Labstory

Integrated ELN and LIMS

Enterprise Server Manual

Table of Contents

System requirements

- The software has been tested on OSX 10.6, Ubuntu Linux 12.04 and Windows XP
- Java 1.5+, for example, through OpenJDK
- Harddrive requirements depend heavily on user behavior. Images, sound and in particular video can drastically increase disk space usage. If this is a problem, it is possible to limit access to these facilities.
- Working knowledge of how to run software from the console and how to edit plain text files
- To customize the server beyond what is possible with the configuration file, the administrator must have working knowledge of Java programming and packaging.

Security

The entire software (client and server) is written according to the principle that “it's better with obviously no flaws rather than no obvious flaws”. It is our goal that you should dare to have unrestricted access from the internet, without any firewalls in between. The communication protocol is designed to avoid denial-of-service attacks; for example, it does not rely on Java RMI, class serialization, and it checks the size of incoming packets before accepting them. That said, the software has not yet been penetration tested, and we ask administrators to be vary of attempts. Regular backups with roll-back ability is strongly recommended. Further information about potential vulnerabilities and critical areas are available to users having support contracts.

Installation

Copying the files

The software is designed to have 3 different locations: The server binaries, the configuration files and the data files. These need not be different, but it is recommended to store the data files separately, as to ensure that intruders cannot overwrite the config files or binaries. These can later be specified with the -config and -data command-line flags.

Custom components

The server allows the end-user to write and load Java class-files for the purpose of customization. These should be packed as .jar-files and placed in XXXX/. The server will detect and load these when it is started. To reload these files, the server must be rebooted.

Connect to an SQL server

Once you have configured the server you should as soon as possible connect it to an SQL back-end (see later section). The default configuration is not suitable for production use.

Configuration

The common configuration file

The server is configured primarily through the file serversettings.xml. Below is a simple example:

```
<serverconfig>
	<port>28100</port>
	<name>test server</name>
	<accesspolicy class="AccessPolicyACL"/>
	<authentication class="AuthenticationServiceOr">
		<authentication class="AuthenticationServiceUserPasswordFile"/>
		<authentication class="AuthenticationServiceCrypto"/>
	</authentication>
	<logger class="LoggerTerminal"/>
</serverconfig>
```

port: One or more ports can be open. At least one port must be open for the server to run.

name: Name of the server (optional). This name is displayed to the client after connecting.

accesspolicy class="nameOfPolicy": Specifies the service determining access policy. See separate chapter for a list of policies.

authentication class="nameOfService": Specifies the service determining authentication policy. See separate chapter for a list of methods.

logger class="nameOfService": Specifies a service handling logging of events. Optional, multiple possible (one XML element for each). See separate chapter for a list of available loggers.

Authentication policies

Authentication policies determine how the login process should work. The server-client handshaking protocol however limits the policies to either confront the user with a user/password, or challenging a user with a crypto key.

A user will only be authenticated by one policy in the end, with the policies tested in the order of definition.

Pre-defined policy: AuthenticationServiceUserPasswordFile

This policy confronts the user with a username & password. These entries are stored in an XML-file called passwords.xml. A minimal example looks like this:

```
<passwords>
	<login user="johan" password="123"/>
	<login user="guest" password=""/>
</passwords>
```

This maps for example “guest” to the role “u:guest”, requiring no password.

To be able to log in as an administrator, you also have to give administrator access to one role. These are stored in a separate permission database. You can insert permissions like this:

```
java -jar server.jar -addrolebit "u:root" a
```

This adds the “a” (administrator) permission bit to the user “root”. See the -help flag for a full list of commands.

TODO hashed passwords

Pre-defined policy: AuthenticationServiceLDAP

This policy forwards authentication to an LDAP server. The settings are taken from the separate ldapauth.xml configuration file. Below is an example:

```
<ldap>
	<url>ldap://localhost:636/o=JNDITutorial</url>
	<kerberosconfig>/etc/krb5.conf</kerberosconfig>
	<userpath>cn=%, ou=NewHires, o=JNDITutorial</userpath>
</ldap>
```

url: If you want to use Kerberos for authentication, set it to “kerberos”. Then you need also set kerberosconfig to further configure this option. If instead you want to use SSL set the URL to e.g.

ldap://localhost:636/o=JNDITutorial , or if you prefer LDAPS, ldaps://localhost:636/o=JNDITutorial.

kerberosconfig: Optional. Set to the path of the kerberos configuration file.

userpath: The user name from the client is mapped to a string on the server using this string. If the path is “cn=%, cn=Users,dc=tilion,dc=org,dc=uk”, the user name provided will replace the %.

Implementation note 1: Plain-text connections are not supported. Unless you use Kerberos, DIGEST-MD5 will be used to cover the password. Thus LDAPv3 is required.

Implementation note 2: A caveat with all LDAP-based solutions is that the password will have to be transferred to the Labstory application server without md5-digest. Some users might be uncomfortable with this (for good reason). On request, it might be possible to enable Kerberos authentication at the client, but this will instead require that the client is on the same network and can reach the server.

Pre-defined policy: AuthenticationServiceCrypto

This policy allows users to sign in using the private crypto key. The server allows any user with a registered public key to enter.

To enable users to access the server, these users should communicate their public key to the server administrator, in some secure manner (e.g. USB-stick, or sent in a signed E-mail). The administrator can then upload the keys to the server. Optionally, limited password access can be used for uploading.

Pre-defined policy: AuthenticationServiceOr

TODO: maybe not needed – always use OR instead?

Custom authentication policies

New access policies should implement the interface AuthenticationService.

Access policies

Access policies defines what an authenticated user is able to do.

Pre-defined policy: AccessPolicyFull

This policy gives full access to any user who can log in. Primarily useful for small groups, and for setting up new servers.

Pre-defined policy: AccessPolicyACL

This is the policy we recommend, and it gives access based on Access Control Lists. These allow fine-grained access policies, dictated by those who create documents. This is the permission model described in the rest of this manual.

Custom policies

New access policies should implement the interface AccessPolicy or extend a previous access policy.

Logging facilities

A log facility receives log output from the server and takes action based on it. Any number of facilities may be installed concurrently.

Pre-defined facility: LoggerTerminal

The terminal logger is the most basic facility and mainly useful during testing the server. It simply writes all messages to the terminal. There are no options.

Pre-defined facility: LoggerLog4j

This policy covers the needs of most users by forwarding all messages to the Log4j framework. If no options are given, the server will look for log4j.properties in the configuration directory. Otherwise you may specify a file using e.g.

```
<configfile>/etc/log4j.properties</configfile>
```

The Log4j framework is extremely general and the log can be sent to for example:

- Files
- Remote sockets
- The NT Event Log
- Local and remote Unix Syslog daemons

How to set up Log4j itself is out of scope of this manual. See instead

http://logging.apache.org/log4j/1.2/

Custom logging facility

It is possible to create a custom class implementing the interface LabnoteLogger. This makes it possible to for example do much more elaborate logging as the default formatter does not output all available data. If the goal is only to send the text to another location then consider instead extending LoggerFormattedText.

Storing data in an SQL database

Introduction

Labstory can store data in multiple ways. The default configuration is chosen to make it very easy to install, requiring no SQL database (actually, for some parts it uses a small built-in database). This cannot be changed for the free client, but for the server there are options. Labstory can use a separate existing SQL database resulting in several benefits:

- Performance is increased for the modules otherwise relying on simpler file formats
- The software will handle power failures better (assuming proper SQL configuration). With the default configuration, if a file is being written as the hardware fails, the software could be left in an inconsistent state. SQL databases solves this by implementing check-pointing and transactions.
- The LIMS back-end is exposed. This allows cross-referencing to external SQL tables used by other software (such as other LIMS software).

It is expected that the server (not client) is installed with an industrial strength SQL database. Ecobima AB is not liable for data losses caused by hardware failures.

Recommended SQL back-ends

Labstory uses the SQL92 standard as far as possible, meaning that it should be compatible with any SQL database implementing this standard. Most do, but a few exceptions exist. We test the software with the open source database PostgreSQL, which we also recommend. Other databases can be supported on a per-contract-basis.

Basic configuration

Here you are assumed to have an SQL server up and running. You should *not* have stored data in the server before (or it will have to be migrated afterwards). You should have created a new database (or chosen an existing one), and have verified that you can log into it with user/password. Finally you need to have a JDBC-driver for it (a .jar-file).

1. Copy the .jar-file (or symlink it) to installationDirectory/jdbc.jar
2. Edit serverconfig.xml. Add the following record:

```
<serverconfig>
….
      <sql class="org.postgres.." name="..." user="..." pass="...">
      </sql>
</serverconfig>
```

This is everything required to get going with an SQL backend!

Details for a PostgreSQL configuration

Install a PostgreSQL server. If you are using Linux then this can be done very easily from the package repository, otherwise see http://www.postgresql.org/. We also recommend that you download http://www.pgadmin.org/ which is a very convenient graphical user interface. See their documentation how to tune the server, and in particular, to ensure that the files are stored on safe storage.

Create a database called “labstory” and assign full permissions to the user “labstory”. This can be done either using pgAdmin or the command-line tool psql.

Download a suitable JDBC driver from http://jdbc.postgresql.org/. The driver has to match the version of your SQL installation. Copy the jar-file to installationDirectory/jdbc.jar.

Add the following to the server configuration file:

```
<sql class=”org.postgresql.Driver” name=”jdbc:postgresql://localhost/labstory”
user=”labstory” pass=””/>
```

This assumes that the Labstory application server is running on the same computer as PostgreSQL (localhost); if you wish to have PostgreSQL running on another computer, you need to edit the PostgreSQL configuration file to allow remote connections.

Mapping SQL tables to Labstory LIMS tables

It is possible to make an existing SQL table visible as a read-only Labstory table (see the general LIMS chapter). To the <sql> tag, you may add the following:

```
<mapToLims from=”name of sql table” to=”ID in labstory”/>
```

To map the opposite direction, add:

```
<mapFromLims from=”ID in labstory” to=”name of sql table”/>
```

There is no limit to how many tables you can map. However, you may not put foreign keys against the exported tables as these are rewritten from time to time. Doing so will break the table reindexing.

Migrating data from one storage to another

Labstory itself does not contain any tools for copying data from one source to another. However, you may attempt to dump the database as text, and then import this to your other database. If you have previously used the built-in database then there are command-line and graphical tools available at http://www.hsqldb.org/.