SIB Developer Manual

Version 1.37, 2020-12-30 19:58:19 -0600

Table of Contents

Chapter 1. Overview

This guide has been split into a few key sections. The following sections will provide details for each step of the process. The build process for SIB contains many parts that need to be performed in a specific order. It is recommended to use a build system such as Jenkins to automate this process. The layout of the process has been taken directly from the steps performed by Jenkins. Additionally, examples of how to make some common modifications will be given. Knowledge of how to use GIT is required to properly maintain local customizations to SIB.

The SIB source code is made available at GitHub in the following repositories:

`deployment`
: Contains the SIB installer.

`db`
: Database and database upgrade paths.

`services`
: Java and Perl code for assessments.

`swamp-web-server`
: SIB web server that includes customized Laravel framework code.

`www-front-end`
: Web front end code.

Additional supporting code is contained in the following repositories:

- `c-assess`
- `java-assess`
- `ruby-assess`
- `script-assess`
- `resultparser`

This document does not cover building the frameworks or result parser.

Chapter 2. Before You Begin

Building SIB includes components that are not built with the build process described in this document. These compoents are included with the SIB installer to be installed in an SIB instance.

2.1. Frameworks and HTCondor

2.1.1. Frameworks

These are found as directories within the `/opt/swamp/thirdparty` directory on an already installed SIB instance. Recreate the `.tar` files from the directories. Make sure you are extracting these frameworks from the same SIB release version that you are attempting to build. SIB releases depend on specific versions of these frameworks.

The GitHub repositories listed above are incomplete. Do NOT attempt to create the required `.tar` files from the GitHub repositories.

Obtaining Frameworks and Result Parser Example

```
cd /opt/swamp/thirdparty
for framework in c-assess-1.2.6 \
java-assess-2.7.6 \
ruby-assess-1.2.5 \
script-assess-1.4.5 ;
do tar cf /tmp/${framework}.tar ${framework} ; done
tar cf /tmp/resultparser-3.3.4.tar resultparser-3.3.4
```

Copy the `.tar` files to `/swampcs/releases` on the machine you are building SIB.

2.1.2. HTCondor

SIB includes a custom release of HTCondor. To successfully build the SIB installer, copy the HTCondor `.tar` files from `<sib installer>/dependencies/htcondor` directory to `/swampcs/htcondor` on the machine you are building SIB.

Chapter 3. Building SIB

3.1. Obtaining the SIB Source

Obtain the SIB source by cloning the following GIT repositories from GitHub. Next, create branches based on the release <tag> for the current release.

`deployment`

```
git clone https://github.com/mirswamp/deployment.git
cd deployment; git checkout tags/v1.37 -b 1.37-release
```

`db`

```
git clone https://github.com/mirswamp/db.git
cd db; git checkout tags/v1.37 -b 1.37-release
```

`services`

```
git clone https://github.com/mirswamp/services.git
cd services; git checkout tags/v1.37 -b 1.37-release
```

`swamp-web-server`

```
git clone https://github.com/mirswamp/swamp-web-server.git
cd swamp-web-server; git checkout tags/v1.37 -b 1.37-release
```

`www-front-end`

```
git clone https://github.com/mirswamp/www-front-end.git
cd www-front-end; git checkout tags/v1.37 -b 1.37-release
```

3.2. GitHub vs Workspace Repository Clones

The repository clones (GitHub Clones) created in the previous section are where you will commit your local customizations. During the building of SIB, you will be building from clones of the ***GitHub Clones***, these clones will be called ***Workspace Clones***.

In this example, the current directory is `~/sib`.

GitHub Clone

```
mkdir sibsource
cd sibsource
git clone https://github.com/mirswamp/deployment.git
cd deployment; git checkout tags/v1.37 -b 1.37-release
```

Workspace Clone

```
mkdir sibbuild
cd sibbuild
git clone ../sibsource/deployment deployment
cd deployment; git checkout tags/v1.37 -b 1.37-release
```

3.3. Build Parameters

The following parameters are used for building SIB. `BRANCH`, `RELEASE_NUMBER`, and `BUILD_TAG` are typically set up as Build Parameters in Jenkins and `WORKSPACE` and `BUILD_NUMBER` are auto populated by Jenkins.

- `BRANCH`
- `RELEASE_NUMBER`
- `BUILD_TAG`
- `BUILD_NUMBER`
- `WORKSPACE`

3.4. Building the Tools Bundle

The following GIT repositories are used for building the tools bundle.

- `deployment`

If there are no changes to the tools bundle, you can skip building it and download `swampinabox-1.37-tools.tar.gz`. To rebuild the tools bundle, download the version corresponding to your SIB release and extract it and move the files to `/swampcs/releases`. Otherwise, copy it to your `BUILD_ROOT` directory. It is also included with initial installer download. Simply copy this file into your `BUILD_ROOT` to save a step of downloading the file again.

Save the following script as `build_tools_bundle.bash` in the directory where your ***Workspace Clones*** are located. This location is your workspace. Edit the parameters at the top of the script as necessary.

build_tools_bundle.bash

```
#!/usr/bin/env bash

# Set these to a location with significant space
BUILD_ROOT_LOCATION=/sib/buildroot ①
WORKSPACE=/sib ②

# Build versioning parameters
RELEASE_NUMBER=1.37 ③
BRANCH=1.37-release ④
BUILD_TAG=sib ⑤
BUILD_NUMBER=1 ⑥

TYPE=tools
ARCHIVE_ROOT_DIR=swampinabox-${RELEASE_NUMBER}-${TYPE}
SIB_ROOT=$WORKSPACE/deployment/swampinabox
BUILD_ROOT=${BUILD_ROOT_LOCATION}/swampinabox/bundles/$ARCHIVE_ROOT_DIR

rm -rf "${ARCHIVE_ROOT_DIR}" "${BUILD_ROOT}"

mkdir -p "${BUILD_ROOT}"

"$SIB_ROOT"/distribution/util/build-archive.pl \
    --inventory    "$WORKSPACE/deployment/inventory/tools-bundled.txt" \
    --inventory    "$WORKSPACE/deployment/inventory/tools-metric.txt" \
    --output-file  "${BUILD_ROOT}/${ARCHIVE_ROOT_DIR}.tar.gz" \
    --root-dir     "${ARCHIVE_ROOT_DIR}" \
    --version      "${RELEASE_NUMBER}" \
    --build        "${BUILD_NUMBER}" \
  || exit 1

for path in "$BUILD_ROOT"/* ; do
    if [ -f "$path" ]; then
        { cd -- "$(dirname -- "$path")" && md5sum "$(basename -- "$path")"
        } > "$path".md5
    fi
done

chmod -R u=rwX,og=rX "$BUILD_ROOT"
```

① Set this to the path of where the build artifacts will be saved.

② Set this to the path of your WORKSPACE location. If you are attmpting to use this script inside Jenkins, comment this out.

③ Set this to the release number. If you are attempting to use this script inside Jenkins, make this a build parameter and comment this out.

④ Set this to the branch you are building. If you are attempting to use this script inside Jenkins, make this a build parameter and comment this out.

⑤ Set this to the tag of the build. It can be any string with the purpose of labeling if a build is a development build or not. If you are attempting to use this script inside Jenkins, make this a build parameter and comment this out.

⑥ Set this to the build number. Increment this between builds of the same release so that they will be treated as upgrades. If you are attempting to use this script in Jenkins, comment this out.

3.5. Building the SIB Release

The tools bundle must be in place prior to building SIB itself.

This step assumes that all local customizations have been committed to the ***GitHub Clones***. Instructions for where to apply local customizations within the source code will be shown later in this document. So, commit your local customizations and then generate your ***Workspace Clones*** into a clean workspace prior to initiating a new build. The build scripts utilized were written with the behavior of Jenkins in mind. So, don't perform the build using ***GitHub Clones*** directly.

Install the following RPM packages on the machine you are building SIB.

Install Build Dependencies

```
yum install rpm-build cmake ruby gcc doxygen libxml2-devel
```

Save the following script as `build_sib_installer.bash` in the workspace directory where you created the ***WorkSpace Clones***. Edit the parameters at the top of the script as necessary.

build_sib_installer.bash

```
#!/usr/bin/env bash

# Set these to a location with significant space
BUILD_ROOT_LOCATION=/sib/buildroot ①
WORKSPACE=/sib ②

# Build versioning parameters
RELEASE_NUMBER=1.37 ③
BRANCH=1.37-release ④
BUILD_TAG=sib ⑤
BUILD_NUMBER=1 ⑥

SIB_ROOT=$WORKSPACE/deployment/swampinabox
BUILD_ROOT=$WORKSPACE/export/swampinabox/distribution

mkdir -p "${BUILD_ROOT}"

"$SIB_ROOT"/distribution/util/build-installer.bash \
    "${RELEASE_NUMBER}" \
    "${BUILD_NUMBER}.${BUILD_TAG}" \
  || exit 1

"$SIB_ROOT"/distribution/util/build-release.bash \
    "${RELEASE_NUMBER}" \
    "${BUILD_ROOT_LOCATION}/swampinabox/builds/${RELEASE_NUMBER}-
${BUILD_NUMBER}.${BUILD_TAG}" \
    "${BUILD_ROOT}/swampinabox-${RELEASE_NUMBER}-installer" \
    "${BUILD_ROOT_LOCATION}/swampinabox/bundles/swampinabox-${RELEASE_NUMBER}-tools" \
  || exit 1
```

① Set this to the path of where the build artifacts will be saved.

② Set this to the path of your WORKSPACE location. If you are attmpting to use this script inside Jenkins, comment this out.

③ Set this to the release number. If you are attempting to use this script inside Jenkins, make this a build parameter and comment this out.

④ Set this to the branch you are building. If you are attempting to use this script inside Jenkins, make this a build parameter and comment this out.

⑤ Set this to the tag of the build. It can be any string with the purpose of labeling if a build is a development build or not. If you are attempting to use this script inside Jenkins, make this a build parameter and comment this out.

⑥ Set this to the build number. Increment this between builds of the same release so that they will be treated as upgrades. If you are attempting to use this script in Jenkins, comment this out.

This build environment needs to be ran as a normal user. To initiate a build, run `build_sib_installer.bash`. When the script finishes, you will see the following message:

Successful Build Message

```
Finished building the installer

### Assembling SWAMP-in-a-Box Release

Version:            1.37
Release directory:  /sib/buildroot/swampinabox/builds/1.37-1.sib
Working directory:  /sib

Finished assembling the release
```

The new build artifacts have been saved to the release directory. This build can then be used to upgrade an existing SIB instance to this build that now includes the local customizations.

If there were errors detected in the build process, the following message will be displayed instead.

Build Failure Message

```
Finished building the installer, but with errors
```

If your build does not display either of these messages, something went wrong beyond the error dection of the build scripts. Examine the build output generated for failures. This typically happens when a build time dependency has not been met.

Chapter 4. Customizing SIB

4.1. Documentation

The SIB documentation is built separately from SIB itself. However, the built documentation files are included within a build of a SIB release. The documentation resides in the `deployment` repository in the `swampinabox/runtime/doc` directory.

Modifications to the documentation occur in the ***GitHub Clone***.

4.1.1. Before You Begin

The documentation is built using `asciidoctor` and `asciidoctor-pdf`. Perform these steps to install these tools.

Install asciidoctor

```
yum install asciidoctor
```

Install asciidoctor-pdf

```
yum install centos-release-scl-rh
yum --enablerepo=centos-sclo-rh -y install rh-ruby23
scl enable rh-ruby23 'gem install asciidoctor-pdf --pre'
```

4.1.2. Building The Documentation

There are two types of files envolved. The AsciiDoc source files and the generated HTML and PDF files. Refer to `Asciidoctor Documentation` for the proper syntax for modifying the documentation.

The AsciiDoc source files are contained within subdirctories of the `swampinabox/runtime/doc` directory. The current manuals are these:

- *SIB Administrator Manual* in the `administrator_manual` directory
- *SIB Reference Manual* in the `reference_manual` directory
- *SIB Developer Manual* in the `developer_manual` directory

There is a script that is ran to generate the PDF and HTML files from the AsciiDoc sources. This script is in the `swampinabox/distribution/util` directory.

In this example, the current directory is `swampinabox/runtime/doc`:

build_manuals.bash Example

```
scl enable rh-ruby23 ../../distribution/util/build_manuals.bash
```

Once the `build_manuals.bash` script completes, commit the changes to the AsciiDoc sources, the HTML files, and the PDF files. The HTML and PDF files will be included in the SIB installer.

4.2. RPMs

SIB uses RPM as its package management format. The following RPMs are built and included in the SIB installer. The additional separate `.txt` files are shared with the SWAMP facility packages.

Package `swamp-rt-perl`

- RPM `swamp-rt-perl-1.37-1.sib.noarch.rpm`
- SPEC `swamp-rt-perl.spec`
- Location
  - `deployment/swamp/runtime-installer/SPECS`

Package `swampinabox-backend`

- RPM `swampinabox-backend-1.37-1.sib.norarch.rpm`
- SPEC `swampinabox-backend.spec`
- Location
  - `deployment/swmap/installer/SPECS`
- Additional Files
  - `common-files-data.txt`
  - `common-files-exec.txt`
  - `common-files-submit.txt`
  - `common-install-data.txt`
  - `common-install-exec.txt`
  - `common-install-submit.txt`
  - `swampinabox-files-data.txt`
  - `swampinabox-files-exec.txt`
  - `swampinabox-files-general.txt`
  - `swampinabox-files-submit.txt`
  - `swampinabox-install-data.txt`
  - `swampinabox-install-exec.txt`

    - swampinabox-install-general.txt
    - swampinabox-install-submit.txt

Package swamp-web-server

- RPM swamp-web-server-1.37-1.sib.noarch.rpm
- SPEC swamp-web-server.spec
- Location
    - deployment/swmap/swamp-web-server-installer/SPECS

Chapter 5. Perl Architecture

There are currently three deployment flavors of SWAMP - the original multi-host facility deployment, the singleserver facility deployment *(primarily used for development purposes but should be extended as a replacement for the multi-host version in the future)*, and the swampinabox deployment intended for distribution and installation at user sites.

For the multi-host design, the relevant host nodes consist of:

- Web Server
  - The HTML service routes are executed on this server.
- Data Server
  - The MariaDB database server is executed here.
- Submit Server
  - The SWAMP daemon(s) execute here.
- Execute Servers (one or more)
  - The viewer and assessment machines *(virtual machines 'vm' and docker containers 'dc')* execute here.

There are additional host(s) used for the installation and deployment of HTCondor which is used as a job management tool. In particular, there is a HTCondor host where the collector runs. This collector is briefly described below.

The Perl backend is the component that manages jobs in the SWAMP system.

There are two kinds of jobs:

- assessments
- viewers

All jobs are submitted to HTCondor to be executed in either a virtual machine (vm universe) or a docker container (docker universe). One fundamental difference between assessment jobs and viewer jobs is that assessment jobs are expected to run to completion; typically on the order of minutes for processing, while viewer jobs are run under end user control and therefore are expected to run indefinitely as long as the user's client web application interface maintains a connection to the viewer machine.

The Perl backend is responsible for responding to requests from the database to launch assessment and viewer jobs and to kill jobs by removing them from the HTCondor queue. All job management is performed by executing operating system shell level commands using the perl `system` procedure call. All requests initiated by the database are performed using the `sys_exec` and `sys_eval` user extensions to the MariaDB server. The two database user commands only differ in that `sys_exec` executes shell commands without gathering the command output, while `sys_eval` provides the extra step of gathering

the output of the commands and returns it to the caller.

Each job has an execution record uuid (execrunuid) that uniquely identifies the job. There is also an HTCondor job id that is used to identify the job once it is submitted to and managed by HTCondor. The mechanism for making the correspondence between the SWAMP execrunuid and the HTCondor job is a Perl subroutine `getHTCondorJobId` that invokes the `condor_q` command with the execrunuid as a constraint.

Assessment and viewer job progress is monitored by Perl monitoring scripts that are launched with each job and communicate with the job's machine (vm or dc) via a tty to file connection. Inside the machine, the connection looks like a tty, outside the machine, the monitoring script sees a file. Communication between the Perl monitoring scripts and the rest of the SWAMP system is obtained by use of HTCondor's collector functionality. A separate HTCondor collector is started when the SWAMP system is installed, that is used to facilitate this communication. In this way, the web service code can check the HTCondor collector for status information regarding jobs that are initiated by the web service.

5.1. Database `bash` Scripts Called by `sys_exec` and `sys_eval`

There are three entry points into the Perl backend from the database. They are simple bash scripts that call their corresponding Perl scripts, passing all arguments and returning all results.

- `/usr/local/bin/launch_viewer` - launch a viewer job in HTCondor
- `/usr/local/bin/kill_run` - remove an assessment or viewer job from the HTCondor queue
- `/usr/local/bin/execute_execution_record` - launch an assessment job in HTCondor

5.1.1. Perl Entry Scripts Called by Database `bash` Scripts

These three Perl scripts correspond directly with their bash counterparts described above. Each of these scripts perform setup computations and then use `RPC::XML` to communicate requests to the child daemons described below. These scripts take an execrunuid as argument from the database invocation as the SWAMP job identifier. When needed, these scripts execute database queries to obtain the requisite data to submit the job or perform the task.

`vmu_launchviewer.pl`

This script is the backend entry point for submitting an HTCondor viewer job, monitoring its progress, and collecting the output when the job finishes. Collected output consists of the CodeDX viewer database, the CodeDX configuration files, and the CodeDX log files. This script is also responsible for preparing native viewer json data and error/warning report json data. In these later cases, no HTCondor job is submitted.

This script prepares the input directory for the viewer machine, prepares the HTCondor submit file, and submits the HTCondor job. In the case of CodeDX viewer jobs, this script uploads the package archive and the viewer SCARF results to the viewer machine using the `curl` command via the Perl

`system` call. It is capable of uploading to a personal CodeDX viewer machine or an enterprise wide CodeDX installation.

`vmu_calldorun.pl`

This is the analagous backend entry point for submitting an HTCondor assessment job. It has similar charactersitics as the viewer launch script. Collected output consists of the archive of the output disk of the assessment machine which contains either the assessment results if successful, or the error report data in the case of failure.

`vmu_killrun.pl`

This script is the backend entry point for removal of the HTCondor job that corresponds to a SWAMP job. Translation is made between the SWAMP execrunuid and the HTCondor job id and `condor_rm` is used to shutdown the job. For viewer jobs, and attempt is made to ensure a graceful shutdown so that the viewer output can be collected. For assessment jobs, a hard kill is performed and any job results are ignored.

5.2. Daemons Managed by The `swamp` Init Script

The SWAMP daemon(s) are architected for the original multi-host SWAMP facility. They are used in the singleserver and SIB versions of SWAMP to keep the code base uniform across implementation flavors. A strongly recommended future direction is to eliminate the multi-host implementation, move all job control functionality to the web service code, and thereby eliminate the Perl backend altogether.

The daemon(s) are implemented as an operating system service. The entry point for the service is the vmu_swamp_monitor. It is controlled by `systemctl start`/`stop`/`status`. It in turn controls the remaining Perl script child daemons that are written to run indefinitely, detached from a tty. There is keep-alive functionality built in to the service that restarts child daemons when their crash is detected.

5.2.1. `vmu_swamp_monitor`

This is the operating system service daemon. It serves only to manage the child daemons.

5.2.2. `vmu_LaunchPad.pl`

This child daemon serves to kill HTCondor jobs via condor_rm, and to launch assessment jobs after the BOG (Bill of Goods) is prepared.

5.2.3. `vmu_AgentMonitor.pl`

This child daemon serves to launch viewer jobs after the BOG is prepared. This is written as a separate daemon to simplify giving viewer jobs higher priority than other tasks.

5.2.4. vmu_csa_agent.pl

This child daemon manages preparation of HTCondor job submit files from BOGs. Assessment jobs have their BOG written to the filesystem and this daemon monitors the filesystem for such BOGs to be executed in queue fashion. There is a single instance of this daemon that runs continuously, monitoring for assessment job BOGs.

This child daemon is also started on demand to submit HTCondor jobs for viewers. This provides for immediate response to a viewer request.

5.2.5. vmu_perl_launcher

This is not a daemon. It is a script referenced by the vmu_csa_agent.pl daemon described above. It is the preamble setup script handed off to HTCondor via the submit file, that is used to manage the execution of the HTCondor jobs scripts described below.

5.3. HTCondor Job Scripts

These scripts are essentially controlled by HTCondor after the job's submission, and in the case of the multi-host facility, run on the execute node along side the job machine (vm or dc). The `Pre*` scripts are HTCondor hooks invoked after the HTCondor job is submitted and before its machine is started. Each `Monitor*` script is started by the corresponding `Pre*` script so it is not technically started by HTCondor, but HTCondor does control removing this script upon cleanup of abnormal termination of the HTCondor job.

In the case of assessment jobs, the framework that runs the assessment includes the controlling script that is started by the init system of the assessment machine (vm or dc). Communication with this controlling script is obtained by a configuration file. In the case of viewer jobs, the controlling script that is started by the init system of the viewer machine (vm or dc) is configured and passed in to the machine in the input directory.

5.3.1. vmu_PreAssessment.pl

This script creates the input and output directories of the machine. In the case of vm, they are implemented as mountable filesystems where mounting is done in the machine. In the case of dc, they are implemented as folders that are accessed in the filesystem of the machine's execution environment. The script populates the input directory with the framework needed to run the assessment, and patches the machine's init system to run a prescribed script on startup. This allows for the unattended execution of assessments. This script sets up the monitoring connection between the machine and its execution environment.

5.3.2. vmu_MonitorAssessment.pl

This script opens a connection to the machine via the filesystem. It reads single word status lines from the event file until it detects the final status of a successfully executed machine task. These statuses are

communicated to the rest of the SWAMP system (in particular the web service) via the HTCondor collector. In the case of abnormal termination of a job, this script is expected to have transmitted its most recently detected status to the collector and it is cleaned up by the HTCondor supervisor.

5.3.3. vmu_PostAssessment.pl

This script archives the input and output of the machine and stores it in a temporary location for analysis. Upon completion of the job, the output archive is also stored permanently in the filesystem with its location recorded and managed by the database.

5.3.4. vmu_PreViewer.pl

This script is analagous to the assessment job version. A major difference is that it prepares the controlling script and puts it in the input directory instead of preparing a configuration file to drive an embedded script.

5.3.5. vmu_MonitorViewer.pl

This script works analagously to the assessment job version.

5.3.6. vmu_PostViewer.pl

This script collects job output which consists of the codedx user database, configuration, package archive files and some log files that are useful for debugging.

5.4. SCARF Stream Parsing to JSON

This subsystem is used to provide stream parsing of assessment job output SCARF xml files. This is a performance requirement since the Perl code and modules available to perform the task of parsing xml files is not able to handle extremely large files without exceeding memory requirements and causing an execution crash. This code is written in the C language and utilizes framework support for stream parsing of SCARF xml files. The Makefile to build the executable stream parsing program is provided with the subsystem. It is invoked as part of the installation.

```
Makefile
AttributeJsonReader.c
AttributeJsonReader.h
ScarfCommon.h
ScarfJson.h
ScarfJsonWriter.c
ScarfXml.h
ScarfXmlReader.c
vmu_Scarf_Cparsing.c
```

This source code can be found in the `services` repository in the `services/perl/agents/Scarf_Parsing_C` directory.

5.5. Perl Backend Support Modules

These Perl modules are written uniformly to expose their set of subroutines to caller scripts.

5.5.1. `vmu_Support.pm`

This provides general suppport subroutines that are used by both assessment jobs and viewer jobs. There are logging support routines, configuration support routines, HTCondor support routines, libvirt support routines (for virtual machines) and operating system interface routines.

5.5.2. `vmu_AssessmentSupport.pm`

This provides the assessment job specific routines. There are database routines, routines to program the job control state machine, and some configuration routines.

5.5.3. `vmu_ViewerSupport.pm`

This provides the viewer job specific routines. There are database routines, viewer state routines to support managing the viewer state in the collector, and logging support routines.

5.5.4. `CodeDX.pm`

This provides the interface to the codedx tomcat based web application. Communication and data transmission is obtained by curl command calls, using the Perl `system` command.

5.5.5. `FloodlightAccess.pm`

This provides the interface to the `floodlight` flow control software that is used in the multi-host facility in order to constrain access to the assessment machine public ip space. This machine ip space is typically accessed from the general public ip space to allow end user access to assessment machines for inspection while jobs are executing. It also controls internal access to license servers for those commercial tools that require license for operation and use.

5.5.6. `FrameworkUtils.pm`

This provides the error analysis functionality that is driven the the assessment `status.out` file produced by the framework. There is an extensive document that describes the results produced by the assessment framework, with instructions for its interpretation and presentation.

5.5.7. `Locking.pm`

This provides locking and mutual exclusion for the daemons to ensure that only one daemon or

background process can run at any time.

5.5.8. mongoDBUtils.pm

This provides the interface to MongoDB which is utilized for storing large data blobs. Future work includes searching and filtering of result data.

5.5.9. PackageTypes.pm

This provides the enumeration of the package type constants that match the constants in the database. This data should ideally be queried from the database as the authoritative source for this data so as to avoid synchronization issues.

5.5.10. ScarfXmlReader.pm

This provides the Perl support for stream parsing of SCARF xml files.

5.6. HTCondor Command Utilization

Table 1. Subroutines That Call HTConder Commands (via Module)

Module	Subroutine	Command(s)
vmu_ViewerSupport	updateClassAdViewerStatus	condor_advertise
	getViewerStateFromClassAd	condor_status
vmu_AssessmentSupport	updateClassAdAssessmentStatus	condor_advertise
vmu_Support	getHTCondorJobId	condor_status
		condor_q
	HTCondorJobStatus	condor_status
		condor_q
	identifyPreemptedJobs	condor_q
	isJobInHistory	condor_history
	isJobInQueue	condor_q
	getRunDirHistory	condor_history

Table 2. Subroutines That Call HTConder Commands (via Script)

Calling Script (* Daemon)	Subroutine	Command(s)
vmu_csa_agent.pl*	condorJobExists	condor_history
		condor_q
	startHTCondorJob	condor_submit
vmu_killrun.pl via vmu_LaunchPad.pl*	_launchpadKill	condor_rm

Table 3. Subroutines Called by Script

Subroutine	Script
updateClassAdViewerStatus	vmu_killrun.pl
	vmu_csa_agent.pl
	vmu_AgentMonitor.pl
	vmu_launchviewer.pl
	vmu_PreViewer.pl
	vmu_MonitorViewer.pl
	vmu_PostViewer.pl
getViewerStateFromClassAd	vmu_AgentMonitor.pl
	vmu_launchviewer.pl
updateClassAdAssessmentStatus	vmu_killrun.pl
	vmu_csa_agent.pl
	vmu_calldorun.pl
	vmu_PreAssessment.pl
	vmu_MonitorAssessment.pl
	vmu_PostAssessment.pl
getHTCondorJobId	vmu_killrun.pl
HTCondorJobStatus	vmu_killrun.pl
identifyPreemptedJobs	vmu_csa_agent.pl
condorJobExists	vmu_csa_agent.pl
startHTCondorJob	vmu_csa_agent.pl
launchPadKill	vmu_killrun.pl

Table 4. HTCondor Commands Called by Subroutine

Command	Subroutine
condor_advertise	updateClassAdViewerStatus
	updateClassAdAssessmentStatus
condor_status	getViewerStateFromClassAd
	_getHTCondorSubmitNode
condor_q	getHTCondorJobId
	HTCondorJobStatus
	identifyPreemptedJobs
	isJobInQueue
condor_history	isJobInHistory
	getRunDirHistory
condor_submit	startHTCondorJob

Command	Subroutine
condor_rm	_launchpadKill

5.7. Building the Perl Runtime Distribution Archive

These instructions assume that the SWAMP deployment project from git has been cloned into SWAMP/deployment. The literal string <major.minor.patch> in these instructions are to be substituted with the literal string representing the major.minor.patch version string of the new Perl core installation.

The Perl runtime must be installed to /opt/perl5.

Install perlbrew to obtain the core Perl distribution

```
curl -L https://install.perlbrew.pl | bash
export PERLBREW_ROOT=/opt/perl5
perlbrew init
perlbrew install perl-<major.minor.patch> ①
export PATH=/opt/perl5/perls/perl-<major.minor.patch>/bin:$PATH ①
```

① E.g. <major.minor.patch> = 5.26.1

Install additional Perl modules using cpanm

```
Log::Log4perl
File::Remove
XML::Parser ①
RPC::XML::Server
Readonly
ConfigReader::Simple
Data::UUID
DBI
JSON
Date::Parse
XML::LibXML
XML::LibXSLT ②
XML::XPath
DBD::mysql ③
File::Slurp
MongoDB
XML::Simple
```

① XML-Parser requires expat-devel (yum install expat-devel)

② XML::LibXSLT requires libxslt and libxml2 and their *-devel packages (yum install libxml2-devel libxslt-devel)

③ DBD::mysql requires mysql-devel (`yum install mysql-devel`)

Additional Perl modules for SWAMP Perl Runtime are installed using `cpanm <module-name>`. Repeat this command while replacing `<module-name>` with a module from the list above.

The Perl runtime must be built on CentOS 6 to be compatible with both CentOS 6 and CentOS 7.

Building the Perl Runtime Archive

```
cd /opt/perl5

# Remove all folders except the one named 'perls'.
# /opt/perl5/perls should have one folder
# named `perl-<major.minor.patch>'. ①

cd /opt
sudo tar -czvf perlbin_exec_w-<major.minor.patch>.tar.gz ./perl5
mv perlbin_exec_w-<major.minor.patch>.tar.gz SWAMP/deployment/perl/.

# edit SWAMP/deployment/inventory/dependencies.txt
#  change the perl: to read perl:<major.minor.patch> ①

# To perform Perl syntax checking on the SWAMP Perl code
# after the Perl Runtime upgrade:

cd SWAMP/services/perl/agents
for f in *.pl; do perl -cw $f; done
cd SWAMP/services/perl/vmtools
for f in masterify_vm start_vm vm_cleanup vm_masterinfo vm_output; do perl -cw $f; done
```

① e.g. `<major.minor.patch>` = `5.26.1`