# The automated build system

**eugennc** **FastPathLogic, 2008**

The automated build system is intended to run the regression script(s) periodically, and to inform the team about its(their) status(es). This document explains its structure and some of the choices made, and provides a short tutorial that might be helpful if re-creating it.

Luntbuild is used to implement the automation process; it, according to a certain schedule, calls the regression scripts and wrappers, which in turn run as if they were launched manually ( i.e. are not aware of the context in which they are being run ). Some of the scripts are designed specifically to move and archive the reports provided by the regression scripts, after which they can be viewed and managed remotely. Also, luntbuild itself can be managed remotely, up to a certain degree.

Luntbuild is an open-source, java-based program that runs in either apache tomcat or can launch a built-in instance of jetty to host itself. Its URL is http://luntbuild.javaforge.com/. It is advised to host it on a specific, dedicated machine.

Proper partitioning will help increase the speed and decrease the likelihood of errors and failures during automated build system operation.

In this case, we had at our disposal a 500 GB HDD and 2 GiB of RAM. The first 10 GiB were allocated for the operating system ( / ) on an ext3 filesystem ( for maximum interoperability while providing journaling ); 450 GB were used to store the logs, reports and back-ups of luntbuild and CSLC ( /storage/back-up ) on an xfs filesystem ( for maximum general performance ); 34 GB were used to hold a ReiserFS partition ( /srv/luntbuild/work/CSLC ) – used to hold the CSLC files during svn checkout and build; last was a 5 GiB linux-swap partition.

The ReiserFS partition deserves special notice: because it is used to hold many small, rapidly changing files ( source files and binary objects mainly ), this filesystem was chosen; due to the fact that ReiserFS seems more prone to errors ( and also because this filesystem will have much more requests than any other fs during checkout, build and testing ) it only holds a local copy of the SVN tree and temporary objects – no critical information. Also, in order to make it available for both run_regress and dist_regress to work on, without changing the scripts, it is mounted both in /srv/luntbuild/work/CSLC and /storage/users/dist_regress/repo1/cslc/.

Before installing Luntbuild, care should be taken that its prerequisites ( chiefly Java ) are being properly installed and up-to-date. Sometimes, outdated Java installations can cause problems, from luntbuild not running at all to, disconcertingly, sprouting specific or random errors. Usually, luntbuild's online manual provides good information, if sometimes outdated. This instance of luntbuild was installed using manual install ( in /srv/luntbuild ), and deployed using its own jetty server. Due to similar running prerequisites ( networking and lack of external config files ), a modified NTPD init script was used to schedule boot-time luntbuild start-up:

```
#!/bin/bash
#
# Source function library
. /etc/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

if [ -f /etc/sysconfig/luntbuild ];then
        . /etc/sysconfig/luntbuild
fi

RETVAL=0
prog="/srv/luntbuild/bin/luntbuild.sh"


start() {
        # Check that networking is up.
        [ "$NETWORKING" = "no" ] && exit 1

        # Start daemons.
        echo -n $"Starting $prog: "
        daemon /srv/luntbuild/bin/luntbuild.sh &
        RETVAL=$?
        touch /var/run/luntbuildjava.pid
          ps x -o pid,cmd:100 | grep -e 'java.*luntbuild' | grep -v grep | awk '{print $1}' >
/var/run/luntbuildjava.pid
        echo
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/luntbuild
        return $RETVAL
}

stop() {
        echo -n $"Shutting down $prog: "
        killall -9 luntbuild.sh
        kill -9 `cat /var/run/luntbuildjava.pid`
        rm /var/run/luntbuildjava.pid -f
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/luntbuild
        return $RETVAL
}
# See how we were called.
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  status)
        status luntbuild
        RETVAL=$?
        ;;
  restart|reload)
        stop
        start
        RETVAL=$?
        ;;
  condrestart)
```

```
        if [ -f /var/lock/subsys/luntbuild ]; then
            stop
            start
            RETVAL=$?
        fi
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart|condrestart|status}"
        RETVAL=3
esac

exit   $RETVAL
```

To manually start luntbuild, simply running /srv/luntbuild/bin/luntbuild.sh would do. This script can take hostname and port information e.g. /srv/luntbuild/bin/luntbuild.sh localhost 4000.

The stop-luntbuild.sh script shuts down luntbuild ( if host or port were customized during start-up, same information should be provided here e.g. /srv/luntbuild/bin/stop-luntbuild.sh localhost 4000 ). To stop luntbuild, simply accessing the localhost with a web browser at a specific port will suffice. The port is usually the start-up port +1 ( since default port is 4000, accessing port 4001 on the same machine would shut down luntbuild ). Therefore, the firewall should be configured to enforce the desired policy in this aspect ( e.g. blocking the port if remote web-based shut-down is undesired ).

To access luntbuild, and to check if it is running, the address http://localhost:4000/luntbuild would be used in the default case. Please note, that, if running luntbuild stand-alone, http://localhost:4000 would point to a non-existing page, making jetty point to a default error page.

After accessing the page, a redirect would occur ( to the ./app.do page ). By default, this presents lunbuild's state read-only. Clicking the log-out button will then allow logging in as the administrator ( using the username luntbuild and the password chosen during install ).

On the administration tab, the choices to export and import projects and whole luntbuild configurations ( including any projects ) is presented.

The properties tab includes a few interesting configuration: Url to access luntbuild servlet, when configured properly, can be used to access luntbuild remotely from www, even if it is running on a local machine and only port 4000 is routed publicly.

Database backup file : is set, for CSLC, to /storage/back-up/database.xml ( the xfs partition ), and

Database backup cron expression : is set to 0 0 22 * * ? ( 2200 hours of every day ).

Here appropriate notification engines can be provided authentication information ( e.g. E-mail notification by providing a server name, port, luntbuild username and password for the e-mail account etc ).

The users tab manages users in a very straight-forward way. Apart from providing access permissions, users can be used to specify notification groups.

The build tab provides an easy way to visualize builds, schedule and run statuses, to manage them ( enable schedule, disable schedule, trigger manually ) and check a list of all builds of a particular schedule.

The projects tab allows for creation, administration and deletion of projects, and, for the configuration of a project, is most important. Its sub-tabs of interest are:

VCS adaptors: this allows setting a versioning system to use, from simple local file-tree to CVS, SVN and beyond.

The needed information is:

Subversion

| | |
|---|---|
| Repository url base | svn://build/TOT/fpl/cslc |
| Repository layout | multiple |
| Directory for trunk | |
| Directory for branches | |
| Directory for tags | |
| Username | <USERNAME> |
| Password | ****** |
| Web interface | |
| URL to web interface | |
| Quiet period | |

Builders: here the greatest quantity on information regarding the actual way to run the builds is collected.

For example, to build the Debug binary, the builder would look like this:

Debug

| | |
|---|---|
| Builder type | Ant builder |
| Command to run Ant | /opt/tools/ant/bin/ant |
| Build script path | /srv/luntbuild/work/CSLC/trunk/src/build.xml |
| Build targets | debug |
| Build properties | buildVersion="${build.version}"<br>artifactsDir="${build.artifactsDir}"<br>buildDate="${build.startDate}"<br>junitHtmlReportDir="${build.getReportUrl("JUnit")}" |
| Environment variables | JAVA_HOME=/usr/java/jdk1.5.0_12 |
| Build success condition | result==0 and builderLogContainsLine("BUILD SUCCESSFUL") |

After checking out the SVN tree locally and building the desired binary, running a series of tests will provide information about the status of the software and signal possible regressions. The two perl scripts that test CSLC for regressions are run_regress.pl and ssh_dist_regress.pl. The first is local, the second distributed.

To check the Golden Suite for regressions locally ( and also perform valgrind-assisted memory-checking ) a builder might look like:

Valgrind CSL Regression

| | |
|---|---|
| Builder type | Command builder |
| Build command | /srv/luntbuild/work/CSLC/misc/scripts/run_regress_valgrind.pl 2>/dev/null |
| Run command in directory | /srv/luntbuild/work/CSLC/trunk/scripts/ |
| Wait for process to finish before continuing? | Yes |
| Environment variables | PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/srv/luntbuild/work/CSLC/trunk/scripts<br>REGRESSION_PATH=/srv/luntbuild/work/CSLC/trunk/scripts/<br>WORK=/srv/luntbuild/work/CSLC<br>JAVA_HOME=/usr/java/jdk1.5.0_12 |
| Build success condition | builderLogContainsLine("REGRESSION DONE") |

In order to run the Golden Regression test with dist_regress ( for this example, without valgrind ), a few steps need to be taken. Those steps are necessary to ensure that running dist_regress on the same files as normal regression causes no problems. First, we need to change the ownership of the files ( run_regress.pl runs as root user, while ssh_dist_regress.pl runs as dist_regress ):

chown dist_regress

| | |
|---|---|
| Builder type | Command builder |
| Build command | chown dist_regress /storage/users/dist_regress/repo1/cslc -R |
| Run command in directory | /storage/users/dist_regress/repo1/cslc |
| Wait for process to finish before continuing? | Yes |
| Environment variables | |
| Build success condition | |

The dist_regress builder looks like this:

| CSL_DIST_REGRESS | |
|---|---|
| Builder type | Command builder |
| Build command | su -c "cd /storage/users/dist_regress/repo1/cslc/trunk/scripts; ./run_dist_regress_silent.sh -hdl csl" - dist_regress |
| Run command in directory | /srv/luntbuild/work/CSLC/trunk/scripts |
| Wait for process to finish before continuing? | Yes |
| Environment variables | PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/srv/luntbuild/work/CSLC/trunk/scripts:/opt/tools/ant/bin<br>REGRESSION_PATH=/srv/luntbuild/work/CSLC/trunk/scripts<br>WORK=/srv/luntbuild/work/CSLC<br>JAVA_HOME=/usr/java/jdk1.5.0_12 |
| Build success condition | builderLogContainsLine("Overall regression status = passed") |

Please note the way dist_regress is being run: using `su` ( because luntbuild executes builders through java, there is no actual shell, and `sudo` cannot be employed ), using the flag -c ( signaling it to wholly execute the next argument ) and `- dist_regress` ( there is a whitespace there – to run it as the dist_regress user ). Since `su` is launched by luntbuild which runs, daemonised, as root, no password will be required. Also note the addition of `/opt/tools/ant/bin` to `$PATH`, needed because dist_regress will use ( both SVN and ) ant manually ( that is, will not rely on luntbuild ).

Afterwards, changing ownership back to root is made through a builder similar to that presented earlier; in order to maintain context, its paths are changed to `/srv/luntbuild/work/CSLC`; this is not very important, since both paths point to the same partition.

The builders are structured into schedules; a schedule, apart from calling builders and being launched at a specific time, also syncs the local file tree with the repository and provides notifications. For example, this is how the hourly build would look like:

Hourly_Regression

| | |
|---|---|
| Execution status | success at 2008-09-29 17:11 |
| Description | |
| Next build version | hourly 1173 |
| Variables | |
| Work subdirectory | |
| Trigger type: | cron |
| Cron expression: | 0 0 7,9,11,13,15,17,19,21 * * ? |
| Build necessary condition | always |
| Associated builders | Debug , CSL_NO_POST_COMPILE Regression , MoveOutput , Clean-up |
| Associated post-builders | moveOutputWhenFailed , Clean-up |
| Build type | increment |
| Post-build strategy | post-build if failed |
| Label strategy | do not label |
| Notify strategy | notify always |
| Schedules current schedule depends on | |
| Dependency triggering strategy | trigger schedules this schedule depends on |
| Build cleanup strategy | do not cleanup builds automatically |
| Latest build | hourly 1172 success at 2008-09-29 17:11 |

The cron expression ( if in doubt: man 5 crontab ) needs a bit of care – do no set both the day of the month and the day of the week to *, but rather set one to * and the other to ? . * means all while ? means any ( not in the logical, but rather in the standard linguistic sense; * will *make* the cron expression to execute during those days, while ? will *let* cron expression execute on those days if something else makes it want to execute then ).

The course of the build is the following: when the cron expression is evaluated to true ( luntbuild will not let two builds of the same project run simultaneously, and after a build has ended, only the latest in the queue will execute, by default ), the scheduled job will start; if Build type is clean, it will delete the projects local file tree before syncing, otherwise it will update the local copy. After check-out, it will begin executing jobs. Here, first is the debug build. Afterwards it will run the Golden Regression test suite, and then it will archive the reports and perform various needed clean-up actions. Since the post-build strategy is post-build if failed, it will run the post-builders if the standard build interrupted with an error ( i.e. the success conditions specified in each builder were not met – it will not post-build for a non-interrupting error ). In this case, the slightly modified moveOutput script will label the report as failed.

The moveOutput script is as follows:

```
#!/bin/bash

cp /srv/luntbuild/work/CSLC/trunk/scripts/regress_html_templates/style.css
/srv/luntbuild/work/CSLC/test/report/        #copy .css template for archival purposes
cp /srv/luntbuild/work/CSLC/trunk/scripts/regress_html_templates/style.css /storage/back-
up/logs/trunk/scripts/regress_html_templates/ #copy .css template for html viewing

cp /srv/luntbuild/work/CSLC/test/report /storage/back-up/logs/CSLC/tempdir -r
#copy report in a temp dir
find /storage/back-up/logs/CSLC/tempdir -type f -exec chmod 644 {} \; && find /storage/back-
up/logs/CSLC/tempdir -type d -exec chmod 755 {} \;     #set permissions for web access

if [ "$1" == "nightly" ]
then
    archiveName="/storage/back-up/logs/CSLC/report__`date`$2.tar.gz"
    /bin/tar -pczf "$archiveName"  "/storage/back-up/logs/CSLC/today" #archive daily reports
    chmod 644 "$archiveName"
    rm -Rf /storage/back-up/logs/CSLC/today/*
fi

mv /storage/back-up/logs/CSLC/tempdir/* /storage/back-up/logs/CSLC/today/

lastRun=`cat /storage/back-up/logs/CSLC/today/last_run`       #latest report name extraction
if [ $? -eq "0" ]                                     #only if cat succeeded ( in finding file
et caetera )
then
  name=`basename "$lastRun"`
  echo "From last: $lastRun has been extracted the name $name"
  rm -f /storage/back-up/logs/CSLC/today/latest
  ln -s -f "/storage/back-up/logs/CSLC/today/$name" "/storage/back-up/logs/CSLC/today/latest"
#create link
fi

rm -Rf /storage/back-up/logs/CSLC/tempdir
rm -Rf /srv/luntbuild/work/CSLC/test/report
#find /storage/back-up/logs/CSLC -type f -exec chmod 644 {} \; && find /storage/back-up/logs/CSLC
-type d -exec chmod 755 {} \;     #set permissions for web access
exit 0
```

The script, if called with the nightly as the first arg, will archive the reports collected in the publishing dir and delete them, leaving in place only the new report; thus, reports from a day ( nightly reports and hourly reports of the following day ) can be archived in a single file. From there, a https enabled apache server can publish the reports internally, requiring username/passwd authentication.

In order to send appropriate notifications, some luntbuild templates must be changed. For e-mail notification, the files are:
/srv/luntbuild/templates/email/set-template.txt:

buildTemplate=simple-build.vm
scheduleTemplate=simple-schedule.vm

This sets the aliases of templates to actual files. The files are XHTML code; they take specific luntbuild variables ( e.g. ${build_isFailure} to set the color and message ), and can be freely modified to, for example, provide customized links to the above-mentioned https report archive.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
<title>[luntbuild] build of "${build_project}/${build_schedule}/${build_version}"
${build_status}</title>
</head>

<body>
#set ($GREEN = "#00AA00")
#set ($RED = "#AA0000")
#set ($OTHER = "#AAAA00")

#if ($build_isSuccess)
  #set ($color=$GREEN)
#elseif ()
  #set ($color=$RED)
#else
  #set ($color=$OTHER)
#end

<span style="color:${color}">
Build of ${build_project}/${build_schedule}/${build_version} finished with status:
${build_status}.
<p>
This build has started at ${build_start}, and has finished at ${build_end}.
</p>
<p>
${build_user_msg}
</p>
<p>
See the build log and the revision log for details.
</p>
<big><b>Build Artifacts:</b></big></br>
</hr>
<table summary="" border="2" cellpadding="10" cellspacing="0">
  <tbody>
    <tr>
                <td>Build Log</td>
                <td><a href="${build_buildlog_url}">${build_buildlog_url}</a></td>
        </tr>
    <tr>
                <td>Revision Log</td>
```

```
                <td><a href="${build_revisionlog_url}">${build_revisionlog_url}</a></td>
        </tr>
    <tr>
                <td>Login Page</td>
                <td><a href="${luntbuild_servlet_url}">${luntbuild_servlet_url}</a></td>
        </tr>
   <tr>                 <td>Latest Build</td>
                        <td><a  href="http://fpl64-14/regression/CSLC/today/latest/">http://fpl64-
14/regression/CSLC/today/latest/</a></td>
       </tr>
  <tr>
                <td>Today's Builds</td>
                              <td><a  href="http://fpl64-14/regression/CSLC/today/">http://fpl64-
14/regression/CSLC/today/</a></td>
      </tr>
  </tbody>
</table>

-- luntbuild
</span>
</body>
</html>
```

The information presented up to this point should be enough to help in the setting up or basic maintenance of a luntbuild-based automated build system. Some finer ( yet being able to pose problems ) points are explained in the Caveat section below.

Addendum

Luntbuild can be used in other ways; for example to send e-mail reminders. A simple schedule

| Send weekly report reminder | |
|---|---|
| Execution status | success at 2008-09-26 12:00 |
| Description | |
| Next build version | report 13 |
| Variables | |
| Work subdirectory | |
| Trigger type: | cron |
| Cron expression: | 0 0 12 ? * FRI |
| Build necessary condition | always |
| Associated builders | Weekly Report Reminder |
| Associated post-builders | |
| Build type | increment |
| Post-build strategy | do not post-build |
| Label strategy | do not label |
| Notify strategy | do not notify |
| Schedules current schedule depends on | |
| Dependency triggering strategy | trigger schedules this schedule depends on |
| Build cleanup strategy | do not cleanup builds automatically |
| Latest build | report 12 success at 2008-09-26 12:00 |

can use a simple builder:

| Weekly Report Reminder | |
|---|---|
| Builder type | Command builder |
| Build command | /storage/back-up/send-mail.sh "Please remember to send out your weekly report." "Weekly Report Reminder" "company@fplsrl.com" |
| Run command in directory | /storage/back-up |
| Wait for process to finish before continuing? | Yes |
| Environment variables | |
| Build success condition | |

to run a simple script:

#!/bin/bash

```
# script to send simple email
# email subject
SUBJECT="Weekly Report Reminder"
# Email To ?
EMAIL="company@fplsrl.com"
# Email text/message
EMAILMESSAGE="Please remember to send out your weekly report."

if [ "$#" -ge "1" ]
then
  EMAILMESSAGE=$1
  if [ "$#" -ge "2" ]
  then
    SUBJECT=$2
    if [ "$#" -ge "3" ]
    then
    EMAIL=$3
    fi
  fi
fi
# send an email using /bin/mail
#/bin/mail -s "$SUBJECT" "$EMAIL" < $EMAILMESSAGE
echo
echo $EMAILMESSAGE | /bin/mail -s "$SUBJECT" "$EMAIL"
```

that will remind everybody about the weekly reports ( or can be easily customized through arguments to deliver any message or subject, to a certain e-mail address). Thus, it can be run without arguments, and provide the persistent, hard-coded information, or can be remotely managed through luntbuild.

Caveat I:

When running jobs that produce large amounts of output, luntbuild might crash.

This is usually an overflow error in a HTML generation java class. To avoid this, a straightforward measure might be to reduce the size of the output. Thus, redirecting output to logs or to /dev/null ( if appropriate ) will make luntbuild work. A script like:

```
#!/bin/bash
exec 2>/dev/null
nice -n 5 ./run_regress.pl $* 2> /dev/null
echo ""
```

will remove the stderr output, both from the script.run itself and from other sources ( e.g. ls reporting non-existing nodes ). This is suitable for critical regression scripts, but when running larger jobs, all but the Golden suite are allowed to fail. Also, when running tens of scripts with tens of thousands tests, even standard output might cause luntbuild to crash. Thus, ( again, for all but the golden regression test suite ) a slightly changed script like this might help:

```
#!/bin/bash
exec 2>/dev/null
nice -n 5 ./run_regress.pl $* > /dev/null 2> /dev/null
echo "REGRESSION DONE"
echo ""
```

Here, both stderr and stdout are supressed, and the only two lines the script will output are the standard success message and a void line ( to provide better separation of builds when looking over or parsing the luntbuild log ).

Caveat II:

As seen in the scripts above, running the regression with higher niceness than default might help: since the regression can be very resource-intensive, giving it a slightly reduced priority ( CPU-time-wise ) will allow luntbuild to run smoothly ( or acceptably, depending on the system load ) even when running scheduled jobs, and provide up-time for developers checking its status.

Caveat III:

CSLC uses RLM; when a license is unavailable, for internal, debug purposes, RLM might be turned off ( by simply issuing the target/flag rlm_off when building, for safety before the actual target: ant rlm_off debug ). The Debug builder is thus modified ( Build targets from **debug** to **rlm_off debug** ):

Debug

| | |
|---|---|
| Builder type | Ant builder |
| Command to run Ant | /opt/tools/ant/bin/ant |
| Build script path | /srv/luntbuild/work/CSLC/trunk/src/build.xml |
| Build targets | rlm_off debug |
| Build properties | buildVersion="${build.version}"<br>artifactsDir="${build.artifactsDir}"<br>buildDate="${build.startDate}"<br>junitHtmlReportDir="${build.getReportUrl("JUnit")}" |
| Environment variables | JAVA_HOME=/usr/java/jdk1.5.0_12 |
| Build success condition | result==0 and builderLogContainsLine("BUILD SUCCESSFUL") |