Java Connector Server

Java Remote Connector Server is using the same kind of connectors that midPoint itself is using. The Java Remote Connector Server is used in situations where a connector needs a local access to some resource to be able to work with it. It is usually used for connectors that require local access to files such as CSVFile Connector (legacy). This avoids the need to copy the file using FTP or a similar mechanism which is difficult to do right and it is quite error-prone (e.g. problems with partially downloaded files, error handling, atomicity, etc.) Java Remote Connector Server is also used in situations that require firewall traversal or securing insecure communication protocol.

Requirements

- Java SE 6 or later

Download

Version	Download	Sources	Note
1.4.2.12	ZIP	https://github.com/Evolveum/ConnId/tree/connid-1.4.2.12	
1.4.0.49	ZIP	https://github.com/Evolveum/ConnId/tree/connid-1.4.0.49	
1.1.1.0	ZIP	OpenICF 1.1.1.0 tag	obsolete version, probably won't work with current midPoint

You may also download from the OpenICF download page.

Installation

1. Download and unzip the binary distribution (or clone git repo with sources and build your own with *mvn clean install* command)
2. In the installation folder (that contains *bin*, *conf*, and *lib* directories) create a directory for connector bundles, named *bundles*. In the following text, we assume */opt/connid-connector-server directory* for Linux.
3. Copy connectors you need into *bundles* directory (e.g. *connector-csvfile-1.4.0.49.jar* for CSV connector)
4. Set the secret key by invoking the command:
    a. (on Windows): *bin\ConnectorServer.bat /setkey <your secret key here>*
    b. (on Linux):
    java -cp "lib/framework/connector-framework.jar:lib/framework/connector-framework-internal.jar:lib/framework/groovy-all.jar" org.identityconnectors.framework.server.Main -setKey -key <your secret key here> -properties conf/ConnectorServer.properties
5. Fix the logging configuration:
    a. replace the line "*connectorserver.loggerClass=org.identityconnectors.common.logging.slf4j.SLF4JLog*" in *conf/ConnectorServer.properties* file with "*connectorserver.loggerClass=org.identityconnectors.common.logging.impl.JDKLogger*"
    b. add "*-Djava.util.logging.config.file=conf/logging.properties*" to your startup parameters to actually use logging
    c. update the conf/logging.properties to log to file in *logs* directory:

```
handlers=java.util.logging.FileHandler
##handlers=java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
java.util.logging.FileHandler.pattern = logs/connectorserver%u.log
java.util.logging.FileHandler.limit = 102400
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
java.util.logging.FileHandler.append = true
.level=INFO
```

6. Run the connector server e.g. by invoking the command:
    a. (on Windows): *bin\ConnectorServer.bat /run*
    b. (on Linux): *java -cp "lib/framework/connector-framework.jar:lib/framework/connector-framework-internal.jar:lib/framework/groovy-all.jar" org.identityconnectors.framework.server.Main -run -properties conf/ConnectorServer.properties*

Connector Server will run on foreground/console. CTRL+C will stop it.

Automatic Server Startup

Systemd

Create user/group for running the service (e.g. connid, connid). This user must have access to the connector server files.

Create systemd service file /etc/systemd/system/java-connector-server.service (as root) - inspiration from http://stackoverflow.com/questions/21503883/spring-boot-application-as-a-service/22121547#22121547:

```
[Unit]
Description=Java Connector Server Service
[Service]
User=connid
WorkingDirectory=/opt/connid-connector-server
ExecStart=/usr/bin/java -Xmx256m -cp "lib/framework/connector-framework.jar:lib/framework/connector-framework-
internal.jar:lib/framework/groovy-all.jar" org.identityconnectors.framework.server.Main  -run -properties conf
/ConnectorServer.properties
SuccessExitStatus=143
[Install]
WantedBy=multi-user.target
```

Issue the following commands (as root):

```
systemctl daemon-reload
systemctl enable java-connector-server
```

You can start/stop the service using:

```
systemctl start java-connector-server
systemctl stop java-connector-server
```

SysV Init

Create start script to be run by startup script */opt/connid-connector-server/start*.

```
#!/bin/bash
MAIN_DIR=/opt/connid-connector-server
cd $MAIN_DIR
exec java -Djava.util.logging.config.file=conf/logging.properties -cp "lib/framework/connector-framework.jar:lib
/framework/connector-framework-internal.jar:lib/framework/groovy-all.jar" org.identityconnectors.framework.
server.Main -run -properties conf/ConnectorServer.properties
```

Set file permissions:

```
chmod 755 /opt/connid-connector-server/start
```

Create startup script */etc/init.d/connid-connector-server* - inspiration from: https://orrsella.com/2014/11/06/initd-and-start-scripts-for-scala-java-server-apps/

```
#!/bin/bash
START_SCRIPT=/opt/connid-connector-server/start
PID_FILE=/var/run/connid-connector-server.pid
DAEMON=$START_SCRIPT
start() {
  PID=`$DAEMON $ARGS > /dev/null 2>&1 & echo $!`
}
case "$1" in
start)
    if [ -f $PID_FILE ]; then
        PID=`cat $PID_FILE`
        if [ -z "`ps axf | grep -w ${PID} | grep -v grep`" ]; then
            start
        else
            echo "Already running [$PID]"
            exit 0
        fi
    else
        start
    fi
    if [ -z $PID ]; then
        echo "Failed starting"
        exit 3
    else
        echo $PID > $PID_FILE
        echo "Started [$PID]"
        exit 0
    fi
;;
status)
    if [ -f $PID_FILE ]; then
        PID=`cat $PID_FILE`
        if [ -z "`ps axf | grep -w ${PID} | grep -v grep`" ]; then
            echo "Not running (process dead but pidfile exists)"
            exit 1
        else
            echo "Running [$PID]"
            exit 0
        fi
    else
        echo "Not running"
        exit 3
    fi
;;
stop)
    if [ -f $PID_FILE ]; then
        PID=`cat $PID_FILE`
        if [ -z "`ps axf | grep -w ${PID} | grep -v grep`" ]; then
            echo "Not running (process dead but pidfile exists)"
            exit 1
        else
            PID=`cat $PID_FILE`
            kill -HUP $PID
            echo "Stopped [$PID]"
            rm -f $PID_FILE
            exit 0
        fi
    else
        echo "Not running (pid not found)"
        exit 3
    fi
;;
restart)
    $0 stop
    $0 start
;;
*)
    echo "Usage: $0 {status|start|stop|restart}"
    exit 1
esac
```

Set file permissions:

```
chmod 755 /etc/init.d/connid-connector-server
```

Start the service:

```
/etc/init.d/connid-connector-server start
```

Set the service to autostart (using your distribution command; here Red Hat-based distributions "chkconfig" is used:

```
chkconfig connid-connector-server on
```

You may need to use different command and edit the script to use dependencies or service startup ordering.

Original instructions for OpenICF Connector Server: http://openicf.forgerock.org/connector-framework-internal/connector_server.html

Configuring SSL

The Connector Server is a SSL server. Therefore is needs a keypair (private key + certificate). Java connector server expects the keypair to be present in a keystore. It is using standard Java JCE keystore for this purpose. The keystore does not exist at the time of the initial installation. It needs to be created and populated with a keypair.

Creating and Populating a Keystore

The keypair is usually distributed in a PKCS#12 format (a file with `p12` or `pfx` extension). This format needs to be converted in Java JCE keystore. There is `keytool` utility that is part of Java platform that can be used for conversion:

Converting PKCS#12 key and certificate to java keystore

```
keytool -importkeystore -srckeystore mykeycert.p12 -srcstoretype pkcs12 -destkeystore keystore.jks -
deststoretype JKS
```

The command above creates a `keystore.jks` file which is the actual Java JCE keystore. The `keytool` command will ask for two passwords:

- A password on the PCKS#12 files as these files are usually protected by password (because they contain a private key)
- A password for a newly created keystore. Make sure you remember this.

But there is a catch. The Java JCE keystore as a whole is protected by a password. But also each individual key is protected by a password. These passwords are usually the same and that is exactly what the connector server expects. However when the keystore is converted from PCKS#12 the keystore password is set to the supplied password but the key password remains the same as was the password on PCKS#12 file. If these passwords were not the same then the key password needs to be changed in one extra step:

Changing a key password

```
keytool -keystore keystore.jks -storepass changeit -keypasswd -alias mykey
```

See Keystore Configuration page for some more tips and tricks dealing with keystore. But please note that this page deals with **midPoint keystore** which is slightly different than **Connector server keystore**.

Passing Keystore Parameters to Connector Server

The connector server is a Java application that looks for a default keystore. The location, type and password of the default keystore needs to be passed to the connector server in a form of Java options:

```
java ... -Djavax.net.ssl.keyStore=keystore.jks -Djavax.net.ssl.keyStorePassword=changeit -Djavax.net.ssl.
keyStoreType=JKS ...
```

Add these options to the script that is starting connector server.

Enabling Connector Server SSL

Change the `connectorserver.usessl` option to `true` in the `connectorserver.properties` configuration file.

You can start the server now. Please do not forget to configure the midPoint side as well.

Troubleshooting

Error "Cannot recover key": Make sure that the key password in the keystore is the same as the keystore password.

See Also

- Connector Server