.131

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://192.168.1.200/clover_db?charSet=UTF-8
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
jdbc.username=clover
jdbc.password=clover

cluster.enabled=true
cluster.node.id=node01
cluster.http.url=http://192.168.1.131:8080/clover
cluster.jgroups.bind_address=192.168.1.131
cluster.jgroups.start_port=7800

cluster.group.name=TheCloverCluster2

sandboxes.home=/home/clover/shared_sandboxes
```

Configuration of Node 2 on 192.168.1.132

```
cluster.datasource.type=remote                              (1)
cluster.datasource.delegate.nodeIds=node01

cluster.enabled=true
cluster.node.id=node02
cluster.http.url=http://192.168.1.132:8080/clover
cluster.jgroups.bind_address=192.168.1.132
```

```
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.131[7800]

cluster.group.name=TheCloverCluster2

sandboxes.home=/home/clover/shared_sandboxes
```

1   These two lines describe access to database via another node.

2-nodes Cluster with Load Balancer

If you use any external load balancer, the configuration of CloverDX Cluster will be same as in the first example.
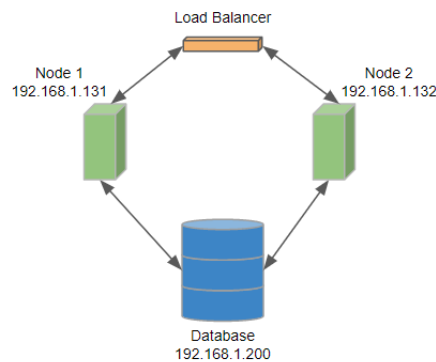
Figure 42.3. Configuration of 2-nodes Cluster with load balancer

The `cluster.http.url` and `cluster.jgroups.bind_address` are URLs of particular Cluster nodes even if you use a load balancer.

Configuration of Node 1 on 192.168.1.131

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://192.168.1.200/clover_db?charSet=UTF-8
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
jdbc.username=clover
jdbc.password=clover

cluster.enabled=true
cluster.node.id=node01
cluster.http.url=http://192.168.1.131:8080/clover
cluster.jgroups.bind_address=192.168.1.131
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.132[7800]

cluster.group.name=TheCloverCluster3

sandboxes.home=/home/clover/shared_sandboxes
```

Configuration of Node 2 on 192.168.1.132

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://192.168.1.200/clover_db?charSet=UTF-8
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
jdbc.username=clover
jdbc.password=clover

cluster.enabled=true
cluster.node.id=node02
cluster.http.url=http://192.168.1.132:8080/clover
cluster.jgroups.bind_address=192.168.1.132
```

```
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.131[7800]

cluster.group.name=TheCloverCluster3

sandboxes.home=/home/clover/shared_sandboxes
```

Example of 3 Node Cluster Configuration

Basic 3-nodes Cluster Configuration

This example describes a Cluster with three nodes where each node has a direct connection to a database.
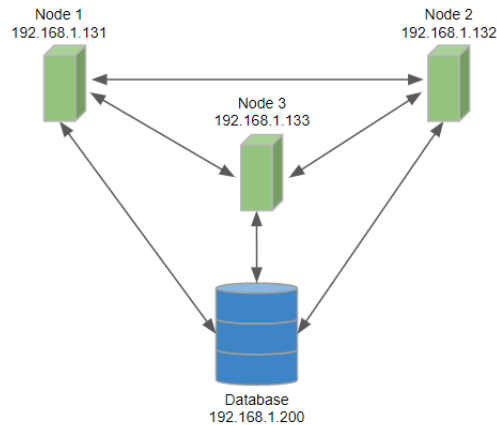
Figure 42.4. Configuration of 3-nodes cluster, each node has access to a database

Configuration of Node 1 on 192.168.1.131

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://192.168.1.200/clover_db?charSet=UTF-8
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
jdbc.username=clover
jdbc.password=clover

cluster.enabled=true
cluster.node.id=node01
cluster.http.url=http://192.168.1.131:8080/clover
cluster.jgroups.bind_address=192.168.1.131
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.132[7800],192.168.1.133[7800]

cluster.group.name=TheCloverCluster4

sandboxes.home=/home/clover/shared_sandboxes
```

Configuration of Node 2 on 192.168.1.132

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://192.168.1.200/clover_db?charSet=UTF-8
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
jdbc.username=clover
jdbc.password=clover

cluster.enabled=true
cluster.node.id=node02
cluster.http.url=http://192.168.1.132:8080/clover
cluster.jgroups.bind_address=192.168.1.132
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.131[7800],192.168.1.133[7800]

cluster.group.name=TheCloverCluster4

sandboxes.home=/home/clover/shared_sandboxes
```

Configuration of Node 3 on 192.168.1.133

```
jdbc.driverClassName=org.postgresql.Driver
jdbc.url=jdbc:postgresql://192.168.1.200/clover_db?charSet=UTF-8
jdbc.dialect=org.hibernate.dialect.PostgreSQLDialect
jdbc.username=clover
jdbc.password=clover

cluster.enabled=true
cluster.node.id=node03
cluster.http.url=http://192.168.1.133:8080/clover
cluster.jgroups.bind_address=192.168.1.133
cluster.jgroups.start_port=7800
cluster.jgroups.tcpping.initial_hosts=192.168.1.131[7800],192.168.1.132[7800]

cluster.group.name=TheCloverCluster4

sandboxes.home=/home/clover/shared_sandboxes
```

Jobs Load Balancing Properties

Multiplicators of load balancing criteria. A load balancer decides which Cluster node executes the graph. It means, that any node may process a request for execution, but a graph may be executed on the same or on different node according to current load of the nodes and according to these multiplicators.

The higher the number, the higher the relevance for decision. All multiplicators must be greater than 0.

Each node of the Cluster may have different load balancing properties. Any node may process incoming requests for transformation execution and each may apply criteria for loadbalancing in a different way according to its own configuration.

These properties aren't vital for Cluster configuration - default values are sufficient

Table 42.4. Load balancing properties

property	type	default	description
cluster.lb.balance.runningraphs	float	3	Specify importance of running graphs for load balancing.
cluster.lb.balance.memload	float	0.5	Specify importance of used memory for load balancing.
cluster.lb.balance.cpus	float	1.5	Specify importance of number of CPUs for load balancing.
cluster.lb.balance.request_bonus	float	2	Specify importance of the fact that the node is the same which processes the request for execution. The same node which decides where to execute the graph. If you specify this multiplicator great enough, the graph will be always executed on the same node which processes the request for execution.
cluster.lb.balance.node_bonus	float	1	Overall ratio bonus for configured node. Values greater then "1" increase probability of the node to be chosen by the loadbalancer. Value "1" means no bonus or penalty. "0" means that the node will never be chosen by the loadbalancer; however, it still may execute graphs, e.g. when there is no other node in the cluster or when the graph is designed to run on the node.

Running More Clusters

If you run more clusters, each Cluster has to have its own unique name. If the name is not unique, the Cluster nodes of different clusters may consider foreign Cluster nodes as part of the same cluster. The Cluster name is configured using `cluster.group.name` option. See .

Chapter 43. Recommendations for Cluster Deployment

1. All nodes in the cluster should have a synchronized system date-time.

2. All nodes share sandboxes stored on a shared or replicated filesystem. The filesystem shared among all nodes is a single point of failure. Thus, the use of a replicated filesystem is strongly recommended.

3. All nodes share a DB, thus it must support transactions. I.e. The MySQL table engine, MyISAM, may cause unusual behavior because it is not transactional.

4. All nodes share a DB, which is a single point of failure. Use of a clustered DB is strongly recommended.

5. Configure the license by `license.file` property or upload it in the Web GUI, so it is stored in the database. Do not use `clover-license.war`.

Chapter 44. Troubleshooting

Cluster Reliability in Unreliable Network Environment

CloverDX Server instances must cooperate with each other to form a Cluster together. If the connection between nodes doesn't work at all, or if it is not configured, Cluster can't work properly. This chapter describes Cluster nodes behavior in an environment where the connection between nodes is somehow unreliable.

Nodes use three channels to exchange status info or data

1. synchronous calls (via HTTP/HTTPS)

   Typically NodeA requests some operation on NodeB, e.g. job execution. HTTP/HTTPS is also used for streaming data between workers of parallel execution

2. asynchronous messaging (TCP connection on port 7800 by default)

   Typically heart-beat or events, e.g. job started or finished.

3. shared database – each node must be able to create DB connection

   Shared configuration data, execution history, etc.

Following scenarios are described below one by one, however they may occur together:

NodeA Cannot Establish HTTP Connection to NodeB

When HTTP request can't be established between nodes, jobs which are delegated between nodes or jobs running in parallel on more nodes will fail. The error is visible in the Execution History. Each node periodically executes a check-task which checks the HTTP connection to other nodes. If the problem is detected, one of the nodes is suspended, since they can't cooperate with each other.

Time-line describing the scenario:

- 0s network connection between NodeA and NodeB is down
- 0-40s a check-task running on NodeA can't establish HTTP connection to NodeB; check may last for 30s until it times-out; there is no re-try, if connection fails even just once, it is considered as unreliable, so the nodes can't cooperate.
- status of NodeA or NodeB (the one with shorter uptime) is changed to "suspended"

The following configuration properties set the time intervals mentioned above:

`cluster.node.check.checkMinPeriodicity` Periodicity of Cluster node checks, in milliseconds.

Default: 20000

`cluster.sync.connection.readTimeout` An HTTP connection response timeout, in milliseconds.

Default: 30000

`cluster.sync.connection.connectTimeout` Establishing HTTP connection timeout, in milliseconds.

Default: 7000

NodeA Cannot Establish TCP Connection (Port 7800 by Default) to NodeB

TCP connection is used for asynchronous messaging. When the NodeB can't send/receive asynchronous messages, the other nodes aren't notified about started/finished jobs, so a parent jobflow running on NodeA keeps waiting for the event from NodeB. A heart-beat is vital for meaningful load-balancing, the same check-task mentioned above also checks a heart-beat from all Cluster nodes.

Time-line describing the scenario:

- 0s network connection between NodeA and NodeB is down
- 60s NodeA uses the last available NodeB heart-beat
- 0-40s check-task running on NodeA detects missing heart-beat from NodeB
- status of NodeA or NodeB (the one with shorter uptime) is changed to `suspended`

The following configuration properties set the time intervals mentioned above:

`cluster.node.check.checkMinInterval` Periodicity of Cluster node checks, in milliseconds.

<div align="center">

Default: 40000

</div>

`cluster.node.sendinfo.interval` Periodicity of heart-beat messages, in milliseconds.

<div align="center">

Default: 2000

</div>

`cluster.node.sendinfo.min_interval` A heart-beat may occasionally be sent more often than specified by `cluster.node.sendinfo.interval`. This property specifies the minimum interval in milliseconds.

<div align="center">

Default: 500

</div>

`cluster.node.remove.interval` The maximum interval for missing a heart-beat, in milliseconds.

<div align="center">

Default: 50000

</div>

NodeB is Killed or It Cannot Connect to the Database

Access to a database is vital for running jobs, running scheduler and cooperation with other nodes. Touching a database is also used for detection of dead process. When the JVM process of NodeB is killed, it stops touching the database and the other nodes may detect it.

Time-line describing the scenario:

- 0s-30s last touch on DB
- NodeB or its connection to the database is down
- 90s NodeA sees the last touch
- 0-40s check-task running on NodeA detects obsolete touch from NodeB
- status of NodeB is changed to `stopped`, jobs running on the NodeB are `solved`, which means that their status is changed to `UNKNOWN` and the event is dispatched among the Cluster nodes. The job result is considered as `error`.

The following configuration properties set the time intervals mentioned above:

`cluster.node.touch.interval` Periodicity of a database touch, in milliseconds.

<div align="center">

Default: 20000

</div>

`cluster.node.touch.forced_interval` An interval when the other nodes accept the last touch, in milliseconds.

Default: 60000

`cluster.node.check.checkMinInterval` Periodicity of Cluster node checks, in milliseconds.

Default: 40000

`cluster.node.touch.forced_resume` A boolean value which can switch the resolving of running jobs mentioned above.

Auto-Resuming in Unreliable Network

In version 4.4, auto-resuming of suspended nodes was introduced.

Time-line describing the scenario:

- NodeB is suspended after connection loss
- **0s** NodeA successfully reestablishes connection to NodeB
- **120s** NodeA changes NodeB status to `forced_resume`
- NodeB attempts to resume itself if the maximum auto-resume count is not reached.
- If the connection is lost again, the cycle repeats; if the maximum auto-resume count is exceeded, the node will remain suspended until the counter is reset, to prevent suspend-resume cycles.
- **240m** auto-resume counter is reset

The following configuration properties set the time intervals mentioned above:

`cluster.node.check.interval.beforeForceResume` Time before node has to be accessible to be forcibly resumed, in milliseconds.

Default: 120000

`cluster.node.check.maxAutoResumeCount` How many times a node may try to auto-resume itself.

Default: 3

`cluster.node.check.interval.resetAutoResumeCount` Time before the autoresume counter will be reset, in minutes.

Default: 240

Long-Term Network Malfunction May Cause Jobs to Hang on

Jobflow or master execution executing child jobs on another Cluster nodes must be notified about status changes of their child jobs. When the asynchronous messaging doesn't work, events from the child jobs aren't delivered, so parent jobs keep running. When the network works again, the child job events may be re-transmitted, so hung parent job may be finished. However, the network malfunction may be so long, that the event can't be re-transmitted.

See following time-line to consider proper configuration:

- job A running on NodeA executes job B running on NodeB

- network between NodeA and NodeB is down from some reason

- job B finishes and sends the `finished` event, however it can't be delivered to NodeA – the event is stored in the `sent events buffer`

- Since the network is down, a heart-beat can't be delivered as well and maybe HTTP connections can't be established, the Cluster reacts as described in the sections above. Even though the nodes may be suspended, parent job A keeps waiting for the event from job B.

- **now, there are 3 possibilities:**

a. Network finally starts working and since all undelivered events are in the `sent events` buffer, they are re-transmitted and all of them are finally delivered. Parent job A is notified and proceeds. It may fail later, since some Cluster nodes may be suspended.

b. Network finally starts working, but the number of the events sent during the malfunction exceeded the `sent events` buffer limit size. So some messages are lost and won't be re-transmitted. Thus the buffer size limit should be higher in the environment with unreliable network. Default buffer size limit is 10,000 events. It should be sufficient for thousands of simple job executions; basically, it depends on number of job phases. Each job execution produces at least 3 events (job started, phase finished, job finished). Please note that there are also other events fired occasionally (configuration changes, suspending, resuming, cache invalidation). Also messaging layer itself stores own messages to the buffer, but the number is negligible (tens of messages per hour). Heart-beat is not stored in the buffer.

There is also an inbound events buffer used as a temporary storage for events, so events may be delivered in correct order when some events can't be delivered at the moment. When the Cluster node is inaccessible, the inbound buffer is released after timeout, which is set to 1 hour, by default.

c. Node B is restarted, so all undelivered events in the buffer are lost.

The following configuration properties set the time intervals mentioned above:

`cluster.jgroups.protocol.NAKACK.max_size` Limits the size of the sent events buffer; Note that each stored message takes 2kB of heap memory.

Default: 10000

`cluster.jgroups.protocol.NAKACK.inbound_buffer_timeout` An inbound buffer timeout of unaccessible Cluster node

Part VIII. Security

Chapter 45. Security Recommendations for CloverDX Server

To improve security of **CloverDX Server**, you should:

- Change the default password for **clover** user. Without changing the password, everybody would be able to log in as **clover**. See Change Users Password (p. 128).

- Create a user different from **clover** and add it to the **admin** group. If there are more administrators, create a user account for each. See Users (p. 128).

- Set the **master password**. Without the master password, you cannot use secure parameters. See Chapter 20, Secure Parameters (p. 119).

- Run **CloverDX Server** with privileges of an ordinary user, e.g. create a system account `clover` used only for running **CloverDX Server**. Do not run **CloverDX Server** under the **root** account.

- Communication with system database may be unencrypted. Consider encrypting the connection to system database too.

- If database provides you with a root/admin account, do not use this account for **CloverDX Server**. Create a separate database user account, e.g. **clover**.

- Run **CloverDX Server** on HTTPS. If you communicate over HTTP, your data is sent unencrypted and eavesdroppers can easily see it.

- Disable the HTTP API if you do not need it. See Chapter 36, Simple HTTP API (p. 234).

- In Data Services, put keystores outside a sandbox and run the service on HTTPS. If you have a keystore in a sandbox, a user with write permissions could replace it with another key store. HTTPS Connectors (p. 265).

- Enable **user lockout** after repeated failed login attempts. If you use this feature in Cluster, make sure that all cluster nodes have the same lockout configuration. See User Lockout (p. 142)

List of Figures

List of Tables