Getting Starting with OSCARS 0.6

April 19, 2011

Introduction

This is the more detailed of two documents intended to guide a user in setting up an OSCARS 0.6 installation. The other document, QuickStart.pdf, lists the steps needed to go from either an SDK binary releases or from the latest sources. This document explains in more detail how to perform some of the steps and what the various configuration files are and how you may want to customize them for your site.

Installing MYSQL

OSCARS uses mysql version 5. If mysql v5 or later is not installed, you can use the following commands to install mysql. You need to be root to execute these commands.

On a Centos machine, mysql can be installed using the following commands

```
yum install mysql-server
yum install mysql
yum install mysql-dlevel
```

On a Ubuntu machine, mysql can be installed using the following commands

sudo apt-get install mysql-server

On Redhat Enterprise Linux machine, mysql can be installed using the following command
up2date install mysql-server

If you are downloading mysql and manually installing it, the instructions for installation for your specific package can be found on the oracle mysql webpage.

Initializing MYSQL

If mysql has just been installed on your machine you must run **`mysql_install_db --user mysql`**, where mysql is the user id under which mysql will run to initialize it for your host.

To have mysqld start up at boot check if there is a mysqld start file in init.d. If not, you should copy support-files/mysql.server to the right place for your system.

The daemon can be started with **`/usr/bin/mysqld_safe &`**

To set the password for the mysql root user

```
/usr/bin/mysqladmin -u root password 'new-password'
/usr/bin/mysqladmin -u root -h <localhost> password 'new-password'
```

The log file is in /var/log/mysqld.log and is readable by user and group mysql only. For mysql ver 5.5 the log is in /usr/local/mysql/data.

Create Databases

If you are setting up OSCARS on a fresh machine, make sure mysql is installed.
If the mysqld isn't running look at the instructions above.

Before the servers can be used for real requests, the databases authn, authz and rm need to be created and initialized. The script **$OSCARS_DIST/bin/deployOscarsSrc.sh** will do this among other things. It calls the script **$OSCARS_DIST/bin/oscarsdb** to do the following:

`oscarsdb init`
: creates the mysql user 'oscars'@'localhost'' user with a password of 'mypass'. This matches the value for the default user in {authN, authNPolicy, authZ, authZPolicy, resourceManager} /config/*.yaml.template files.
See note below on how to change the default password.

`oscarsdb ct`
: creates and populates the default tables.

`oscarsdb rt`
: removes all the tables.

If you have not set a password for mysql root edit the script to use -u $SQLROOT rather than –u $SQLROOT -p

If you wish to have the resourceManager tables populated with a set of anonymized reservation data use:

```
mysql -u root < resourceManager/sql/createAnonTables.sql
```

If you are running on a machine that already has OSCARS 0.5 databases, you can use the updgradeTables0.5-0.6.sql scripts to create the new tables and copy over the entries from the old tables. These scripts can be found in the {authN,authZ,resourceManager}/sql directories.

Note: To change the default oscars@localhost myslq password
Anyone who has login access to the host on which the mysql server is running and knows the oscars mysql password can modify the OSCARS database files with mysql commands. Thus before you put any critical user or reservation information in those files, you should change the default password and protect the files that contain it to the user that the services will run as. The distribution includes the following files:

authN/config/authN.yaml.template
authNPolicy/config/authNPolicy.yaml.template
authZconfig/authZ.yaml.template
authZPolicy/config/authZPolicy.yaml.template
resourceManager/config/config.yaml.template

The deployOscarsSrc.sh script will copy the yaml.template files to .yaml files if such files do not already exist. To change the default password edit the *.yaml files and change the protection of those files to be readable only by the userId that the service runs under. Each of the 5 yaml files must have the same password. Once you have changed the password that the OSCARS software will use, you must change the password in the mysql database. To do this:

```
mysql –u Oscars –p mypass
SET PASSWORD = password('newpassword');
quit;
```

Configuration

set the following environment variables:
$OSCARS_DIST - source directory
$OSCARS_HOME - deployment directory

Each service has the following configuration files in <service>/config.

manifest.yaml
contains the path names of the configuration files needed by that service. Variations of the configuration files are selected for UNITTEST, SDK, DEVELOPMENT and PRODUCTION contexts. The context value is held in the ContexConfig class and should be defined by the program that starts the service (often named Invoker, e.g. es.net.oscars.coord.common.Invoker.java). The startServer.sh scripts take a command line argument and pass it to the invoker. For standalone clients, the context needs to be defined in whatever script or program is being used to call the client interface.

{service>|config}.yaml.[template]
contains the publishTo address and optionally hibernate user values. If a yaml.template exists, it contains dummy hibernate oscars-user and password values. It will be copied to a .yaml file if one does not exist. The .yaml file should be edited to match the values in the local database.

log4j.properties – variations for DEBUG, MESSAGES, INFO
contains log4j logging properties

server-cxf.xml - variations for SSL or HTTP
contains https configuration, enables cxf message logging

client-cxf.xml - variations for SSL or HTTP
contains https configuration, enables cxf message logging

<service>.cxf.xml
contains the hibernate database configuration if the service uses a database

The following scripts are used to edit and install configuration files into the $OSCARS_HOME area.

bin/exportconfig

copies the config files to $OSCARS_HOME/<ServiceName>/conf and edits some file names to match $OSCARS_HOME. **Does not replace any files that already exist in $OSCARS_HOME**. If you want to get new versions of the configuration files deployed, you must delete any existing ones in $OSCARS_HOME before running bin/exportconfig.

sampleDomain/bin

contains scripts to create the keystores that are needed by https and message signing. It also edits all the servers' deployed client/server-cxf files to use the correct pathnames for the keystores. Any time you do an exportconfig for a server, it must be followed by sampleDomain/bin/exportconfig. The exportconfig scripts are not run automatically, but can all be run from the top by $OSCARS_DIST/bin/exportconfig which will run sampleDomain last.

Building from source

The OSCARS sources are kept in an svn repository at *https://oscars.es.net/repos/oscars*.
The latest version is at *https://oscars.es.net/repos/oscars/trunk*.
Releases are at *https://oscars.es.net/repos/oscars/branches/<versionNum>*.

OSCARS uses maven to build and deploy. The pom in the top level directory, $OSCARS_DIST, defines a parent project. Currently it defines three shared properties: cxf.version, java.version and oscars.version.
Running mvn compile in OSCARS will build everything except sampleDomain and template. The first time maven is run, it will download all the third party dependencies. So far there have been no major problems with missing classes at least when Maven 2.x is used.

Define the environment variable **OSCARS_DIST** to be the directory that the sources are in.
Define the environment variable **OSCARS_HOME** to be the directory where you want the servers deployed.

```
cd $OSCARS_DIST

bin/deployOscarsSrc.sh <oscarsVersion> (e.g 06-20110603)

    checks out a copy of the sources from the svn repository at
    https://oscars.es.net/repos/oscars/trunk.

    copies template configuration files to non-template ones
```

do any edits to the configuration files in $OSCARS_DIST that are needed for your site

```
bin/exportconfig     -- copies configuration files to $OSCARS_HOME

mvn install [-DskipTests]
```

compiles, initializes test databases, runs unit tests (unless skipTests was specified) and installs the maven artifacts in your local repository. It will finally run the bin/exportconfig script which copies the various servers' configuration files to $OSCARS_HOME and creates necessary default keystores.

Use the deployOscarsSrc.sh script the first time you build the system. Subsequent builds can be done with just

```
svn update
bin/copyTemplates.sh  (if any template files have changed)
```

```
mvn install
```

Running

mvn install creates the jar and one-jar files in the directories, $OSCARS_DIST/<service>/target, and puts them in your maven repository. It then runs all the unit tests. You can use the `–DskipTests` switch to mvn to skip the unit tests, or the –Dmaven.test.failure.ignore=true to run the tests but ignore all failures. Note: the Coordinator unit tests will fail if there is already a Coordinator service running.

A server is run by the command **`java -jar target/<service>-$Oscars.version.one-jar.jar`**. There are **`<service>/bin/startServer.sh`** scripts for each service that contain this command and some useful options. The scripts **`$OSCARS_DIST/bin/{startServers.sh,stopServers.sh}`** will start and stop servers. The testServers will check to see that all the servers have been started and are listening on their service ports.

```
startServers.sh <context> <server >
```
<context> is one of: PRODUCTION | pro UNITTEST | test DEVELOPMENT | dev SDK | sdk
<server> is either ALL or one or more of:
authN authZ api coord topoBridge rm stubPSS lookup wbui stubPCE
bwPCE connPCE dijPCE vlanPCE nullAGG stubPSS notifyBridge wsnbroker

```
stopServers <server>
```
<server> is either ALL or one or more of:
authN authZ api coord topoBridge rm stubPSS lookup wbui stubPCE
bwPCE connPCE dijPCE vlanPCE nullAGG stubPSS notifyBridge wsnbroker

```
testservers <CONTEXT>
```
<context> is one of: PRODUCTION | pro UNITTEST | test DEVELOPMENT | dev SDK | sdk

Note: it can take up to several minutes for all the services to be fully initialized. It is recommended to run on a host with at least 4G of memory. Occasionally the OSCARSService (api) will fail to register itself with the Lookup Service because the Lookup Service was not started in time. In this case, you can just stop the OSCARSService and restart it once the LookupService is responding

```
stopServers.sh  api
startServers.sh <context> api
```

Testing

Currently the api, authN, authZ, coordinator and resourceManager bin directories contain scripts to run client test programs. To test the whole system, only the commands in api/bin are needed. The other scripts can be used to test just their specific services. The scripts contain comments on how to run them and can be called with a –h option for more information. These scripts use the library jars in <service>/target/tmp/lib which is created from the latest one-jar.jar.

The scripts in api/bin are:

```
createRes.sh –pf <paramFile>
modifyRes.sh –pf <paramFile>  (not tested completely yet)
cancelRes.sh –gri <gri>
query.sh –gri <gri>
list.sh –n <numReq> -o <offset> -st <status>
setupPath.sh –gri <gri>
teardownPath.sh –gri <gri>
```

All the commands also take an optional context parameter: –C sdk | dev | pro which defaults to dev (DEVELOPMENT). If you are running the servers in PRODUCTION context, you must also run the clients in PRODUCTION mode since it requires https connections. SDK and DEVELOPMENT contexts are compatible, but SDK will not print the messages sent to the server.

Some sample paramFiles can be found in api/src/test/resources/*.yaml

- autoTD1.yaml – creates a reservation of 4 minutes duration, starting immediately in testdomain-1. It should work with the default installation.
- autoTD2.yaml – like TD1 except it creates the reservation in testdomain-2. To use this you must change the localDomain:id to testdomain-2 in topoBridge/config/config.yaml
- signalTD1.yaml – creates a signal-xml reservation in testdomain-1 that will be exist for 5 days. You can use this reservation to test the setupPath and teardownPath operations.
- autoTD3TD4.yaml – creates a mutli-domain reservation starting in testdomain-3 and ending in testdomain-4.

Multi-domain testing

To test a multi-domain reservation you will need to have two OSCARS IDC services running: the first should have the localDomain:id in utils/config/config.yaml set to testdomain-3, the second should have testdomain-4 as its localDomain:id. An OSCARSService finds a peer service for another domain using a Lookup Service. The lookup service that we distribute is a bridge server can contact an external Lookup service and cache the URLs of services for peer domains. Since we are not currently deploying and external service, we need to add the service/domain information directly to the Lookup Service cache. This is done for testdomain-3 (running on host3) and testdomain-4 (running on host3) by the following commands:

On the host3 that is serving testdomain-3:

```
cd $OSCARS_DIST/lookup/
bin/oscars-idcadd -d testdomain-4 -p http://oscars.es.net/OSCARS/06
      -l http://host4:9001/OSCARS
```

On the host4 that is serving testdomain-4:

```
cd $OSCARS_DIST/lookup/
bin/oscars-idcadd -d testdomain-3 -p http://oscars.es.net/OSCARS/06
      -l http://host3:9001/OSCARS
```

These commands will add the appropriate entries to $OSCARS_HOME/LookupService/data and only need to be repeated if you delete that directory.

The two servers also need to share a common root CA and RA, so that the signed messages that are passed between them can be verified. To accomplish this:
On the primary IDC domain, e.g testdomain-3, when first running sampledomain/bin/gencerts use an empty $OSCARS_HOME/sampleDomain/certs which will cause a new CA, RA, client, localhost and oscarsidc keystores to be created. On the second IDC, testdomain-4, copy the sharedCred.tar file that was created in the first domain into $OSCARS_HOME/sampleDomain/certs on the second host. Then this command will use the shared RA to generate and sign a new oscarsidc certificate and keystore.

Using the WBUI

If you wish to use the web interface to manage users and their privileges you must first create an administrative user. To do this:

```
cd $OSCARS_DIST/tools
mvn install
bin/idc-useradd
```

The idc-useradd commad will prompt for all the necessary values. Be sure to grant your user the OSCARS-administrator attribute (1). If you wish to create, query or list reservations from the WBUI, you should also give this user the OSCARS-engineer attribute.

Verify that the authN , authZ and wbui servers are running and use your browser to go to https:/localhost:8443/OSCARS. Once there you will need to login as user you just created.

Currently you can use the browser interface to manager users, create, query or list reservations.

Customizing your configuration

Use of manifest.yaml

The manifest.yaml file specifies the configuration files to be used in a given context. All the servers need to be run in the same context so that they can establish connections when necessary, but each server can independently set its configuration choices for each context. For example, in the SDK context all the servers run on http, but in the DEVELOPMENT context AuthN and the WBUI run on https while the other servers run on http. In a PRODUCTION context all the services will run by default over https, but a site can change that to http for any services that are run on a common host. You can also edit the manifests to determine the amount of logging you want for each service.

Customizing Logging

Getting the right amount of logging is often a delicate balance, so we have provided ways to customize the logging in the OSCARS configuration files. In our default logging setup all the log4j messages (root logging) are put into per service .out files in $OSCARS_DIST. In addition, only the OSCARS specific messages are put into per service log files in $OSCARS_HOME/logs. We have provided three default logging profiles:
INFO: intended for production systems that only logs *info* level log messages and in addition puts the log messages from all the services into $OSCARS_HOME/oscars.log
DEBUG: which logs the *debug* level log messages
MESSAGES: which logs *debug* level log messages and enables the cxf message logging which will put the all the inter-service messages in the .out files.

By default, the PRODUCTION context uses the INFO log files, SDK uses DEBUG level and DEVELOPMENT uses MESSAGE. Note that the message logging can be very verbose for the coordinator/pce messages as they contain the entire topography. One can change the log level used by any service, by editing that service's manifest file.

Once a log4j.*.properties file has been copied to $OSCARS_HOME, it can be site-edited there. It will not be overwritten by the distributed version unless it is deleted first.

Two useful customizations are:

1. To add or delete message logging by editing the manifest.yaml file.
2. To decrease the number of log messages by editing the $OSCARS_HOME/<ServiceName/conf/log4j.*propertiesfile by changing the line "log4j.logger.net.es.oscars=DEBUG, <logName>"to "log4j.logger.net.es.oscars.<serviceName>=DEBUG, <logName>" which will eliminate the log messages from the utils classes. <serviceName> matches the name of the source directory for the service: e.g. api,coordinator,authN.

Gotchas

If you modify code in utils or common-soap you need to install it and then do a *mvn clean*, *mvn install* in all the services that import code or extend classes from there.

mvn install runs all the unit tests. Some of the units tests, e.g. coordinator, do not succeed unless other servers are running. You can use the maven defines:

`-DskipTests` - to skip all tests
`-Dmaven.test.failure.ignore=true` - to run tests but continue even if there are failures

Modifying wsdl and xsd:
mvn install will run wsdltojava only when the wsdl file has been modified. It does not know about any of the include files, such as <service>.xsd. or the oscars common files. So if you have changed anything in a service interface, you need to do a "touch" on the wsdl file, and then run *mvn install* in common-soap. Another shortcut is to delete the common-soap/cxf-codegen-plugin-markers/.src_main_resources<module>.DONE file.
The best way to be sure that an interface change will take effect is to run *mvn clean* followed by *mvn install* from the OSCARS directory which will delete all the generated java code, the compiled code and targets, and then rebuild the whole system.

Note that all the wsdl and xsd files are in `$OSCARS_DIST/common-soap/src/main/resources.`

Directory layout

There is a directory and maven project for each service. The AuthN and AuthZ services each provide two interfaces, e.g. AuthNService and AuthNPolicyService.

All the schemas are in common-soap/src/main/resources/<service>. The generated java classes are in common-soap/src/main/java/. No part of this java tree should be committed to SVN.

api contains the external interface to OSCARS. It defines the 0.5 and 0.6 interfaces and an internal-api that is used for forwarding messages between domains. Only messages in this interface are signed. It also contains the wsdl and xsd files for the PCE interface, since that interface is considered to be part of the public OSCARS interface and will be versioned to provide backwards compatibility.

sampleDomain contains scripts to set up the keystores needed for https and message signing
authNStub, trivCoord, stubPCE, and stubPSS are examples of the interfaces. They are not run when the real modules are used.

template is a very out-of-date example of how to structure a service.

Port usage

These values are defined in the files <service>/config/config.*.yaml. They may vary depending on the context and may be assigned different values by a site.

The following are the values that are set in the config files that we distribute.

api: 9001, 9002 (internal)
authN: 9090
authNPolicy: 9004
authNStub: 9011
authZ: 9190
authZPolicy: 9005

coordinator: 9003
coordinator:pceRuntime: 10000
lookup: 9014
notify: 9012
PCEs
	stubPCE 9007 (only run if no other PCE's are run)
	connectivityPCE 9007
	bandwidthPCE 9009
	vlanPCE	9010
	dijkstraPCE: 9008
resourceManager: 9006
PSS: 9050
topoBridge: 9019
wbui:http 8080
wbui:https 8443