How to build CloudStack

Starting from 4.1.0, a.k.a, the master branch, we are using Maven to build all the artifacts. If you are interested in how to build cloudstack on 4.0.x branch, please refer to the 4.0 instructions.

Maven uses the notion of a build life-cycle to which plugins can attach. Plugins are similar to Ant tasks. When a Maven build is invoked, we specify a point in the life-cycle up to which the build should proceed. The compile phase comes before test, and test comes before package, and package comes before install. Once we have Maven setup, we can invoke the Struts build, and specify which phase the build should use.

- TODOs
- Installing Maven
    - XML indentation
    - Understanding POMs
        - What is -SNAPSHOT and why should I care
- Dependencies
- Building CloudStack
    - Deploying Database
    - Running a server for test/debug purposes.
- Integrated Simulator+Marvin test
- Building Specific Versions of CloudStack
- Packaging
    - Building RPM packages
    - Building DEB packages
- Development with a IDE
    - Good Ol' Terminal
    - IntelliJ IDEA
    - Eclipse
- Other Uses
    - RAT: ASF License Checking
    - Site Generation and Reporting
    - Versioning / Releases
    - Skip unit test
- Troubleshooting
- Install the SystemVM Template
- Remote debugging
    - Before CS 4.11 (tomcat7)
    - From CS 4.11 onwards (jetty):

TODOs

1. Deploydb:
   - see if we can use http://code.google.com/p/flyway/ etc. for database upgrading/migration during development (suggested by John Burwell <jburwell@basho.com>)
2. Compile:
   - fix tomcat webapp compilation issue
3. Unit tests:
   - fix unit tests so they are more useful during compilation
4. Deploy and Debug:
   - configure tomcat plugin to do tomcat:deploy and tomcat:run, debug
5. Site/doc generation with maven

Installing Maven

We need maven 3, you can either install it from distribution, or install it from maven binary. Following is the steps to install it from maven binary:

Get Maven binary: http://maven.apache.org/download.html

For Mac and Linux:

Extract the latest stable binary release to /usr/local/maven

Add following to your .zshrc/.bashrc/.aliasrc:

```
export M2_HOME=/usr/local/maven
export PATH=${M2_HOME}/bin:${PATH}
```

*For Windows: *

```
Right click on my computer and click on Properties
Click on Advanced system settings
Click on Environment Variables
Click on New... under System variables
Add M2_HOME to Variable name and /usr/local/maven to Variable value
Find Path and click Edit...
Add ;%M2_HOME%/bin at the end
Click OK, OK, and close to get out of all the dialogues
Locate a copy of genosimage.exe.  It comes with cygwin and can be copied from %CYGWIN_HOME%\bin
copy %CYGWIN_HOME%\bin\genosimage.exe %SOURCE_HOME%\mkisofs
```

XML indentation

Two spaces for XML files, which can be enforced by editors.

In Eclipse,

```
Window -> Preferences -> XML -> Editor -> select "Indent using spaces", also set "Indentation size 2 spaces"
```

In VIM:

```
# sudo apt-get install libxml2-utils
# .vimrc
au FileType xml setlocal equalprg=xmllint\ --format\ --recover\ -\ 2>/dev/null
# gg=G
```

Understanding POMs

POMs are the maven project configuration files. Here you can configure everything from mailinglists that belong to the project, compilation of the various pieces of the project all the way to the final deployment of release binaries.

To debug POMs, read through the effective POM:

```
$ mvn help:effective-pom
```

To see how the dependencies are related to the modules:

```
$ mvn dependency:tree
```

As with anything, feel free to browse the documentation on maven: This guide is very helpful.

What is -SNAPSHOT and why should I care

-SNAPSHOT is a special marker in maven that tell everybody that this particular piece of code is under development and might change at any time. Opposed to a normal versioned release like '4.0.0', a snapshot is never a stable release. Maven will often try to get a more recent version of a SNAPSHOT during compile. A stable release is fixed, once published there is no changing it anymore. Snapshots are normally used to indicate that a build is made of the HEAD or trunk of a project. Later on this page more about releases using maven.

Dependencies

Some modules, e.g. vmware/netscaler/netapp, require dependencies that are not freely available or have an incompatible license. These dependencies you need to download yourself. See to following instructions to add them to your maven repository.

1. You can download the following jars and put them in the deps directory:

- cloud-iControl.jar http://zooi.widodh.nl/cloudstack/build-dep/cloud-iControl.jar
- cloud-manageontap.jar http://zooi.widodh.nl/cloudstack/build-dep/cloud-manageontap.jar
- vmware-vim.jar http://zooi.widodh.nl/cloudstack/build-dep/vmware-vim.jar
- vmware-vim25.jar http://zooi.widodh.nl/cloudstack/build-dep/vmware-vim25.jar
- vmware-apputils.jar http://zooi.widodh.nl/cloudstack/build-dep/vmware-apputils.jar

As of 4.3 you should no longer need to download the following netscaler jars. You may receive an error in the install-non-oss.sh script as it will still look for these files.

- cloud-netscaler.jar, cloud-netscaler-sdx.jar http://zooi.widodh.nl/cloudstack/build-dep/cloud-netscaler-jars.zip

In case of 4.2/master, Min suggests on ML: To build non-oss build, you need to first download Vmware 5.1 SDK from https://my.vmware.com/group/vmware/get-download?downloadGroup=VSP510-WEBSDK-510 (Version: 5.1, Release-date: 2012-09-10, Build: 774886) to a temp directory. This is a zip file, unzip file and you will see a *vim25.jar* in *<unzip folder>/SDK/vsphere-ws/java/JAXWS/lib*. Place this *vim25.jar* in deps folder and rename it as vim25_51.jar, then run: deps/install-non-oss.sh to install it into your m2 repo.

2. Go into the deps directory and run install-non-oss.sh

```
$ cd deps
$ mv cloud-manageontap.jar manageontap.jar
$ mv vmware-apputils.jar apputils.jar
$ mv vmware-vim.jar vim.jar
$ mv vmware-vim25.jar vim25_51.jar
$ unzip cloud-netscaler-jars.zip
$ ./install-non-oss.sh
```

Building CloudStack

Running the maven command in the top directory of the CloudStack sources will compile all sources for CloudStack. The resulting jar files will be in the directory target in the subdirectory for a particular module. The default build will have all components that depends on non opensource (non-oss) libraries disabled. If you want to enable this, just add -D noredist to the mvn command line or see the table below for more fine-grained options. The install goal will compile all sources, make war and jar files and add them to your local repository.

The CloudStack build process uses maven profiles. The following table shows the more common profiles that are defined:

Available profiles	Enabled by property	Requires profiles	Description
deps			Enables a convenience download for all dependencies into the deps directory, usage mvn -P deps -pl deps install
developer			Enables a convenience pom with developer functions
vmware	noredist		Enables the build of vmware-base and the vmware plugin, requires vmware SDK to be present. See also Hypervisor VMWare.
netapp	noredist		Enables the build of the netapp plugin, requires NetApp manageontap sdk.
f5	noredist		Enables the build of the f5 plugin, requires f5 iControl library
netscaler	noredist		Enables the build of the netscaler plugin, requires additional libraries.
srx	noredist		Enables the build of the juniper srx plugin, requires additional libraries.
marvin	marvin.config	developer	Allows you to configure cloudstack using a marvin json configuration
Available properties		**Required profile**	**Description**
deploydb		developer	Clears and creates the cloud database in the mysql server configured utils/conf/db.properties
noredist			Enables all modules that are not part of the standard ASF build
systemvm		systemvm	Enabled the build of the systemvm.iso, requires mkisofs to be available on the commandline
awsapi		awsapi	Enables building of awsapi module along with rest of the CloudStack
simulator		developer	Enables the simulator spring context for running the integration test

Due to licensing issues, vhd-util was removed. Please download vhd-util: http://download.cloud.com.s3.amazonaws.com/tools/vhd-util

Copy vhd-util to this location in the source tree: scripts/vm/hypervisor/xenserver/vhd-util

To build the oss code(if you don't care about vmware/netscaler/netapp, use this command):

Make sure to install "python-setuptools", "mkisofs", "mysql-server" before running below command

```
mvn clean install -P developer,systemvm
```

To build noredist code:

```
$ mvn clean
$ mvn install -Dnoredist
```

Deploying Database

Note: The following is a WIP feature, use the old Ant command for a complete refresh. This command assumes you have a proper setup in utils/conf/db.properties and the cloud user already exists on your mysql database. If you have a root password, copy db.properties to db.properties.override and put your password there.

```
$ mvn -P developer -pl developer -Ddeploydb
```

To customize hosts, password etc. just copy utils/conf/db.properties to utils/conf/db.properties.override (this is git ignored by default) and edit the properties as needed they will override the defaults in utils/conf/db.properties.

Running a server for test/debug purposes.

One can use maven to deploy and debug the management server. First export the MAVEN_OPTS variable to open a transport socket on port 8787 (same as before).

```
$ export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=500m -Xdebug -Xrunjdwp:transport=dt_socket,address=8787,server=y,
suspend=n"
```

For most versions of java on most version of macosx you need to add '`-Djava.net.preferIPv4Stack=true`' to that so to be on the save side; on macosx

on macosx

```
$ export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=500m -Xdebug -Xrunjdwp:transport=dt_socket,address=8787,server=y,
suspend=n -Djava.net.preferIPv4Stack=true"
```

`code-Djava.net.preferIPv4Stack=truecode`

If you don't export using above configuration, the management server will run with no debugger attached.

On some (macosx) systems you might want to add "-Djava.net.preferIPv4Stack=true". Escpecially when using mixed wired and wireless networks.

This will run a jetty server on localhost port 8080 with the management server. Make sure that no other processes are using the port 8080. The default port for tomcat is 8080. You can either shut down tomcat service or switch to another port.

```
$ mvn -pl :cloud-client-ui jetty:run
```

To run a noredist management server:

```
$ mvn -pl :cloud-client-ui jetty:run -Dnoredist
```

NOTE : In case if BUILD process goes fine but, you are unable to access management server then please make sure that 8080 is opened in iptables.

To run awsapi app

```
$ mvn -Pawsapi -pl :cloud-awsapi jetty:run
```

Note: The following is a WIP tomcat-plugin feature, use the above jetty command for now.

```
$ mvn org.apache.tomcat.maven:tomcat7-maven-plugin:2.0:run -pl :cloud-client-ui -am -Pclient
```

Open the following URL on your browser to access the Management Server UI:
http://localhost:8080/client/
Or,
http://management-server-ip-address:8080/client

The default credentials are; user: admin, password: password and the domain field should be left blank which is defaulted to the ROOT domain.

Under development

This feature is still being developed and tested, might not work as expected

Integrated Simulator+Marvin test

The agent simulator and marvin are integrated into build steps to help a developer ensure that some simple tests pass before making a commit. The developer environment needs to have Marvin installed for the integration-test to work: These tests are lightweight and should ensure that your checkin doesn't break critical functionality for others working with branch: master.

To run the basic tests please refer to the Marvin wiki section on checkin tests.

Building Specific Versions of CloudStack

NOTE

Follow the following wiki page for building specific versions of CloudStack from source code.
Versions covered: 4.0 and above

Checkout Building where we document building process for specific version of CloudStack such as 4.0 oss/nonoss etc.

Packaging

Building RPM packages

Use package.sh to build packages for linux systems that use rpms, like RedHat, CentOS and fedora.

```
$ cd packaging/centos63
$ ./package.sh
```

Building DEB packages

On Master:

```
$ mvn install -P deps && dpkg-buildpackage
```

If you have compiled with additional flags (ex: -Dnoredist), you can get them packaged into the DEB by exporting those flags into the ACS_BUILD_OPTS prior to running dpkg-buildpackage.

Example:

mvn clean install -P deps -Dnoredist; export ACS_BUILD_OPTS="-Dnoredist"; dpkg-buildpackage

or

mvn install -P deps -Dnoredist -Dnonoss; export ACS_BUILD_OPTS="-Dnoredist -Dnonoss"; dpkg-buildpackage

WIP: Package using deb profile (feature under development uses jdeb plugin, in https://github.com/bhaisaab/incubator-cloudstack/tree/debs-maven debs-maven branch):

```
$ mvn package -P deb
```

Development with a IDE

Good Ol' Terminal

Following Building CloudStack section above.

IntelliJ IDEA

Import the project as Maven project. Build with maven, set socket type debugger on port 8787, follow Building CloudStack section above.

Eclipse

See Using Eclipse With CloudStack

Other Uses

RAT: ASF License Checking

```
$ mvn --projects='org.apache.cloudstack:cloudstack' org.apache.rat:apache-rat-plugin:0.10:check
```

The output file is in ${project.build.directory}/rat.txt, by default in target/rat.txt

Site Generation and Reporting

```
$ mvn site
```

Versioning / Releases

Update the version number of the project

```
mvn release:update-versions -DautoVersionSubmodules=true -Dnoredist -Pdeps,developer -
DdevelopmentVersion=<version>-SNAPSHOT
```

Releases can also be done using the release plugin. This plugin will automagically take care of setting the currect version (removing the -SNAPSHOT), making a tag in SCM and deploying the jars to the maven repository. This should only be executed by the release engineer. Maven assumes that once a version is released it will never change anymore and released binaries will never be updated, so any subsequent releases will have to bump the version number.

Skip unit test

```
mvn install -DskipTests=true
```

Troubleshooting

- Offline Mode:

Set commandline parameter -o to force maven to only use the local repository. This might trigger build failures if artifacts are not available locally, but might speed up the build process in some cases.

```
$ mvn -o
```

- Out of memory error, out of heap space:

```
// Bash
export MAVEN_OPTS=-Xmx512m
// Windows
set MAVEN_OPTS=-Xmx512m
```

Install the SystemVM Template

The install-sys-tmplt script will need to be run (before you launch the management server). The example below is for XenServer, if you are using KVM or VMware use "-h" option to specify a different Hypervisor. The choice of the systemVM template from the download site is also Hypervisor specific. This script will place the systemVM image the secondary storage. The systemVMs (e.g. SecondaryStorageVM) are cloned from this image and with the systemv.iso that is built from the image the SystemVM is patched. The install-sys-tmplt script is meant to run in a production environment, and will fail if you run it from a development environment. Here are the step to run it from a development environment:

- Create /etc/cloudstack/management/ directory
- Copy your db.properties (or db.properties.override if you have it) to this location
- Run install-sys-tmplt script (if you have a previous systemVM image use -F to force update). You will find the script in your build directory as shown below:

```
# mkdir -p /etc/cloudstack/management/
# cp ./client/target/cloud-client-ui-4.3.0-SNAPSHOT/WEB-INF/classes/db.properties /etc/cloudstack/management/
# ./client/target/generated-webapp/WEB-INF/classes/scripts/storage/secondary/cloud-install-sys-tmplt -m /mnt
/secondary -u http://jenkins.buildacloud.org/view/All/job/build-systemvm64-master/lastSuccessfulBuild/artifact
/tools/appliance/dist/systemvm64template-master-4.6.0-xen.vhd.bz2 -h xenserver -F
```

Remote debugging

To enable remote debugging on a system go to the cloudstack-management file (for centOS 7 this file can be found under the following path: /etc/default /cloudstack-management).|

Before CS 4.11 (tomcat7)

Add

-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5000

To the JAVA_OPTS:
JAVA_OPTS="-Djava.awt.headless=true -Dcom.sun.management.jmxremote=false -Xmx2g -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/var/log/cloudstack/management/ -XX:PermSize=512M -XX:MaxPermSize=800m -Djavax.net.ssl.trustStore=/etc/cloudstack/management/cloud.jks -Djavax.net.ssl.trustStorePassword=vmops.com **-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5000** "

Restart your management server (**systemctl restart cloudstack-management**).
You can now connect your java debugger on port 5000.

From CS 4.11 onwards (jetty):

Add

JAVA_TOOL_OPTIONS="-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5000"

as a new line.
Restart your management server (**systemctl restart cloudstack-management**).
You can now connect your java debugger on port 5000.