

ezDL – Operator Guide

Working Group Information Engineering, University of Duisburg-Essen

Contents

1	Installation and setup	5
1.1	Database	5
1.1.1	MySQL	5
2	Agents	9
2.1	Configuration	9
2.2	Starting agents	10
3	Wrapper agents	11
3.1	Types of wrapper agents	11
3.2	Toolkit wrappers and web proxies	12
3.2.1	The wrapper	12
3.2.2	Squid	12
4	Deployment	15
4.1	Signing GFrameDL application	15
4.1.1	Key stores	15
4.1.2	Creating a key store	15
4.1.3	Converting a PCKS12 key store to a default Java key store	15
4.1.4	Signing JARs	15

Chapter 1

Installation and setup

In this chapter we will assume that you work under a Unix-related OS and have already created a system user `ezdl` with group `ezdl` and a home directory `/home/ezdl` that will contain all ezDL related data such as compiled classes, scripts and so on.

The chapter will first introduce the database setup necessary to start the ezDL backend. The following chapters then outline the configuration and start of the agents and wrapper agents.

1.1 Database

This section will walk you through the process for the simplest setup that uses one single database on one single system for all agents. This setup is later assumed in the chapter about the configuration and start of the agents. It is strongly recommended to first setup the system in the way outlined in this document and then moving to the desired target setup.

1.1.1 MySQL

Configuration and encoding issues

One of the most annoying issues with computing in the whole time frame after maybe 1950 is the prevalence (in the epidemiology sense of the word) of different character encodings—and ezDL is no exception. EzDL uses UTF-8 for transferring information whenever applicable but some infrastructure systems like databases by default don't.

To use MySQL with ezDL, you have to configure MySQL accordingly.

First, set the MySQL server to UTF 8 by adding the following line in the MySQL config (usually found at `/etc/mysql/my.cnf`):

```
[mysqld]
character_set_server = utf8
```

After that character set conversion should work as expected.

Sometimes you will want to implement auxilliary systems (e.g. for automatically adding user accounts). In these software products, the encoding must also be set correctly. There are some configurations of the MySQL server and command line tool that make it very hard to spot problems with the encoding used to insert data into the database. To make sure that your output is actually what is in the database and does not just look correct because your command line client is set so to reverse the wrong encoding used for the data in the database, some variables have to be set. This is what your command line client should look like:

```
mysql> show variables like '%char%';
+-----+-----+
| Variable_name          | Value                               |
+-----+-----+
| character_set_client   | utf8                                |
| character_set_connection | utf8                                |
| character_set_database | utf8                                |
| character_set_filesystem | binary                              |
| character_set_results  | utf8                                |
| character_set_server   | utf8                                |
| character_set_system   | utf8                                |
| character_sets_dir     | /usr/share/mysqlCharsets/         |
+-----+-----+
```

You can set a variable by issuing the following command at the prompt:

```
set <variablename> = <new value>
```

E.g.:

```
set character_set_client = utf8
```

If your output looks as above and also your terminal program is set to UTF-8 encoding, the output of a **SELECT** statement should look correct. If not, it is likely that the data was inserted into the database by an external program such as a web app.

Database creation

After configuring MySQL to use the proper encoding, the database has to be initialized. This section will walk you through the process for the simplest setup that uses one single database on one single system for all agents.

The first step is to create a database as follows.

```
mysql -u <root_user_name> -p
mysql> create database <databasename>;
mysql> grant all privileges on <databasename>.*
to '<ezdlbuser>'@'localhost'
identified by '<ezdlbpasswd>';
```

Please note that `mysql` won't echo your password.

The first line creates the database while the second line creates a user `jezdldbuser` with a password of `jezdldbpasswd`. These values have to be identical with those configured in the properties (see section 2.1).

You are free to choose the name of the database to use but the name has to be the same one as in the properties files that the agents use to initialize themselves (see section 2.1 for details).

The ezDL package provides a set of initialization files that can be used to create the necessary tables. The files reside in the directory `db` at the top level of the ezDL directory tree. Every single file whose name begins with `mysql_` should be piped into the MySQL database as follows.

```
mysql -u <root_user_name> -p <databasename> < <filename>
```

Now the database is initialized and ready to be used.

Chapter 2

Agents

2.1 Configuration

EzDL ships with default configuration files for all agents. These files reside in the directory `examples` at the top level of the ezDL directory tree. A first step is to copy all files in the subfolders `agents` and `wrappers` to the target directory `/home/ezdl/.ezdl`.

The agents and wrappers read their configuration as follows:

1. The internal file `global.properties` is read.
2. The file `/home/ezdl/.ezdl/agent.properties` is read, overriding the definitions in `global.properties`.
3. The properties file given as a parameter at the start of the agent is read, further overriding all previous definitions.

This allows to configure system-wide defaults in `agent.properties` and configure single agents or wrappers to use a different configuration – e.g. for the database.

The example files contain an `agent.properties` that defines the database as follows:

```
db.url=jdbc:mysql://localhost/ezdl
db.user=ezdl
db.password=ezdl
```

This configures to use a MySQL database named `ezdl` on `localhost`, using a user name of `ezdl` and a password `ezdl`, which is, granted, not very creative or secure, but a good start to get things going. These values can be changed anytime, provided they are changed in the configuration *and* the database.

2.2 Starting agents

The easiest way to start an agent is using the supplied `ant` scripts. Three scripts are placed in the top level of the ezDL directory tree: `build-common.xml` contains start information for all agents and wrappers in the package. This file cannot be used for starting as it is included in the other two files. `build-spawn.xml` is a start script that starts agent processes in the background. Doing so has one huge advantage: all agents can be started at once. The huge disadvantage is that there is no way to see any output of the agents except for the log files. The other file, `build-single.xml`, can be used to start exactly one agent because the agent is started in the foreground and blocks until it is halted. Since the agent is started in the foreground, it is possible to see the agent's output on the console. The latter file is mostly used for debugging.

Starting all agents at once is done like follows:

```
ant -buildfile build-spawn.xml start
```

The command results in some output that shows which agents were started but nothing else. To verify the success of the start, the directory agent's web admin page can be used. The location of it is given in `directory.properties`. The files shipped with ezDL have the following configuration in them:

```
webadmin.port=3456  
webadmin.user=ezdl  
webadmin.password=ezdl
```

That means that the web site is located at `localhost`, port 3456 using the given credentials for log-in. You can point your web browser to the location and hopefully see a list of agents that are running.

Chapter 3

Wrapper agents

Wrapper agents are agents that abstract from accessing (remote) resources. Wrapper agents are “normal” agents whose functionality is defined by a `Wrapper` class. Wrapper agents are often called “wrappers” for short, even though that actually only refers to the concrete class that implements the `Wrapper` interface.

Since wrapper agents aren’t so different from the other agents, they are started and configured as such. There are some wrapper-specific configuration options, though, that are outlined in this chapter.

3.1 Types of wrapper agents

There are two different types of wrappers. The first one—called “Solr wrapper”—wraps a Solr server that is normally operated locally. Local Solr servers are useful for caching data that is easily available. This makes accessing the data very fast and reliable. The second kind wraps an external web service. This is done by downloading the web pages from that server and parsing them. The process is facilitated by an internal component called the “toolkit”. This is why this kind of wrappers is called “toolkit wrappers”. There is another difference between Solr wrappers and toolkit wrappers: when a Solr wrapper has a code defect that leads to abnormal behavior like ongoing multiple requests, usually only your own resources suffer from the load. When a toolkit wrapper freaks out, somebody else’s resources suffer from what might seem to them as a DoS attack. See section `æixorq-tikloot` for a way to make that less likely.

3.2 Toolkit wrappers and web proxies

Toolkit wrappers post requests for web resources to remote web sites and parse the responses. This might lead to problems if a wrapper has a defect that leads to hammering the remote site with requests. To prevent this from happening, caching the web requests can be used. Since the wrappers' requests often contain a query string, most proxies will not cache these resources out-of-the box. Additional configuration has to be done to both wrapper and proxy.

3.2.1 The wrapper

To tell a wrapper to use a web proxy, the following lines have to be included in the wrapper's properties:

```
toolkit.proxy.host=<hostname>
toolkit.proxy.port=<port number>
```

E.g., to configure a wrapper to use a proxy at port 8080 on `localhost`, one would add:

```
toolkit.proxy.host=localhost
toolkit.proxy.port=8080
```

The wrapper then has to be restarted since the properties file is only read at the start.

3.2.2 Squid

Squid is a free-as-in-speech web proxy. Getting a Squid proxy to cache resources whose URL's have query strings, the following changes to the `squid.conf` have to be performed:

There should be two lines that read

```
acl QUERY urlpath_regex cgi-bin \?
cache deny QUERY
```

These lines basically assign the name `QUERY` to URLs that contain either the key word `cgi-bin` or a question mark and then tells Squid to not cache resources whose URL match this pattern. Since this is exactly what we do not want, remove or comment-out these lines.

Further, Squid needs to know when to refresh cached resources. Find a proper spot for a new line by looking for lines starting with `refresh_pattern` and change the lines so that the last two lines read:

```
refresh_pattern -i (/cgi-bin/|\?) 5 50% 1440
refresh_pattern . 0 20% 4320
```

The last line containing the full stop (“.”) may contain whatever values it had before. The first line in the above listing have the following meaning. First, the pattern is defined to match both URLs that have a part `/cgi-bin/` or a question mark in them, which indicates that they contain dynamic content. The first number, 5, then defines that the minimum time to keep that resource in the cache is 5 minutes. The next number, 50%, means that a resource is not refreshed, if it has been in the cache less than 50% of its age (time since last change) if it has no explicit expiry time. The last number, 1440, means that resources are refreshed if they are at most 1440 minutes (a day) in the cache. These numbers are by no way god-given. Adapt them to suit your needs.

The file `/var/log/squid/access.log` can be inspected to see if the configuration works as intended. The following snippet shows a resource that was not cached:

```
134.91.35.179 - - [17/Apr/2010:09:18:20 +0200]
"GET http://portal.acm.org/results.cfm? HTTP/1.1" 200 165084
"--" "Apache-HttpClient/4.0 (java 1.5)" TCP_MISS:DIRECT
```

Please note the `TCP_MISS`, that indicates a cache miss.

If the configuration leads to items being cached, lines like the following should be found in the log:

```
134.91.35.179 - - [17/Apr/2010:09:19:48 +0200]
"GET http://portal.acm.org/results.cfm? HTTP/1.1" 200 165121
"--" "Apache-HttpClient/4.0 (java 1.5)" TCP_HIT:NONE
```

The dead give-away here is the `TCP_HIT`, which means that there was a cache hit for that resource.

Please note that the search agent maintains an internal cache for queries and their result lists. So to test the cache, you not only have to restart the Squid daemon but also the search agent. Otherwise the toolkit wrapper isn't even asked again.

In order to see the query string of the URL, set the following option in the Squid config:

```
strip_query_terms off
```

This clearly has privacy implications as all queries sent to remote sites are logged in full. So this setting should only be used for debugging and testing and not in a production environment.

Chapter 4

Deployment

4.1 Signing GFrameDL application

Signing JAR files is especially recommended for deployment via Java Web Start if the users should not be scared by warning dialogs.

4.1.1 Key stores

A so called key store is required for signing JAR files.

4.1.2 Creating a key store

```
keytool -import -trustcacerts -alias <alias> -file <certificate> -keystore <keystore>
```

4.1.3 Converting a PCKS12 key store to a default Java key store

```
keytool -importkeystore -deststorepass testtest -destkeypass keypass -destkeystore my
```

4.1.4 Signing JARs

With the generated key store (see previous section) the GFrameDL JAR file can be signed. A bash script can be found in the ezDL Mercurial repository (jnlp/ezdl-sign-jar). This script signs JAR files and deploys them via scp to a remote host.

```
tbeckers@artus:~$ ./ezdl-sign-jar
JAR file to sign (default is gframedl.jar):
Signed JAR file (default is gframedl-signed.jar):
Keystore (default is /home/tbeckers/.ezdl/keystore):
Host name to where the signed JAR will be copied (default is ezdl.de):
Final target of signed JAR file (default is /var/www/webstart):
jarsigner -keystore /home/tbeckers/.ezdl/keystore -signed-jar gframedl-signed.jar
Enter Passphrase for keystore:
scp gframedl-signed.jar ezdl@ezdl.de:/var/www/webstart
```