Central Person Registry Build Documentation

The Central Person Registry is composed of the following components:

- Servlet container Jetty.
- Instance of PostgreSQL.
- ActiveMQ.
- Directory Provisioner Application (Only needs to be ran on a single node).
- CPR Core and UI Code.

Assumptions

The configuration information that follows assumes that the home for the CPR is located in **/opt/dbstore/cpr**. The directory structure under the CPR is as follows:

- **apps** - contains CPR developed applications or applications that cannot be installed via yum (activemq, ant, and jetty).
- **db** - contains the PostgreSQL database storage. NOTE: to work with the yum installed version of PostgreSQL, this directory is symlinked to **/var/lib/pgsql9/data**.
    - Important files in that directory are, pg_hba.conf and postgresql.conf (more information on them later).
- **files** - contains important CPR related files and properties.
    - **files/jars** - contains all of the non-packaged up jars that are used by the CPR. At this time, dependencies do not exist for them in the Maven repos, so that is why locate versions of them are used.
    - **files/properties** - contains all of the important CPR properties files. For purposes of CommIT, the only two that are important are the **cpr.properties** and **hibernate.cfg.xml** files.
- **scripts** - not used for this version of the CPR.
- **src** - contains the CPR source code, pulled from Git.
- **/var/log/cpr/core** - contains the logging for core (all directories are owned by **jetty:jetty**).
- **/var/log/cpr/dataflux** - contains the logging for dataflux (not used to OS CPR).
- **/var/log/cpr/services** - contains the service logging.
- **/var/log/cpr/sp** - contains the service provisioning logging.
- **/var/log/cpr/ui** - contains the ui logging.
- **/var/log/cpr/directory** - contains the directory provisioner logging.

Java 1.7.x must be installed and set up as the default JDK. You do that using the "alternatives --config" command.

In the documentation that follows, all references to the environment **$CPR_HOME** is the home location where the CPR software has been installed. On the existing VMs, that location is **/opt/dbstore/cpr**.

JVM tunables for directory provisioner, and activemq should be: **-Xms1G -Xmx1G -XX:+UseG1GC.**

CPR Ports

Port	Usage
80	HTTP
443	HTTPS
5432	PostgreSQL Listener Port
8080	HTTP
8162	AMQ Console SSL
8443	HTTPS
61612	STOMP + SSL
61617	JMS SSL

Starting Point

On **ec2-54-244-223-148.us-west-2.compute.amazonaws.com** in **/opt/dbstore** directory, there is a file called **cpr_dist.tar.gz**, copy that file to the new server. Create a directory that will contain the CPR and extract that file. It contains ActiveMQ, and all of the startup files necessary for the CPR.

Jetty

The CPR is using the standard Jetty install (jetty-distribution-9.0.3.v20130506). It is symlinked to /usr/share/jetty. From a configuration standpoint, the following items were changed:

- Set up a local user and group called, **jetty**.
- Updated **start.ini**, refer to start.ini on either one of the configured nodes for options/configurations used.

```
## JVM Parameters.
--exec
-Xmx2000m
-Xmn512m
-XX:+UseConcMarkSweepGC
-XX:ParallelCMSThreads=2
-XX:+CMSClassUnloadingEnabled
-XX:+UseCMSCompactAtFullCollection
-XX:CMSInitiatingOccupancyFraction=80


## Thread parameters.
OPTIONS=Server,websocket,resources,ext
threads.min=10
threads.max=200
threads.timeout=60000
#jetty.host=myhost.com
jetty.dump.start=false
jetty.dump.stop=false

etc/jetty.xml


## Request Log.
requestlog.retain=30
requestlog.append=true
requestlog.extended=true
etc/jetty-requestlog.xml


## Error Log.
jetty.log.retain=30
etc/jetty-logging.xml


## SetUID start up options.
OPTIONS=setuid
jetty.startServerAsPrivileged=false
jetty.username=jetty
jetty.groupname=jetty
jetty.umask=002
etc/jetty-setuid.xml


## SSL options.
etc/jetty-ssl.xml


## HTTPS options.
jetty.https.port=8443
etc/jetty-https.xml
```

- In the **etc** directory, symlinked the **commit.jks** file to **keystore**, which is referred to in the configuration file.
- Copied all of the non-packaged jars that the CPR uses to the **lib/ext** directory.
- In the **resources** directory, symlinked in the cpr.properties, eduPersonRules.drl, group.client.properties, idcardrules.drl, hibernate.cfg.xml and rules.drl files.
- You need to install jetty so that it starts up. Please the following into a file called, **/etc/init.d/jetty**. Then make it executable, **chmod +x /etc/init.d/jetty**. And finally add it to the system startup, **chkconfig --add jetty**. **NOTE: This script assumes that you have installed or symlinked Jetty into /usr/share/jetty**.

```
#!/bin/bash
#
# jetty
# chkconfig: 2345 99 99
# description: Jetty 9 webserver
#
```

```
# Configuration files
#
# /etc/default/jetty
#   If it exists, this is read at the start of script. It may perform any
#   sequence of shell commands, like setting relevant environment variables.
#
# $HOME/.jettyrc
#   If it exists, this is read at the start of script. It may perform any
#   sequence of shell commands, like setting relevant environment variables.
#
# /etc/jetty.conf
#   If found, and no configurations were given on the command line,
#   the file will be used as this script's configuration.
#   Each line in the file may contain:
#     - A comment denoted by the pound (#) sign as first non-blank character.
#     - The path to a regular file, which will be passed to jetty as a
#       config.xml file.
#     - The path to a directory. Each *.xml file in the directory will be
#       passed to jetty as a config.xml file.
#
#   The files will be checked for existence before being passed to jetty.
#
# $JETTY_HOME/etc/jetty.xml
#   If found, used as this script's configuration file, but only if
#   /etc/jetty.conf was not present. See above.
#
# Configuration variables
#
# JAVA
#   Command to invoke Java. If not set, java (from the PATH) will be used.
#
# JAVA_OPTIONS
#   Extra options to pass to the JVM
#
# JETTY_HOME
#   Where Jetty is installed. If not set, the script will try go
#   guess it by first looking at the invocation path for the script,
#   and then by looking in standard locations as $HOME/opt/jetty
#   and /opt/jetty. The java system property "jetty.home" will be
#   set to this value for use by configure.xml files, f.e.:
#
#    <Arg><Property name="jetty.home" default="."/>/webapps/jetty.war</Arg>
#
# JETTY_PORT (Deprecated - use JETTY_ARGS)
#   Override the default port for Jetty servers. If not set then the
#   default value in the xml configuration file will be used. The java
#   system property "jetty.port" will be set to this value for use in
#   configure.xml files. For example, the following idiom is widely
#   used in the demo config files to respect this property in Listener
#   configuration elements:
#
#    <Set name="Port"><Property name="jetty.port" default="8080"/></Set>
#
#   Note: that the config file could ignore this property simply by saying:
#
#    <Set name="Port">8080</Set>
#
# JETTY_RUN
#   Where the jetty.pid file should be stored. It defaults to the
#   first available of /var/run, /usr/var/run, JETTY_HOME and /tmp
#   if not set.
#
# JETTY_PID
#   The Jetty PID file, defaults to $JETTY_RUN/jetty.pid
#
# JETTY_ARGS
#   The default arguments to pass to jetty.
#   For example
#      JETTY_ARGS=jetty.port=8080 jetty.spdy.port=8443 jetty.secure.port=443
#
# JETTY_USER
```

```
#   if set, then used as a username to run the server as
#

usage()
{
    echo "Usage: ${0##*/} [-d] {start|stop|run|restart|check|supervise} [ CONFIGS ... ] "
    exit 1
}

[ $# -gt 0 ] || usage


##################################################
# Some utility functions
##################################################
findDirectory()
{
  local L OP=$1
  shift
  for L in "$@"; do
    [ "$OP" "$L" ] || continue
    printf %s "$L"
    break
  done
}

running()
{
  local PID=$(cat "$1" 2>/dev/null) || return 1
  kill -0 "$PID" 2>/dev/null
}

started()
{
  # wait for 60s to see "STARTED" in PID file, needs jetty-started.xml as argument
  for T in 1 2 3 4 5 6 7 9 10 11 12 13 14 15
  do
    sleep 4
    [ -z "$(grep STARTED $1 2>/dev/null)" ] || return 0
    [ -z "$(grep STOPPED $1 2>/dev/null)" ] || return 1
    [ -z "$(grep FAILED $1 2>/dev/null)" ] || return 1
    local PID=$(cat "$2" 2>/dev/null) || return 1
    kill -0 "$PID" 2>/dev/null || return 1
    echo -n ". "
  done

  return 1;
}


readConfig()
{
  (( DEBUG )) && echo "Reading $1.."
  source "$1"
}



##################################################
# Get the action & configs
##################################################
CONFIGS=()
NO_START=0
DEBUG=0

while [[ $1 = -* ]]; do
  case $1 in
    -d) DEBUG=1 ;;
  esac
  shift
done
```

```
ACTION=$1
shift

###########################################################
# Read any configuration files
###########################################################
ETC=/etc
if [ $UID != 0 ]
then
  ETC=$HOME/etc
fi

for CONFIG in $ETC/default/jetty{,9} $HOME/.jettyrc; do
  if [ -f "$CONFIG" ] ; then
    readConfig "$CONFIG"
  fi
done


###########################################################
# Set tmp if not already set.
###########################################################
TMPDIR=${TMPDIR:-/tmp}

###########################################################
# Jetty's hallmark
###########################################################
JETTY_INSTALL_TRACE_FILE="etc/jetty.xml"


###########################################################
# Try to determine JETTY_HOME if not set
###########################################################
JETTY_HOME="/usr/share/jetty"
if [ -z "$JETTY_HOME" ]
then
  JETTY_SH=$0
  case "$JETTY_SH" in
    /*)   ;;
    ./*)  ;;
    *)    JETTY_SH=./$JETTY_SH ;;
  esac
  JETTY_HOME=${JETTY_SH%/*/*}

  if [ ! -f "${JETTY_SH%/*/*}/$JETTY_INSTALL_TRACE_FILE" ]
  then
    JETTY_HOME=
  fi
fi


###########################################################
# if no JETTY_HOME, search likely locations.
###########################################################
if [ -z "$JETTY_HOME" ] ; then
  STANDARD_LOCATIONS=(
        "/usr/share"
        "/usr/share/java"
        "${HOME}"
        "${HOME}/src"
        "${HOME}/opt"
        "/opt"
        "/java"
        "/usr/local"
        "/usr/local/share"
        "/usr/local/share/java"
        "/home"
        )
  JETTY_DIR_NAMES=(
        "jetty-9"
        "jetty9"
```

```
        "jetty-9.*"
        "jetty"
        "Jetty-9"
        "Jetty9"
        "Jetty-9.*"
        "Jetty"
        )

  for L in "${STANDARD_LOCATIONS[@]}"
  do
    for N in "${JETTY_DIR_NAMES[@]}"
    do
      POSSIBLE_JETTY_HOME=("$L/"$N)
      if [ ! -d "$POSSIBLE_JETTY_HOME" ]
      then
        # Not a directory. skip.
        unset POSSIBLE_JETTY_HOME
      elif [ ! -f "$POSSIBLE_JETTY_HOME/$JETTY_INSTALL_TRACE_FILE" ]
      then
        # Trace file not found. skip.
        unset POSSIBLE_JETTY_HOME
      else
        # Good hit, Use it
        JETTY_HOME=$POSSIBLE_JETTY_HOME
        # Break out of JETTY_DIR_NAMES loop
        break
      fi
    done
    if [ -n "$POSSIBLE_JETTY_HOME" ]
    then
      # We have found our JETTY_HOME
      # Break out of STANDARD_LOCATIONS loop
      break
    fi
  done
fi


####################################################
# No JETTY_HOME yet? We're out of luck!
####################################################
if [ -z "$JETTY_HOME" ]; then
  echo "** ERROR: JETTY_HOME not set, you need to set it or install in a standard location"
  exit 1
fi

cd "$JETTY_HOME"
JETTY_HOME=$PWD


####################################################
# Check that jetty is where we think it is
####################################################
if [ ! -r "$JETTY_HOME/$JETTY_INSTALL_TRACE_FILE" ]
then
  echo "** ERROR: Oops! Jetty doesn't appear to be installed in $JETTY_HOME"
  echo "** ERROR:  $JETTY_HOME/$JETTY_INSTALL_TRACE_FILE is not readable!"
  exit 1
fi

####################################################
# Try to find this script's configuration file,
# but only if no configurations were given on the
# command line.
####################################################
if [ -z "$JETTY_CONF" ]
then
  if [ -f $ETC/jetty.conf ]
  then
    JETTY_CONF=$ETC/jetty.conf
  elif [ -f "$JETTY_HOME/etc/jetty.conf" ]
```

```
  then
    JETTY_CONF=$JETTY_HOME/etc/jetty.conf
  fi
fi

##################################################
# Get the list of config.xml files from jetty.conf
##################################################
if [ -z "$CONFIGS" ] && [ -f "$JETTY_CONF" ] && [ -r "$JETTY_CONF" ]
then
  while read -r CONF
  do
    if expr "$CONF" : '#' >/dev/null ; then
      continue
    fi

    if [ -d "$CONF" ]
    then
      # assume it's a directory with configure.xml files
      # for example: /etc/jetty.d/
      # sort the files before adding them to the list of CONFIGS
      for XMLFILE in "$CONF/"*.xml
      do
        if [ -r "$XMLFILE" ] && [ -f "$XMLFILE" ]
        then
          CONFIGS+=("$XMLFILE")
        else
          echo "** WARNING: Cannot read '$XMLFILE' specified in '$JETTY_CONF'"
        fi
      done
    else
      # assume it's a command line parameter (let start.jar deal with its validity)
      CONFIGS+=("$CONF")
    fi
  done < "$JETTY_CONF"
fi

##################################################
# Find a location for the pid file
##################################################
if [ -z "$JETTY_RUN" ]
then
  JETTY_RUN=$(findDirectory -w /var/run /usr/var/run $JETTY_HOME /tmp)
fi

##################################################
# Find a pid and state file
##################################################
if [ -z "$JETTY_PID" ]
then
  JETTY_PID="$JETTY_RUN/jetty.pid"
fi

if [ -z "$JETTY_STATE" ]
then
  JETTY_STATE=$JETTY_HOME/jetty.state
fi
JAVA_OPTIONS+=("-Djetty.state=$JETTY_STATE")
rm -f $JETTY_STATE

##################################################
# Setup JAVA if unset
##################################################
if [ -z "$JAVA" ]
then
  JAVA=$(which java)
fi

if [ -z "$JAVA" ]
then
  echo "Cannot find a Java JDK. Please set either set JAVA or put java (>=1.5) in your PATH." 2>&2
```

```
  exit 1
fi

#####################################################
# See if JETTY_PORT is defined
#####################################################
if [ "$JETTY_PORT" ]
then
  JAVA_OPTIONS+=("-Djetty.port=$JETTY_PORT")
fi

#####################################################
# See if JETTY_LOGS is defined
#####################################################
if [ "$JETTY_LOGS" ]
then
  JAVA_OPTIONS+=("-Djetty.logs=$JETTY_LOGS")
fi

#####################################################
# Are we running on Windows? Could be, with Cygwin/NT.
#####################################################
case "`uname`" in
CYGWIN*) PATH_SEPARATOR=";";;
*) PATH_SEPARATOR=":";;
esac


#####################################################
# Add jetty properties to Java VM options.
#####################################################
JAVA_OPTIONS+=("-Djetty.home=$JETTY_HOME" "-Djava.io.tmpdir=$TMPDIR")

[ -f "$JETTY_HOME/etc/start.config" ] && JAVA_OPTIONS=("-DSTART=$JETTY_HOME/etc/start.config" "${JAVA_OPTIONS
[@]}")

#####################################################
# This is how the Jetty server will be started
#####################################################

JETTY_START=$JETTY_HOME/start.jar
[ ! -f "$JETTY_START" ] && JETTY_START=$JETTY_HOME/lib/start.jar

START_INI=$(dirname $JETTY_START)/start.ini
[ -r "$START_INI" ] || START_INI=""

RUN_ARGS=(${JAVA_OPTIONS[@]} -jar "$JETTY_START" $JETTY_ARGS "${CONFIGS[@]}")
RUN_CMD=("$JAVA" ${RUN_ARGS[@]})

#####################################################
# Comment these out after you're happy with what
# the script is doing.
#####################################################
if (( DEBUG ))
then
  echo "JETTY_HOME     =  $JETTY_HOME"
  echo "JETTY_CONF     =  $JETTY_CONF"
  echo "JETTY_PID      =  $JETTY_PID"
  echo "JETTY_START    =  $JETTY_START"
  echo "JETTY_ARGS     =  $JETTY_ARGS"
  echo "CONFIGS        =  ${CONFIGS[*]}"
  echo "JAVA_OPTIONS   =  ${JAVA_OPTIONS[*]}"
  echo "JAVA           =  $JAVA"
  echo "RUN_CMD        =  ${RUN_CMD}"
fi

#####################################################
# Do the action
#####################################################
case "$ACTION" in
  start)
```

```
  echo -n "Starting Jetty: "

  if (( NO_START )); then
    echo "Not starting jetty - NO_START=1";
    exit
  fi

  if [ $UID -eq 0 ] && type start-stop-daemon > /dev/null 2>&1
  then
    unset CH_USER
    if [ -n "$JETTY_USER" ]
    then
      CH_USER="-c$JETTY_USER"
    fi

    start-stop-daemon -S -p"$JETTY_PID" $CH_USER -d"$JETTY_HOME" -b -m -a "$JAVA" -- "${RUN_ARGS[@]}" --daemon

  else

    if [ -f "$JETTY_PID" ]
    then
      if running $JETTY_PID
      then
        echo "Already Running!"
        exit 1
      else
        # dead pid file - remove
        rm -f "$JETTY_PID"
      fi
    fi

    if [ "$JETTY_USER" ]
    then
      touch "$JETTY_PID"
      chown "$JETTY_USER" "$JETTY_PID"
      # FIXME: Broken solution: wordsplitting, pathname expansion, arbitrary command execution, etc.
      su - "$JETTY_USER" -c "
        exec ${RUN_CMD[*]} --daemon &
        disown \$!
        echo \$! > '$JETTY_PID'"
    else
      "${RUN_CMD[@]}" &
      disown $!
      echo $! > "$JETTY_PID"
    fi

  fi

  if expr "${CONFIGS[*]}" : '.*etc/jetty-started.xml.*' >/dev/null
  then
    if started "$JETTY_STATE" "$JETTY_PID"
    then
      echo "OK `date`"
    else
      echo "FAILED `date`"
    fi
  else
    echo "ok `date`"
  fi

  ;;

stop)
  echo -n "Stopping Jetty: "
  if [ $UID -eq 0 ] && type start-stop-daemon > /dev/null 2>&1; then
    start-stop-daemon -K -p"$JETTY_PID" -d"$JETTY_HOME" -a "$JAVA" -s HUP

    TIMEOUT=30
    while running "$JETTY_PID"; do
      if (( TIMEOUT-- == 0 )); then
        start-stop-daemon -K -p"$JETTY_PID" -d"$JETTY_HOME" -a "$JAVA" -s KILL
```

```
      fi

      sleep 1
    done

    rm -f "$JETTY_PID"
    echo OK
  else
    PID=$(cat "$JETTY_PID" 2>/dev/null)
    kill "$PID" 2>/dev/null

    TIMEOUT=30
    while running $JETTY_PID; do
      if (( TIMEOUT-- == 0 )); then
        kill -KILL "$PID" 2>/dev/null
      fi

      sleep 1
    done

    rm -f "$JETTY_PID"
    echo OK
  fi

  ;;

restart)
  JETTY_SH=$0
  if [ ! -f $JETTY_SH ]; then
    if [ ! -f $JETTY_HOME/bin/jetty.sh ]; then
      echo "$JETTY_HOME/bin/jetty.sh does not exist."
      exit 1
    fi
    JETTY_SH=$JETTY_HOME/bin/jetty.sh
  fi

  "$JETTY_SH" stop "$@"
  "$JETTY_SH" start "$@"

  ;;

supervise)
  #
  # Under control of daemontools supervise monitor which
  # handles restarts and shutdowns via the svc program.
  #
  exec "${RUN_CMD[@]}"

  ;;

run|demo)
  echo "Running Jetty: "

  if [ -f "$JETTY_PID" ]
  then
    if running "$JETTY_PID"
    then
      echo "Already Running!"
      exit 1
    else
      # dead pid file - remove
      rm -f "$JETTY_PID"
    fi
  fi

  exec "${RUN_CMD[@]}"
  ;;

check|status)
  echo "Checking arguments to Jetty: "
  echo "START_INI      =  $START_INI"
```

```
    echo "JETTY_HOME     =  $JETTY_HOME"
    echo "JETTY_CONF     =  $JETTY_CONF"
    echo "JETTY_PID      =  $JETTY_PID"
    echo "JETTY_START    =  $JETTY_START"
    echo "JETTY_LOGS     =  $JETTY_LOGS"
    echo "CONFIGS        =  ${CONFIGS[*]}"
    echo "CLASSPATH      =  $CLASSPATH"
    echo "JAVA           =  $JAVA"
    echo "JAVA_OPTIONS   =  ${JAVA_OPTIONS[*]}"
    echo "JETTY_ARGS     =  $JETTY_ARGS"
    echo "RUN_CMD        =  ${RUN_CMD[*]}"
    echo

    if [ -f "$JETTY_PID" ]
    then
      echo "Jetty running pid=$(< "$JETTY_PID")"
      exit 0
    fi
    exit 1

    ;;

  *)
    usage

    ;;
esac

exit 0
```

PostgreSQL

- PostgreSQL needs to be installed via yum to obtain the latest version. The CPR is using the following packages: postgresql9.x86_64, postgresql9-contrib.x86_64, postgresql9-devel.x86_64, postgresql9-libs.x86_64 and postgresql9-server.x86_64.
- Set up local users and groups for **postgres** and **cpruser**.
- **ln -s /opt/dbstore/cpr/db /var/pgsql9/data**.
- You will need to enable PostgreSQL to start up automatically, **chkconfig --level 2345 postgresql on**.
- On the master PostgreSQL node, you will need to edit the pg_hba.conf to add a line that enables replication for the new host.

```
hostssl  replication     replicator      54.244.223.0/24         md5
NOTE: the replica address is the IP address and CIDR mask.  For all of the AWS hosts, the CIDR mask will
be 24.  The above

line is an example.
```

- Next you will make a backup of the primary database for the new replica.

```
###
###  You execute this code on the new replica.  It is going to make a backup of the master and restore
###  it on the slave.
###  NOTE: the password for the replicator userid can be found here:
###  ec2-54-244-223-142.us-west-2.compute.amazonaws.com@/var/lib/pgsql9/data/recovery.conf
###
sudo -u postgres pg_basebackup -h 54.244.223.148 -D /var/lib/pgsql9/data -U replicator -v -P

echo Writing recovery.conf file
sudo -u postgres bash -c "cat > /var/lib/pgsql9/data/recovery.conf <<- _EOF1_
  standby_mode = 'on'
  primary_conninfo = 'host=54.244.223.148 port=5432 user=replicator password=********** sslmode=require'
  trigger_file = '/tmp/postgresql.trigger'
_EOF1_

###
### Again you will need to use the password from the backup.
###
```

- Because the backup was restored from the master, you will need to update the postgresql.conf file to comment out the following lines:

```
wal_level = hot_standby
max_wal_senders = 3
checkpoint_segments = 8
wal_keep_segments = 8
```

- You will need to add the following to the postgresql.conf file:

```
wal_level = hot_standby
max_wal_senders = 3
checkpoint_segments = 8
wal_keep_segments = 8
hot_standby = on
```

- At this point, you should be able to start the replica, "**service postgresql start**".

ActiveMQ

- There is a not a yum package for ActiveMQ, so you will need to AMQ that is packed in the .tar.gz file mentioned in the **Starting Point** section.
- You will need to create a symlink: *ln -s $CPR_HOME/apps/activemq /opt *The default recommended installation point for ActiveMQ is /opt /activemq.
- You will need to create the **/etc/default/activemq** file using the following information: Make sure the file is owned by **activemq:activemq**.

```
# ----------------------------------------------------------------------
# Configuration file for running Apache Active MQ as standalone provider
#
# This file overwrites the predefined settings of the sysv init-script
#
# Active MQ installation dir
if [ -z "$ACTIVEMQ_HOME" ] ; then
# try to find ACTIVEMQ
if [ -d /opt/activemq ] ; then
ACTIVEMQ_HOME=/opt/activemq
fi

if [ -d "${HOME}/opt/activemq" ] ; then
ACTIVEMQ_HOME="${HOME}/opt/activemq"
fi

## resolve links - $0 may be a link to activemq's home
PRG="$0"
progname=`basename "$0"`
saveddir=`pwd`

# need this for relative symlinks
dirname_prg=`dirname "$PRG"`
cd "$dirname_prg"

while [ -h "$PRG" ] ; do
ls=`ls -ld "$PRG"`
link=`expr "$ls" : '.*-> (.*)$'`
if expr "$link" : '.*/.*' > /dev/null; then
PRG="$link"
else
PRG=`dirname "$PRG"`"/$link"
fi
done

ACTIVEMQ_HOME=`dirname "$PRG"`/..

cd "$saveddir"

# make it fully qualified
ACTIVEMQ_HOME=`cd "$ACTIVEMQ_HOME" && pwd`
fi
```

```
if [ -z "$ACTIVEMQ_BASE" ] ; then
ACTIVEMQ_BASE="$ACTIVEMQ_HOME"
fi

# Active MQ configuration directory
if [ -z "$ACTIVEMQ_CONF" ] ; then

# For backwards compat with old variables we let ACTIVEMQ_CONFIG_DIR set ACTIVEMQ_CONF
if [ -z "$ACTIVEMQ_CONFIG_DIR" ] ; then
ACTIVEMQ_CONF="$ACTIVEMQ_BASE/conf"
else
ACTIVEMQ_CONF="$ACTIVEMQ_CONFIG_DIR"
fi
fi

# Configure a user with non root priviledges, if no user is specified do not change user
if [ -z "$ACTIVEMQ_USER" ] ; then
ACTIVEMQ_USER="activemq"
fi

# Active MQ data directory
if [ -z "$ACTIVEMQ_DATA" ] ; then

# For backwards compat with old variables we let ACTIVEMQ_DATA_DIR set ACTIVEMQ_DATA
if [ -z "$ACTIVEMQ_DATA_DIR" ] ; then
ACTIVEMQ_DATA="$ACTIVEMQ_BASE/data"
else
ACTIVEMQ_DATA="$ACTIVEMQ_DATA_DIR"
fi
fi

if [ -z "$ACTIVEMQ_TMP" ] ; then
ACTIVEMQ_TMP="$ACTIVEMQ_BASE/tmp"
fi

setCurrentUser(){
CUSER=`whoami 2>/dev/null`

# Solaris fix
if [ ! $? -eq 0 ]; then
CUSER=`/usr/ucb/whoami 2>/dev/null`
fi
}

if [ ! -d "$ACTIVEMQ_DATA" ]; then
setCurrentUser
if ( [ -z "$ACTIVEMQ_USER" ] || [ "$ACTIVEMQ_USER" = "$CUSER" ] );then
mkdir $ACTIVEMQ_DATA
elif [ "`id -u`" = "0" ];then
su -c "mkdir $ACTIVEMQ_DATA" - $ACTIVEMQ_USER;
fi
fi

# Location of the pidfile
if [ -z "$ACTIVEMQ_PIDFILE" ]; then
ACTIVEMQ_PIDFILE="$ACTIVEMQ_DATA/activemq-`hostname`.pid"
fi

# Location of the java installation
# Specify the location of your java installation using JAVA_HOME, or specify the
# path to the "java" binary using JAVACMD
# (set JAVACMD to "auto" for automatic detection)
#JAVA_HOME=""
JAVACMD="auto"

# Set jvm memory configuration
if [ -z "$ACTIVEMQ_OPTS_MEMORY" ] ; then
ACTIVEMQ_OPTS_MEMORY="-Xms1G -Xmx1G"
fi

# Uncomment to enable audit logging
```

```
#ACTIVEMQ_OPTS="$ACTIVEMQ_OPTS -Dorg.apache.activemq.audit=true"

# Set jvm jmx configuration
# This enables jmx access over a configured jmx-tcp-port.
# You have to configure the first four settings if you run a ibm jvm, caused by the
# fact that IBM's jvm does not support VirtualMachine.attach(PID).
# JMX access is needed for quering a running activemq instance to gain data or to
# trigger management operations.
#
# Example for ${ACTIVEMQ_CONF}/jmx.access:
# ---
# # The "monitorRole" role has readonly access.
# # The "controlRole" role has readwrite access.
# monitorRole readonly
# controlRole readwrite
# ---
#
# Example for ${ACTIVEMQ_CONF}/jmx.password:
# ---
# # The "monitorRole" role has password "abc123".
# # # The "controlRole" role has password "abcd1234".
# monitorRole abc123
# controlRole abcd1234
# ---
#
# ACTIVEMQ_SUNJMX_START="-Dcom.sun.management.jmxremote.port=11099 "
# ACTIVEMQ_SUNJMX_START="$ACTIVEMQ_SUNJMX_START -Dcom.sun.management.jmxremote.password.
file=${ACTIVEMQ_CONF}/jmx.password"
# ACTIVEMQ_SUNJMX_START="$ACTIVEMQ_SUNJMX_START -Dcom.sun.management.jmxremote.access.
file=${ACTIVEMQ_CONF}/jmx.access"
# ACTIVEMQ_SUNJMX_START="$ACTIVEMQ_SUNJMX_START -Dcom.sun.management.jmxremote.ssl=false"
ACTIVEMQ_SUNJMX_START="$ACTIVEMQ_SUNJMX_START -Dcom.sun.management.jmxremote"

# Set jvm jmx configuration for controlling the broker process
# You only have to configure the first four settings if you run a ibm jvm, caused by the
# fact that IBM's jvm does not support VirtualMachine.attach(PID)
# (see also com.sun.management.jmxremote.port, .jmx.password.file and .jmx.access.file )
#ACTIVEMQ_SUNJMX_CONTROL="--jmxurl service:jmx:rmi:///jndi/rmi://127.0.0.1:1099/jmxrmi --jmxuser
controlRole --jmxpassword abcd1234"
ACTIVEMQ_SUNJMX_CONTROL=""

# Specify the queue manager URL for using "browse" option of sysv initscript
if [ -z "$ACTIVEMQ_QUEUEMANAGERURL" ]; then
ACTIVEMQ_QUEUEMANAGERURL="--amqurl tcp://localhost:61616"
fi

# Set additional JSE arguments
ACTIVEMQ_SSL_OPTS="$SSL_OPTS"

# Uncomment to enable YourKit profiling
#ACTIVEMQ_DEBUG_OPTS="-agentlib:yjpagent"

# Uncomment to enable remote debugging
#ACTIVEMQ_DEBUG_OPTS="-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,
suspend=n,address=5005"

# ActiveMQ tries to shutdown the broker by jmx,
# after a specified number of seconds send SIGKILL
if [ -z "$ACTIVEMQ_KILL_MAXSECONDS" ]; then
ACTIVEMQ_KILL_MAXSECONDS=30
fi
```

- You will need to create a new Java keystore for the host you are dealing with. It has to the real DNS name, not the Amazon private names.

```
cd $CPR_HOME/apps/activemq/conf

openssl req \
  -x509 -nodes -days 7300 \
  -newkey rsa:2048 -keyout server.key -out server.crt

NOTE: Enter a "." for all of the fields except the server hostname.

cat server.crt server.key > chain.pem
openssl pkcs12 -export -in chain.pem -out chain.p12
keytool -importkeystore -srckeystore chain.p12 -srcstoretype PKCS12 -srcalias 1 -destkeystore commit.jks
-destalias commit

mv broker.ks broker.ks.DIST
ln -s commit.jks broker.ks

rm -f chain.pem chain.p12
```

- You need to add the new certificate to the cacerts bundle using the following instructions:

```
cd /etc/pki/java
keytool --import --trustcacerts --alias broker# --file $CPR_HOME/apps/activemq/conf/server.crt --
keystore cacerts

NOTE: The broker# value would be which server number you are working on.  Right now we have broker1 and
broker2.

NOTE: The password for the keystore is "changeit".
```

- After you have updated the cacerts bundle, you will need to copy it to all of the existing servers. All servers need to know about the certificates of each other. The cacerts bundle will need to go into **/etc/pki/java** on each of the boxes.
- When you restart ActiveMQ, it should attempt to obtain a lock on the database, which is normal. You should see something like this in the bottom of the log (**$ACTIVEMQ_HOME/data/activemq.log**). Also you can ignore all of the messages about unable to create database tables. Those tables were created by the master broker when it was installed.

```
2013-07-12 18:28:12,337 | INFO  | Attempting to acquire the exclusive lock to become the Master broker | org.
apache.activemq.store.jdbc.DefaultDatabaseLocker | main
```

- You will need to update the **$CPR_HOME/files/properties/cpr.properties** file with the name of the new host in the failover stanza. This update will need to be pushed out to all of the servers. In addition, Jetty will need to be restarted to pick up the new change.

```
## Failover stanza

cpr.jms.broker=failover:(ssl://ec2-54-244-223-148.us-west-2.compute.amazonaws.com:61617,ssl://ec2-54-244-223-
142.us-west-2.compute.amazonaws.com:61617)?jms.useAsyncSend=true
```

- Lastly you need to add the startup script. The code that follows needs to be placed in a file called **/etc/init.d/activemq**. Make sure the file is executable, **chmod +x /etc/init.d/activemq**. And lastly, you need to add it to the startup, **chkconfig --add activemq**.

```
#!/bin/bash
#
# activemq
#
# chkconfig: 2345 99 99
#
# description:  Start up the ActiveMQ server.
# Source function library.
. /etc/init.d/functions

RETVAL=$?
ACTIVEMQ_HOME="/opt/activemq"
case "$1" in
 start)
        if [ -f $ACTIVEMQ_HOME/bin/activemq ];
          then
        echo $"Starting ActiveMQ"
            /bin/su activemq $ACTIVEMQ_HOME/bin/activemq start
        fi
    ;;
 stop)
        if [ -f $ACTIVEMQ_HOME/bin/activemq ];
          then
        echo $"Stopping ActiveMQ"
            /bin/su activemq $ACTIVEMQ_HOME/bin/activemq stop
        fi
    ;;
 *)
    echo $"Usage: $0 {start|stop}"
    exit 1
    ;;
esac
exit $RETVAL
```

Directory Provisioner

The directory provisioner is a standalone Java application that needs to be executed on one of the boxes. On ec2-54-244-223-148.us-west-2.compute.amazonaws.com, the directory provisioner can be found in: **/opt/dbstore/cpr/apps/directory_provisioner** The configuration for the provisioner can be found in the cpr.properties file. *The important things to configure are the LDAP server and the credentials used to allow add/update of entries. I typically start the directory provisioner by executing:

```
nohup ./run_dp.sh &
```

CPR Core and UI

The CPR Core and UI are contained in two .war files, **cprws.war** and **IdentityProvisioning.war**. Both files must be copied to **$JETTY_HOME/webapps**.