restraint Documentation

Release 0.1.39

Bill Peck, Dan Callaghan, Jeff Bastian

May 14, 2019

Contents

1 Features **3**

1.1 Installing . . . 3
1.2 Starting the Daemon . . . 5
1.3 Commands . . . 5
1.4 Jobs . . . 9
1.5 Tasks . . . 11
1.6 Variables . . . 14
1.7 Plugins . . . 15
1.8 Using Restraint . . . 19
1.9 Release Notes . . . 21
1.10 Developer Guide . . . 23
1.11 ToDo . . . 24

2 Additional Information **25**

3 Indices and Tables **27**

Restraint is designed to execute tasks. These tasks can be tests which report results or simply code that you want to automate. Which tasks to execute is determined by a job[1]. The job also describes where to retrieve the tasks from and what parameters to pass in. These tasks can report multiple PASS, FAIL, WARN results along with an optional score. Tasks also have the ability to report log files. Each task can have metadata describing dependencies and max run time for example. Execution and reporting can be further enhanced with plugins.

Restraint can be used with Beaker[2] since it talks Beaker's Harness API[3] for reporting results. It can also be used stand-alone.

[1] Job XML.
[2] Beaker is open-source software for managing and automating labs of test computers.
[3] Alternate Harness API.

CHAPTER 1

Features

- Tasks can be retrieved directly from git.
- Does not rely on Anaconda/kickstart to install task dependencies.
- Can be statically linked to make it easier to test the system without changing the system.
- Can be run stand-alone without Beaker.
  - Tasks are executed with the same environment (no surprises when run later in Beaker).
  - Developing tasks is much quicker since you don't have to build task RPMs, schedule a system, provision a system, etc…
- Can be easily extended with Plugins.
- Uses Beaker's job XML.

The following documentation will show you how to use Restraint in both environments.

Contents:

1.1 Installing

1.1.1 Installing from RPM

Pre-built statically linked versions are available for the following OSes:

- RedHatEnterpriseLinux
- Fedora
- CentOS

To get the appropriate repo file for your OS, use one of the commands listed below:

- RedHatEnterpriseLinux

```
# sudo wget -O /etc/yum.repos.d/beaker-harness.repo https://beaker-project.org/yum/
↪beaker-harness-RedHatEnterpriseLinux.repo
```

- Fedora

```
# sudo wget -O /etc/yum.repos.d/beaker-harness.repo https://beaker-project.org/yum/
↪beaker-harness-Fedora.repo
```

- CentOS

```
# sudo wget -O /etc/yum.repos.d/beaker-harness.repo https://beaker-project.org/yum/
↪beaker-harness-CentOS.repo
```

Once you have the appropriate repo file on your system you can install Restraint via dnf (or yum on older systems). Although you can install both the server and the client on the same machine it is not recommended.

Install the Restraint client on your machine if you want to run stand-alone jobs (i.e.: outside of Beaker):

```
# sudo dnf install restraint-client
```

Install the Restraint server on the systems that will run the tasks/tests:

```
# sudo dnf install restraint
```

1.1.2 Building from Source

Source code is located at https://git.beaker-project.org/cgit/restraint/. Restraint can be built and linked dynamically or statically. To build it dynamically you will need the development libraries for the following packages installed (minimum versions are listed):

- zlib-1.2.8
- bzip2-1.0.6
- libffi-3.0.11
- glib2-2.38.0
- libxml2-2.9.0
- libarchive-3.1.2
- xz-5.0.4
- libsoup-2.48.1
- sqlite-autoconf-3080002
- intltool-0.35.5
- selinux-2.7

Commands that will make sure most of the development libraries required are installed:

```
# sudo dnf install zlib-devel bzip2-devel libffi-devel glib2-devel libxml2-devel
# sudo dnf install libarchive-devel xz-devel libsoup-devel selinux-devel
# sudo dnf install intltool
```

The last set of development libraries for SQLite require that you install SQLite. Following the 'Install SQLite on Linux' instructions available at https://www.tutorialspoint.com/sqlite/sqlite_installation.htm

Once you have all the development libraries installed, you can clone Restraint from git:

```
% git clone git://git.beaker-project.org/restraint
% cd restraint
```

Build Restraint:

```
% make
```

To build it statically first enter the third-party directory and build the support libraries:

```
% pushd third-party
% make
% popd
```

Then build Restraint with the following command:

```
% pushd src
% PKG_CONFIG_PATH=../third-party/tree/lib/pkgconfig make STATIC=1
% popd
```

Installing Restraint:

```
% make install
```

1.2 Starting the Daemon

Regardless if you installed from RPM or from source you start the daemon one of two ways. If the system uses systemd use the following commands:

```
Enable the service for next reboot
# systemctl enable restraintd.service
Start the service now
# systemctl start restraintd.service
```

For SysV init based systems use the following commands:

```
Enable the service for next reboot
# chkconfig --level 345 restraintd on
Start the service now
# service restraintd start
```

1.3 Commands

1.3.1 restraintd

restraintd is the daemon which executes the tasks.

Both a SysV init script and a systemd unit file are provided. The included spec file will use the correct one when built on RHEL/Fedora based systems.

Logging

All messages from restraintd will be printed to stderr and all output from executing commands will be printed to stdout.

stderr is redirected to /dev/console to help debug when things go wrong. The SysV init script will redirect both stdout + stderr to /var/log/restraintd.log. For systemd you can use the journalctl command:

```
journalctl --unit restraintd

-- Logs begin at Fri 2014-04-11 16:39:13 EDT, end at Fri 2014-04-11 16:46:36 EDT. --
Apr 11 16:40:20 virt-test systemd[1]: Starting The restraint harness....
Apr 11 16:40:20 virt-test systemd[1]: Started The restraint harness..
Apr 11 16:40:20 virt-test restraintd[567]: Waiting for client!
Apr 11 16:40:20 virt-test restraintd[567]: * Fetching recipe: http://beaker.example.
↪com:8000//recipes/1079/
Apr 11 16:40:21 virt-test restraintd[567]: ** (restraintd:567): WARNING **: Ignoring␣
↪Server Running state
Apr 11 16:40:21 virt-test restraintd[567]: * Parsing recipe
Apr 11 16:40:21 virt-test restraintd[567]: * Running recipe
Apr 11 16:40:21 virt-test restraintd[567]: ** Fetching task: 1562 [/mnt/tests/
↪distribution/install]
Apr 11 16:40:27 virt-test restraintd[567]: Resolving Dependencies
Apr 11 16:40:27 virt-test restraintd[567]: --> Running transaction check
Apr 11 16:40:27 virt-test restraintd[567]: ---> Package beaker-distribution-install.
↪noarch 0:1.10-15 will be installed
Apr 11 16:40:30 virt-test restraintd[567]: --> Finished Dependency Resolution
.
.
.
Apr 11 16:40:36 virt-test restraintd[567]: Installed:
Apr 11 16:40:36 virt-test restraintd[567]: beaker-distribution-install.noarch 0:1.10-
↪15
Apr 11 16:40:36 virt-test restraintd[567]: Complete!
Apr 11 16:40:36 virt-test restraintd[567]: ** Parsing metadata
Apr 11 16:40:36 virt-test restraintd[567]: ** Updating env vars
Apr 11 16:40:36 virt-test restraintd[567]: ** Updating watchdog
Apr 11 16:40:37 virt-test restraintd[567]: ** Installing dependencies
Apr 11 16:41:00 virt-test restraintd[567]: Nothing to do
Apr 11 16:41:00 virt-test restraintd[567]: ** Running task: 1562 [/distribution/
↪install]
Apr 11 16:41:00 virt-test restraintd[567]: TASK_RUNNER_PLUGINS: /usr/share/restraint/
↪plugins/task_run.d/10_bash_login
/usr/share/restraint/plugins/task_run.d/15_beakerlib /usr/share/restraint/plugins/
↪task_run.d/20_unconfined make run
Apr 11 16:41:01 virt-test restraintd[567]: -- INFO: selinux enabled: trying to switch␣
↪context...
.
.
.
Apr 11 16:41:12 virt-test restraintd[567]: *** Running Plugin: 98_restore
Apr 11 16:41:12 virt-test restraintd[567]: Nothing to restore.
Apr 11 16:41:12 virt-test restraintd[567]: ** Completed Task : 1562
```

1.3.2 restraint

Used for stand-alone execution.

Use the restraint command to run a job on a remote test machine running restraintd. You can run jobs on the local machine but it is not recommended since some tasks reboot the system. Hosts are tied to recipe IDs inside the job XML.

```
restraint --host 1=addressOfMyTestSystem.example.com:8081 --job /path/to/simple_job.
↪xml
```

Restraint will look for the next available directory to store the results in. In the above example it will see if the directory simple_job.01 exists. If it does (because of a previous run) it will then look for simple_job.02. It will continue to increment the number until it finds a directory that doesn't exist.

By default Restraint will report the start and stop of each task run like this:

```
Using ./simple_job.07 for job run
* Fetching recipe: http://192.168.1.198:8000/recipes/07/
* Parsing recipe
* Running recipe
*  T:   1 [/kernel/performance/fs_mark                    ] Running
*  T:   1 [/kernel/performance/fs_mark                    ] Completed: PASS
*  T:   2 [/kernel/misc/gdb-simple                        ] Running
*  T:   2 [/kernel/misc/gdb-simple                        ] Completed: PASS
*  T:   3 [restraint/vmstat                               ] Running
*  T:   3 [restraint/vmstat                               ] Completed
```

You can pass -v for more verbose output which will show every task reported. If you pass another -v you will get the output from the tasks written to your screen as well.

All of this information is also stored in the job.xml which in this case is stored in the ./simple_job.07 directory.

1.3.3 rstrnt-report-result

Report Pass/Fail/Warn, optional score.

Reporting plugins can be disabled by passing the plugin name to the –disable option. Here is an example of reporting a result but disabling the built in AVC (Access Vector Cache) checker:

```
rstrnt-report-result --disable 10_avc_check $RSTRNT_TASKNAME/sub-result PASS 100
```

Multiple plugins can be disabled by passing in multiple –disable arguments.

To stay compatible with legacy RHTS (Red Hat Test System) tasks, Restraint also looks to see if the environment variable AVC_ERROR is set to +no_avc_check. If this is true then it's the same as the above `--disable 10_avc_check` argument.

1.3.4 rstrnt-report-log

Upload a log or some other file.

1.3.5 rstrnt-reboot

Helper to reboot the system. On UEFI systems it will use efibootmgr to set next boot to what is booted currently.

1.3.6 rstrnt-backup

Helper to backup a config file.

1.3.7 rstrnt-restore

Helper to restore a previously backed up file. There is a plugin which is executed at task completion which will call this command for you.

1.3.8 rstrnt-adjust-watchdog

If you are running in Beaker this allows you to adjust the external watchdog. This does not modify the localwatchdog, so its usually only useful to tasks that have no_localwatchdog set to `true` in their task metadata.

1.3.9 check_beaker

Run from init/systemd, will run a Beaker job.

1.3.10 job2html.xml

An XSLT (eXtensible Stylesheet Language Transformations) template to convert the stand-alone job.xml results file into an HTML doc. The template can be found in Restraint's `client` directory.

Here is an example command to convert a job run XML file into an HTML doc. This HTML doc can be easily navigated with a browser to investigate results and logs.

```
xsltproc job2html.xml simple_job.07/job.xml > simple_job.07/index.html
```

1.3.11 job2junit.xml

An XSLT template to convert the stand-alone job.xml file into JUnit results. The template can be found in Restraint's `client` directory.

Here is an example command to covert a job run XML into JUnit results.

```
xsltproc job2junit.xml simple_job.07/job.xml > simple_job.07/junit.xml
```

1.3.12 Legacy RHTS Commands

If you have the restraint-rhts subpackage installed these commands are provided in order to support legacy tests written for RHTS.

rhts-reboot

Use rstrnt-reboot instead.

rhts-backup

Use rstrnt-backup instead.

rhts-restore

Use rstrnt-restore instead.

rhts-environment.sh

Deprecated.

rhts-lint

Deprecated - only provided so that testinfo.desc can be generated.

rhts-run-simple-test

Deprecated.

1.4 Jobs

Restraint parses a sub-set of the Beaker job XML[1]. Here is an example showing just the elements required for running in the stand-alone configuration.

```
<job>
 <recipeSet>
  <recipe>
   <task name="/kernel/performance/fs_mark" keepchanges="yes">
    <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?master
↪#kernel/performance/fs_mark" />
    <params>
     <param name="foo" value="bar"/>
    </params>
   </task>
      .
      .
      .
   <task name="/kernel/foo/checker">
    <rpm name="rh-tests-kernel-foo-checker"/>
   </task>
  </recipe>
 </recipeSet>
</job>
```

1.4.1 Naming Tasks

For reporting purposes it is a good idea to name your tasks. For git tasks we have settled on a standard where we use the sub-directory path from our git repo as the task name. You can see that in the following example.

```
<task name="/kernel/performance/fs_mark">
```

This name will be used when reporting on the status of the task and when reporting results.

[1] Beaker Job XML.

1.4.2 Task Roles

Restraint supports role assignment for tasks or whole recipes for use in multi-host jobs.

```
<job>
 <recipeSet>
  <recipe role="SERVERS">
   <task name="/kernel/filesystems/nfs/connectathon-mh">
    <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?master
↪#kernel/filesystems/nfs/connectathon-mh" />
   </task>
  </recipe>
  <recipe>
   <task name="/kernel/filesystems/nfs/connectathon-mh" role="CLIENTS">
    <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?master
↪#kernel/filesystems/nfs/connectathon-mh" />
   </task>
  </recipe>
 </recipeSet>
</job>
```

The above example results in environment variables “SERVERS” and “CLIENTS” containing hostnames assigned to corresponding recipes. The variables will be available only to tasks with the same padding within recipes.

Recipe roles function as default roles for tasks that have no role specified and can be overridden by task roles.

Apart from role env variables Restraint also exports 2 more hostname-related variables:

- RECIPE_MEMBERS - contains hostnames of all hosts within current recipeSet.
- JOB_MEMBERS - contains hostnames of all hosts in current job.

1.4.3 Keeping Your Task Changes Intact

By default Restraint will fetch tasks every time you run a recipe overwriting any changes you’ve done locally. This is not desirable in some cases, e.g. when debugging a test. Restraint provides the ability to keep local changes by setting task property “keepchanges” to “yes” in the job xml.

```
<task name="/kernel/performance/fs_mark" keepchanges="yes">
```

1.4.4 Installing Tasks

The above example shows that you can install tasks directly from git or from an RPM in a yum repo.

Fetch

The first example shows fetching a task from git.

```
<fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?master
↪#kernel/performance/fs_mark" />

OR

<fetch ssl_verify="off" url="https://fedorapeople.org/cgit/bpeck/public_git/tests.git/
↪snapshot/tests-master.tar.gz#kernel/performance/fs_mark" />
```

The fetch node accepts git URI's that conform to the following:

- Prefixed with git:// OR use tarballs with http:// and cgit can serve them automatically.
- The fully qualified hostname. Remember that the system running restraintd must be able to reach this host.
- The path to the git repo.
- Optionally you can specify a valid reference which can be a branch, tag or SHA-1. ie: ?master
- Optionally you can specify a sub-dir. Restraint will only extract this sub-dir and run the task from here. ie: #kernel/performance/fs_mark. Notice that there is not a preceding slash here.
- If you need to disable SSL certificate checking you can set ssl_verify parameter to "off".

Restraint uses git's archive protocol to retrieve the contents so make sure your git server has enabled this. You can enable this on most servers by putting the following in your git repo config

```
[daemon]
       uploadarch=true
```

RPM

The second example will attempt to install the task via yum/rpm.

```
<rpm name="rh-tests-kernel-foo-checker"/>
```

Currently Restraint does not attempt to set up any repos that you may have specified in your job.xml. This means that in order for it to install the above task you must have already configured the task repo on the machine running restraintd.

1.4.5 Parameters

You can optionally pass parameters to a task via environment variables. The following snippet from our example would create an environment variable named 'foo' with the value 'bar'.

```
<params>
 <param name="foo" value="bar"/>
</params>
```

The parameter RSTRNT_MAX_TIME allows you to specify a different max time than what is specified in the tasks metadata. Setting KILLTIMEOVERRIDE also has the same effect and is provided for compatibility with legacy RHTS (Red Hat Test System).

The parameter RSTRNT_USE_PTY allows you to either enable or disable using a pty for task execution. Use `true` to enable and `false` to disable. Setting this value in the job will override the settings in metadata or testinfo.desc.

1.5 Tasks

Restraint doesn't require tasks to be written in any particular language. In fact, most tests are written in a mixture of shell, python and C code. You do need to provide some metadata in order for things to work best.

1.5.1 Metadata

Restraint will look for a file called metadata in the task directory. The format for that file is a simple ini file which most people should be familiar with.

```
[General]
name=/restraint/env/metadata
owner=Bill Peck <bpeck@redhat.com>
description=just reports env variables
license=GPLv2
confidential=no
destructive=no

[restraint]
entry_point=./runtest.sh
max_time=5m
dependencies=gcc;emacs
use_pty=false
#use_pty=true # to enable a pty
```

The "General" section is mostly used for informational purposes. The only element that Restraint will read from here is the name attribute. If defined this will over write the task name specified from the job XML.

The "restraint" section has the following elements which can be defined:

entry_point

This tells Restraint how it should start running the task. If you don't specify a program to run it will default to 'make run' which is what legacy RHTS (Red Hat Test System) would do. Other examples of entry points:

- entry_point=autotest-local control-file
- entry_point=STAF local PROCESS START SHELL COMMAND "ps | grep test | wc >testcount.txt"

max_time

The maximum time a task is expected to run. When restraintd runs a task it sets up a localwatchdog which will kill the task after this time has expired. When run in Beaker this is also used for the external watchdog (typically 20-30 minutes later than the local watchdog time). Time units can be specified as follows:

- d for days
- h for hours
- m for minutes
- s for seconds

To set a max run time for 2 days you would use the following:

```
max_time=2d
```

dependencies

A semicolon-delimited (`;`) list of additional packages (needed to run this task) to be installed on the system. The task will abort if the dependencies fail to install.

```
dependencies=lib-virt;httpd;postgresql;nfs-utils;net-tools;net-snmp;ethereal;
↪wireshark;tcpdump;rusers;bzip2;gcc
```

environment

A semicolon-delimited (`;`) list of task environment variables to be set on the system.

```
environment=META_VAR1=var1value;META_VAR2=var2value;META_VAR3=var3value
```

softDependencies

A semicolon-delimited (`;`) list of optional additional packages to be installed on the system. The task will proceed even if the soft dependencies fail to install. This is useful for a task that is intended to run on multiple platforms, and the task can test platform-specific features (e.g., NUMA) if the appropriate support packages are installed, but the task will not abort on the other platforms where the support packages do not exist.

```
softDependencies=numactl;numactl-devel
```

repoRequires

A semicolon-delimited (`;`) list of additional tasks needed for this task to run.

```
repoRequires=general/include;filesystems/include
```

Note: When fetching from git (see *Fetch*), this is the `#subdirectory` portion of the URL, so do *not* use a leading `/` character as was done with RhtsRequires in testinfo.desc for Legacy RHTS tasks.

no_localwatchdog

Normally Restraint will setup a localwatchdog which will attempt to recover from a hung task before the external watchdog (if running under Beaker) triggers. But you can tell Restraint to not setup a localwatchdog monitor by including this key with a value of `true`. Only `true` or `false` are valid values.

```
no_localwatchdog=true
```

use_pty

Before version 0.1.24 Restraint would execute all tasks from a pty. This meant that programs thought they were running in an interactive terminal and might produce ANSI codes for coloring and line positioning. Now the default is not to use a pty which will give much cleaner output. If you find your test is failing because it expects a pty you can enable the old behavior by setting this.

```
use_pty=true
```

OSMajor Specific Options

Any of the above elements can be overridden with OSMajor specific options. In order for this to work the OSMajor (or "OS family") attribute must be filled in the job.xml. If the job was run through Beaker this will have been filled in for you. If you run a stand-alone job (with restraint-client) you can set the value in the family attribute of the recipe tag. For example:

```
<job>
  <recipeSet>
    <recipe family="RedHatEnterpriseLinuxServer5">
      ...
```

For example, if a task is known to take twice as long on RedHatEnterpriseLinuxServer5 then you could use following:

```
max_time=5m
max_time[RedHatEnterpriseLinuxServer5]=10m
```

Another example where we will install RHDB on RedHatEnterpriseLinuxServer5 and PostgreSQL on everything else.

```
dependencies=postgresql
dependencies[RedHatEnterpriseLinuxServer5]=rhdb
```

1.5.2 testinfo.desc

Legacy RHTS tests use this file for their metadata[1]. Restraint supports generating (via the Makefile) and reading this file. But Restraint does not understand all the fields in this file. The following are the ones Restraint parses:

- Name - Same as [General] name
- Environment- Same as [restraint] environment
- TestTime - Same as [restraint] max_time
- Requires - Same as [restraint] dependencies
- RhtsRequires - Same as [restraint] dependencies
- RepoRequires - Same as [restraint] repoRequires
- USE_PTY - Same as [restraint] use_pty

Please see the Beaker documentation for how to populate these fields.

1.6 Variables

The following variables are available to tasks.

- RSTRNT_JOBID - Populated from the job_id attribute of the recipe node.
- RSTRNT_OWNER - Populated from the owner attribute of the job node.
- RSTRNT_RECIPESETID - Populated from the recipe_set_id attribute of the recipe node.
- RSTRNT_RECIPEID - Populated from the id attribute of the recipe node.
- RSTRNT_TASKID - Populated from the id attribute of the task node.

[1] RHTS Task Metadata.

- RSTRNT_OSDISTRO - Name of the distro (only defined if running in Beaker).
- RSTRNT_OSMAJOR - Fedora19 or CentOS5.
- RSTRNT_OSVARIANT - Server, Client, not all distros use variants.
- RSTRNT_OSARCH
- RSTRNT_TASKNAME - Name of task "/distribution/install".
- RSTRNT_TASKPATH - Where the task is installed.
- RSTRNT_MAXTIME - Max time in seconds for this task to complete.
- RSTRNT_REBOOTCOUNT - The number of times the system has rebooted for this task.
- RSTRNT_TASKORDER

These variables are provided in order to support legacy tests written for RHTS (Red Hat Test System).

- JOBID - use RSTRNT_JOBID instead.
- SUBMITTER - use RSTRNT_OWNER instead.
- RECIPESETID - use RSTRNT_RECIPESETID instead.
- RECIPEID - use RSTRNT_RECIPEID instead.
- RECIPETESTID - use RSTRNT_RECIPEID instead.
- TESTID - Use RSTRNT_TASKID instead.
- TASKID - use RSTRNT_TASKID instead.
- REBOOTCOUNT - use RSTRNT_REBOOTCOUNT instead.
- DISTRO - Use RSTRNT_OSDISTRO instead.
- VARIANT - Use RSTRNT_OSVARIANT instead.
- FAMILY - Use RSTRNT_OSMAJOR instead.
- ARCH - Use RSTRNT_OSARCH instead.
- TESTNAME - Use RSTRNT_TASKNAME instead.
- TESTPATH - Use RSTRNT_TASKPATH instead.
- RESULT_SERVER - There is no equivalent, communication is only with the Beaker lab controller.

1.7 Plugins

Restraint relies on plugins to execute tasks in the correct environment and to check for common errors or simply to provide additional logs for debugging issues. Here is a typical outline of how plugins are executed:

```
run_task_plugins
 \
  10_bash_login
  |
  15_beakerlib
  |
  20_unconfined
  |
  25_environment
```

(continues on next page)

(continued from previous page)

```
  |
  make run
  |\
  | report_result
   \
    report_result

run_task_plugins
 \
  10_bash_login
  |
  15_beakerlib
  |
  20_unconfined
  |
  25_environment
  |
  run_plugins <- completed.d
  \
   98_restore
```

The report_result commands above cause the following plugins to be executed:

```
run_task_plugins
 \
  05_linger
  |
  10_bash_login
  |
  15_beakerlib
  |
  20_unconfined
  |
  25_environment
  |
  30_restore_events
  |
  35_oom_adj
  |
  run_plugins <- report_result.d
   \
    01_dmesg_check
    |
    10_avc_check
    |
    20_avc_clear
```

These plugins do not run from the task under test. They run from restraintd process. This allows for greater flexibility if your task is running as a non-root user since a non-root user would not be able to inspect some logs and wouldn't be able to clear dmesg log.

1.7.1 Task Run

Task run plugins are used to modify the environment under which the tasks will execute. Simply place the executable in /usr/share/restraint/task_run.d. The list of files in this directory will be passed to exec in alphabetical order.

Restraint currently ships with two task run plugins:

- 05_linger - Enables session bus for user that Restraint is running as. You can disable this with RSTRNT_DISABLE_LINGER=1
- 10_bash_login - invoke a login shell.
- 15_beakerlib - Sets env vars to tell beakerlib how to report results in Restraint.
- 20_unconfined - if selinux is enabled on system run task in unconfined context.
- 25_environment - Will attempt to guess certain variables if they weren't defined, (OSARCH, OSMAJOR, etc..).
- 30_restore_events - Restores Multi-host states after a reboot.
- 35_oom_adj - sets the oom score low so we are less likely to be killed.

So the above plugins would get called like so:

```
exec 05_linger 10_bash_login 15_beakerlib 20_unconfined 25_environment 30_restore_
↪events 35_oom_adj "$@"
```

In order for this to work the task run plugins are required to exec "$@" at the end of the script. Although task run plugins can't take any arguments they can make decisions based on environment variables.

It should be pointed out that the task run plugins are executed for all other plugins! This is to ensure plugins run with the same environment as your task. When executed under all other plugins the following variable will be defined:

```
RSTRNT_NOPLUGINS=1
```

You can do conditionals based on this so lets create a plugin which will start a TCP capture:

```
# Capture tcpdump data from every task
cat << "EOF" > /usr/share/restraint/plugins/task_run.d/30_tcpdump
#!/bin/sh -x

echo "*** Running PLugin: $0"

# Don't run from PLUGINS
if [ -z "$RSTRNT_NOPLUGINS" ]; then
  tcpdump -q -i any -q -w $RUNPATH/tcpdump.cap 2>&1 &
  echo $! > $RUNPATH/tcpdump.pid
fi

exec "$@"
EOF
chmod a+x /usr/share/restraint/plugins/task_run.d/30_tcpdump
```

1.7.2 Report Result

Every time a task reports a result to Restraint these plugins will execute.

- 01_dmesg_check - This plugin checks dmesg output for the following failure strings.

```
Oops|BUG|NMI appears to be stuck|Badness at
```

But then it runs any matches through an inverted grep which removes the following:

```
BIOS BUG|DEBUG|mapping multiple BARs.*IBM System X3250 M4
```

This is an effort to reduce false positives. Both of the above strings can be overridden from each task by passing in your own FAILURESTRINGS or FALSESTRINGS variables.

In some cases the kernel will produce a multi-line error message (including hardware information and stack trace) in the dmesg output which is delimited by a "cut here" line at the beginning and an "end trace" line at the end. This plugin will capture the entire contents of the multi-line trace and considers it as a single failure. The FALSESTRINGS pattern is applied to the whole trace to check for false positives.

- 10_avc_check - This plugin searches for AVC (Access Vector Cache) errors that have occurred since the last time a result was reported.
- 20_avc_clear - This moves the time stamp used by avc_check forward so that we don't see the same AVC's reported again, some tests might generate AVC's on purpose and disable the check but you will still want to move the time stamp forward.

1.7.3 Local Watchdog

These plugins will only be executed if the task runs beyond its expected time limit. Restraint currently ships with three plugins:

- 10_localwatchdog - uploads the resultoutputfile.log of the running task.
- 20_sysinfo - Collects and uploads system information.
  - Uploads system log which contains a collection of system information such as slabinfo, list of blocked tasks derived from `sysrq m, t and w`, and pre-existing system log messages. Depending if journalctl exists, file journalctl or /var/log/messages is uploaded.
  - Uploads ps-lwd.log which contains a verbose list of running processes.
  - Uploads dmesg log if it contains any output.
  - Uploads user logs listed in $TESTPATH/logs2get.
- 99_reboot - Simply reboots the system to try and get the system back to a sane state.

1.7.4 Completed

These plugins will get executed at the end of every task, regardless if the localwatchdog triggered or not. The only plugin currently shipped with Restraint is:

- 98_restore - any files backed up by either rhts-backup or rstrnt-backup will be restored.

To finish our tcpdump example from above we can add the following:

```
#Kill tcpdump and upload
cat << "EOF" > /usr/share/restraint/plugins/completed.d/80_upload_tcpdump
#!/bin/sh -x

kill $(cat $RUNPATH/tcpdump.pid)
rstrnt-report-log -l $RUNPATH/tcpdump.cap
EOF
chmod a+x /usr/share/restraint/plugins/completed.d/80_upload_tcpdump
```

1.8 Using Restraint

1.8.1 Running in Beaker

Beaker will use restraint by default if you are running Red Hat Enterprise Linux version 8 or later or if you are running Fedora version 29 or later.

To use Restraint in Beaker for earlier versions of Red Hat Enterprise Linux or Fedora, you will need to specify 'restraint' as the harness:

```
<recipe ks_meta="harness=restraint">
<repos>
 <repo name="restraint"
       url="https://beaker-project.org/yum/harness/Fedora29/"/>
</repos>
 .
 .
 .
</recipe>
```

If you have tasks/tests that were written for legacy RHTS (Red Hat Test System) you can install the restraint-rhts sub-package which will bring in the legacy commands so that your tests will execute properly. Some tasks/tests have also been written with beakerlib. Here is an example recipe node that will install both for you:

```
<recipe ks_meta="harness='restraint-rhts beakerlib'">
.
.
.
</recipe>
```

If you are using Beaker command line workflows use these command line options:

```
bkr <WORKFLOW> --ks-meta="harness=restraint" --repo https://beaker-project.org/yum/
↪harness/Fedora29/
```

If you need RHTS compatibility and/or beakerlib you can add it here as well:

```
bkr <WORKFLOW> --ks-meta="harness='restraint-rhts beakerlib'" --repo https://beaker-
↪project.org/yum/harness/Fedora29/
```

1.8.2 Running Standalone

Restraint can run on its own without Beaker, this is handy when you are developing a test and would like quicker turn around time. Before Restraint you either ran the test locally and hoped it would act the same when run inside Beaker or dealt with the slow turn around of waiting for Beaker to schedule, provision and finally run your test. This is less then ideal when you are actively developing a test.

You still need a job XML file which tells Restraint what tasks should be run. Here is an example where we run three tests directly from git:

```
<?xml version="1.0"?>
<job>
  <recipeSet>
    <recipe id="1">
      <task name="/kernel/performance/fs_mark">
```

(continues on next page)

(continued from previous page)

```
        <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?
↪master#kernel/performance/fs_mark"/>
      </task>
      <task name="/kernel/misc/gdb-simple">
        <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?
↪master#kernel/misc/gdb-simple"/>
      </task>
      <task name="/kernel/standards/usex" role="None">
       <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git
↪#kernel/standards/usex"/>
      </task>
    </recipe>
  </recipeSet>
</job>
```

Tell Restraint to run a job:

```
% restraint --job /path/to/job.xml
```

You probably don't want to run restraintd on the machine you use for day to day activity. Some tests can be destructive or just make unfriendly changes to your system. Restraint allows you to run tasks on a remote system. This means you can have the task git repo on your development workstation and verify the results on your test system. In order for this to work your git repo and the recipe XML need to be accessible to your test system. Be sure to have the restraint-client package installed on the machine you will be running the restraint command from

Here is an example:

```
% restraint --host 1=addressOfMyTestSystem.example.com:8081 --job /path/to/job.xml
```

This will connect to restraintd running on addressOfMyTestSystem.example.com and tell it to run the recipe with id="1" from this machine. Also remember that the tasks which are referenced inside of the recipe need to be accessible a well. Here is the output:

```
restraint --host 1=addressOfRemoteSystem:8081 --job simple_job.xml -v
Using ./simple_job.07 for job run
* Fetching recipe: http://192.168.1.198:8000/recipes/07/
* Parsing recipe
* Running recipe
*  T:   1 [/kernel/performance/fs_mark                    ] Running
**      1 [Default                                         ] PASS
**      2 [Random                                          ] PASS
**      3 [MultiDir                                        ] PASS
**      4 [Random_MultiDir                                 ] PASS
*  T:   1 [/kernel/performance/fs_mark                    ] Completed: PASS
*  T:   2 [/kernel/misc/gdb-simple                        ] Running
**      5 [/kernel/misc/gdb-simple                         ] PASS Score: 0
*  T:   2 [/kernel/misc/gdb-simple                        ] Completed: PASS
*  T:   3 [/kernel/standards/usex                         ] Running
**  :   6 [/kernel/standards/usex                         ] PASS
*  T:   3 [/kernel/standards/usex                         ] Completed: PASS
```

All results will be stored in the job run directory which is 'simple_job.07' for this run. In this directory you will find 'job.xml' which has all the results and references to all the task logs. You can convert this into HTML with the following command:

```
% xsltproc job2html.xml simple_job.07/job.xml >simple_job.07/index.html
```

`job2html.xml` is found in Restraint's `client` directory.

1.8.3 Running in Beaker and Standalone

Sometimes the tests that I am developing can be destructive to the system so I don't want to run them on my development box. Or the test is specific to an architecture so I can't use a VM for it on my machine. These are cases where it's really handy to use a combination of Beaker for provisioning and Standalone for executing the tests.

First step is to run the following workflow to reserve a system in Beaker:

```
<job><whiteboard>restraint reservesys</whiteboard>
 <recipeSet>
  <recipe ks_meta="harness=restraint" id="1">
   <distroRequires>
    <and>
     <distro_name op="=" value="Fedora-20"/>
     <distro_arch op="=" value="x86_64"/>
    </and>
   </distroRequires>
   <hostRequires/>
   <repos>
    <repo name="myrepo_0" url="http://copr-be.cloud.fedoraproject.org/results/bpeck/
↪restraint/fedora-20-x86_64"/>
   </repos>
   <task name="/distribution/install" role="STANDALONE" />
   <task name="/distribution/reservesys" role="None">
    <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git
↪#distribution/reservesys"/>
   </task>
  </recipe>
 </recipeSet>
</job>
```

This will reserve a ppc64 system running Fedora20. The /distribution/reservesys task will email the submitter of the job when run so you know the system is available. By default the reservesys task will give you access to the system for 24 hours, after that the external watchdog will reclaim the system. You can extend it using extendtesttime.sh on the system. Finally It will also run a second instance of restraintd on port 8082 which you can then connect to with the Restraint client running on your developer machine.:

```
% restraint --host 1=FQDN.example.com:8082 --job simple_job.xml
```

If the task you are developing doesn't work as expected you can make changes and try again. Just remember to push your changes to git, the system under test will pull from the git URL you put in your job XML.

1.9 Release Notes

1.9.1 Restraint 0.1.39

Released 27 February 2019.

- NEW: RHBZ#1552199: Restraint-client now supports changing timeout value for the request.
  (Contributed by Martin Styk)
- FIXED: RHBZ#1670377: Fixed compilation issues for GCC9/Automake.
  (Contributed by Martin Styk)

1.9.2 Restraint 0.1.38

Released 29 January 2019.

- FIXED: RHBZ#1670111: Fixed crash of Restraint for ppc64le and aarch64 architecture.
  (Contributed by Bill Peck)

1.9.3 Restraint 0.1.37

Released 11 January 2019.

- NEW: RHBZ#1665390: Added feature to set family from client XML.
  (Contributed by Bill Peck)
- NEW: RHBZ#1656466: Restraint now supports `@module` syntax for dependencies for RHEL8+.
  (Contributed by Martin Styk)
- FIXED: RHBZ#1663125: Restraint now listens separately for IPv4 and IPv6. One running version of the protocol is sufficient for `restraintd` run.
  (Contributed by Bill Peck)
- FIXED: RHBZ#1663825: When BootCurrent is not available, Restraint will try to fall back to `/root/EFI_BOOT_ENTRY.TXT`.
  (Contributed by Martin Styk)
- FIXED: RHBZ#1659353: Fixed obsolete URL for Bzip2 package in Makefile.
  (Contributed by Martin Styk)
- FIXED: RHBZ#1599550: Fixed crash of Restraint for RHEL6 arch s390 caused by glib2.
  (Contributed by Matt Tyson)
- FIXED: RHBZ#1608262: Fixed guest-host synchronization.
  (Contributed by Dan Callaghan)

1.9.4 Restraint 0.1.36

Released 24 August 2018.

- NEW: RHBZ#1506064: The dmesg error checking plugin can now match patterns against multi-line "cut here" style traces. The plugin now ignores a warning about "mapping multiple BARs" on IBM x3250m4 systems, matching the existing behaviour of the RHTS dmesg checker.
  (Contributed by Jacob McKenzie)
- FIXED: RHBZ#1592376: Restraint resets the SIGPIPE handler before executing task processes. Previously the tasks would inherit the "ignore" action for SIGPIPE from the Restraint parent process, which would prevent normal shell broken pipe handling from working correctly in the task.
  (Contributed by Matt Tyson)
- FIXED: RHBZ#1595167: When the local watchdog timer expires, Restraint will now upload the output from **`journalctl`** in favour of `/var/log/messages` if the systemd journal is present. Previously it would attempt to upload `/var/log/messages` even if the file did not exist, causing the local watchdog handling to enter an infinite loop.
  (Contributed by Matt Tyson)
- FIXED: RHBZ#1593595: Fixed an improper buffer allocation which could cause Restraint to crash with a segmentation fault instead of reporting an error message in certain circumstances.

(Contributed by Róman Joost)

- FIXED: RHBZ#1600825: Fixed a file conflict introduced in Restraint 0.1.35 between the `restraint` package and the `rhts-test-env` package.
  (Contributed by Matt Tyson)
- FIXED: RHBZ#1601705: Fixed a shell syntax error in the RPM %post scriptlet on RHEL4 which caused the package to be un-installable.
  (Contributed by Dan Callaghan)
- FIXED: RHBZ#1585904: Fixed a shell syntax error in the restraintd init script which caused it to fail to start on RHEL4.
  (Contributed by Dan Callaghan)

1.10 Developer Guide

If you have questions related to Restraint's development that are not currently answered in this guide, the two main ways to contact the Restraint development team are the same as those for getting general assistance with using and installing Beaker:

- the development mailing list
- the #beaker IRC channel on FreeNode

This document focuses on the mechanics of working with Restraint's code base with the target audience being a Restraint user interested in learning more about Restraint's working or a potential Restraint contributor.

1.10.1 Getting Started

Restraint is written in C. The source lives in a git repo on http://git.beaker-project.org along with other related projects. Let's create a local clone of the Restraint source:

```
$ git clone git://git.beaker-project.org/restraint
```

Restraint uses a number of external libraries/tools, so before you can build Restraint you need to install the external libraries using `yum-builddep restraint.spec`. Once you have installed these dependencies, running a `make` at the source directory root will compile and build Restraint. To also run a quick sanity check, it is a good idea to run the unit tests using `make check`. The unit tests use a simple Python HTTP server and `git-daemon`, so you will need to install this as well (`yum -y install git-daemon`).

1.10.2 Testing Changes

If you have fixed an existing bug or implemented a new feature, it is a good idea to add a relevant test. The existing tests can be found in the `src/` directory in the source files with names starting with `test_`.

It may also be a good idea to run a recipe by building the Restraint daemon and client from the modified code base. Once you have built the binaries using `make`, run `make install` to install the Restraint daemon, client and other bits. Start the Restraint daemon in one terminal (as `root` user):

```
# restraintd
Waiting for client!
```

From another terminal, use the Restraint client to execute a recipe:

```
# restraint --host 1=127.0.0.1:8081 --job path/to/recipe
```

More details can be found in *Running Standalone*.

1.10.3 Submitting a Patch

Patches must first be submitted for review to the Beaker project's `gerrit` code review installation. It may be convenient to setup a git remote as follows:

```
[remote "restraint-gerrit"]
url=git+ssh://gerrit.beaker-project.org:29418/restraint
```

Once you're happy with the change and the test you have written for it, push your local branch to Gerrit for review:

```
git push restraint-gerrit myfeature:refs/for/master
```

1.11 ToDo

- Nothing currently - File an RFE

CHAPTER 2

Additional Information

CHAPTER 3

Indices and Tables

- genindex
- modindex
- search