Beah

Release dev

Aug 28, 2018

Contents

Beah is the default test harness used in Beaker. See Alternative Harness Guide if you are interesed in using a different harness and Related Projects for an alternative test harness implementation, *restraint*.

CHAPTER 1

Glossary

Controller It is a center piece of harness. It is used to process Commands from Backends, spawn Tasks and process Events from Tasks, and eventually forwarding these to Backends.

Backend It is a client which connects to Controller and which can issue Commands (read from console or socket) and is processing Events (writing to console or to file, to internet socket etc.) Multiple Backends can be connected to a single Controller.

Task It is an executable, which runs a test and is generating Events as a result, sending these to the Controller server. It can use either stdout or socket to send events to Controller. stderr is captured as well, but is considered a raw-data (creating lose_item events.)

Event It is a piece of information generated by running Task (e.g. log-event, result-event) or Controller (e.g. pong-event, bye-event) and sent to Backend.

Command It is a piece of information instructing Controller (and eventually Task) to perform an operation (e.g. run-command to spawn a new Task or kill-command killing a Controller server)

Test It is an executable performing testing producing output in known format. Task adaptor has to be written to translate this output to sequence of Events.

CHAPTER 2

Administrator Guide

2.1 Installing and Upgrading Beah

New Beah releases are made available as RPM packages for Red Hat Enterprise Linux and Fedora via a yum repository here.

If you are upgrading an existing Beah installation, you can simply run `beaker-repo-update` on the Beaker server. To specify an alternative location, use the `-b` switch. For example:

```
beaker-repo-update -b http://beaker-project.org/yum/harness-testing/
```

2.2 Using Beah for IPv6 testing

New in version 0.7.0.

During a test run, periodic network communication over TCP/IP takes place from a Beah daemon on the test system to the lab controller and between Beah services on the test system itself. The following are necessary prerequisites for Beah to be able to function successfully when IPv6 functionality is desired (in a dual IPv4/IPv6 environment) or IPv4 is disabled on the test system to test IPv6 specific functionality (See *Limitations* below).

New in version 0.7.4.

It is possible to ask Beah to use IPv4 exclusively for all it's network communication even when IPv6 connectivity may be possible by specifying `beah_no_ipv6` in the recipe's `ksmeta` variable (see Install options).

2.2.1 Test system environment

- The operating system must support IPv6.
- The network interfaces are appropriately configured (IPv6 address assigned).
- Routing tables are correctly setup for IPv6.

- The version of the Twisted library must be greater than or equal to `12.1`.

Note: In the absence of any of the above, communication within the test system falls back to using IPv4.

2.2.2 Lab controller

- The IPv6 DNS records must be configured correctly.
- The firewall configuration must be correctly configured to allow connections to the `beaker-proxy` service that runs on port `8000` over IPv6.

Note: In the absence of any of the above, communication with the lab controller falls back to using IPv4.

2.2.3 Limitations

The following limitations exist with regards to using Beah for IPv6 testing:

- Multihost testing is currently not supported when the test systems have IPv4 disabled.
- Beah fetches every task from the Beaker server's task library just before it starts executing it. When IPv4 is disabled, this is not possible, unless `/etc/resolv.conf` on the test system has the IPv6 addresses of the nameservers so that it can successfully communicate over IPv6 with the Beaker server. Of course, the server has to be reachable over IPv6 (IPv6 enabled, DNS records updated and firewall rules appropriately configured).

  One possible workaround is to manually add entries in the `/etc/hosts` file on the test system for the Beaker server (to fetch the Task RPMs) and any other host with which communication may be needed (for example, for downloading packages from a remote yum repository). Here is a sample `<ksappends/>` snippet which can be added to a Beaker Job XML and will setup `/etc/hosts` with IPv6 address and hostname mapping for the beaker server `beaker-server.host.com`:

```
<ks_appends>
<ks_append><![CDATA[
%post
cat >>/etc/hosts <<EOF
2620:52:0:1065:5054:ff:fe22:b7d9 beaker-server.host.com
EOF
%end
]]></ks_append>
</ks_appends>
```

  In the absence of both the above, the recipe will finish without being able to execute the remaining tasks.

CHAPTER 3

Developer guide

Beah is using TCP/IP sockets for IPC. In case of failures it is easy to reconnect. (At least easier than simple capturing stdout.)

Events and Commands are JSON-serialized new-line separated messages sent over TCP/IP socket. Easy to extend, easy to ignore parts of message which are not understood. Well supported in many programming languages. Rather effective enconding/decoding.

Single JSON object on a line, is quite robust: in case of message corruption only that message (and eventually next one) will be affected. In case of events, these will not be lost - the lose_item event containing raw data is generated.

Twisted framework is used for handling non-blocking I/O operations.

3.1 Modules

The source code is available in a git repository here.

(Beware: things are not in place.)

beah.config Default configuration. Update this module from outside

beah.core Independent components - constants, controller, interfaces, basic backends and tasks...

beah.backends, beah.tasks Additional backend and task adaptors

beah.wires Wiring to glue things together

beah.wires.twisted Twisted wiring (mostly metadata - use this protocol, this implementation, JSON over TCP/IP socket, default cfg,...

beah.wires.internals Internal implementation for wirings

beah.misc Things which do not fit elsewhere

beah.filters I/O filters. LineReceiver, ObjReceiver,...

beah.tests Testing harness

beahlib Library to be used from python tasks/tests.

3.2 Beah services and their interaction

During a test run, several Beah components interact over TCP/IP within the system itself and with the Beaker lab controller.

3.2.1 TCP/IP Server processes

When you login to a test system (say, when the /distribution/reservesys is running), you will see that the following Beah specific servers listening:

```
beah-srv       9714    root   10u  IPv6  26970      0t0  TCP *:12432 (LISTEN)
beah-srv       9714    root   12u  IPv6  26972      0t0  TCP *:12434 (LISTEN)
beah-rhts-task 10142   root    7u  IPv6  27966      0t0  TCP localhost:7089 (LISTEN)
```

The `beah-srv` process corresponds to the server started by `start_server()` in `beah/wires/internals/twserver.py` and it basically starts the `TaskListener` and `BackendListener`, whose presence you can usually see in the console logs:

```
2014-03-31 21:58:19,384 beah start_server: INFO Controller: BackendListener listening␣
↪on :: port 12432
2014-03-31 21:58:19,385 beah start_server: INFO Controller: BackendListener listening␣
↪on /var/beah/backend12432.socket
2014-03-31 21:58:19,386 beah start_server: INFO Controller: TaskListener listening on␣
↪:: port 12434
2014-03-31 21:58:19,387 beah start_server: INFO Controller: TaskListener listening on␣
↪/var/beah/task12434.socket
```

These servers exist throughout a recipe run on the test system. The corresponding "client" programs live in `beah/wires/internals/twbackend.py` and `beah/wires/internals/twtask.py`.

The `beah-rhts-task` process (`main()` function in `beah/tasks/rhts_xmlrpc.py`) corresponds to the result server started *per task* by `beah-srv` and exits on a task completion.

Note that on a distro which doesn't have the right twisted library or the IPv6 support has been disabled otherwise, the servers will only listen on IPv4 interfaces (see *Using Beah for IPv6 testing* to learn more about the IPv6 testing support in Beah).

3.2.2 System services

The following beah daemons are started at system boot:

`beah-fwd-backend`: This handles the communication during multi host jobs. The corresponding source file is `beah/beaker/backends/forwarder.py`.

`beah-beaker-backend`: This talks to the Beaker lab controller's `beaker-proxy` process over XML-RPC. The corresponding source file is `beah/beaker/backends/beakerlc.py`.

`beah-srv`: This is the main daemon process we saw above. The corresponding source file is `beah/bin/srv.py`.

3.3 Setting up a development environment

To set-up development environment source dev-env.sh. Type `. dev-env.sh` in BASH, which will set required environment variables (PATH and PYTHONPATH). This is not required when package is installed.

After setup, run:

```
launcher a
```

in the same shell, which will start server and backends in separate terminals. Or launch components yourself.

Development environment provides these shell functions:

- beah-srv - controller server
- beah-cmd-backend - backend to issue commands to controller. Enter `help` when "beah>" prompt is displayed.
- beah-out-backend - backend to display messages from controller
- beah - command line tool. Use `beah help` to display help. This uses the same command set as beah-cmd-backend
- launcher - wrapper to start these programms in new terminal windows.

beah-out-backend, beah-cmd-backend and beah will wait for controller.

Few auxiliary binaries are provided in bin directory:

- mtail_srv - run srv and beah-out-backend in single window (using multitail tool.)
- **beat_tap_filter - a filter taking a Perl's Test::Harness::TAP format on** stdin and producing stream of Events on stdout.

There are few test tasks in examples/tasks directory:

- a_task - a very simple task in python.
- a_task.sh - the same, in bash, with some delays introduced.
- env - a binary displaying environment variables of interest.
- flood - flooding Controller with messges. This task will not finish and has to be killed (in a `pkill flood` manner.)
- socket - a task using TCP/IP socket to talk to Controller.

Actually a_task and a_task.sh are a simple demonstration of how the test might look like, though it is not definite and more comfortable API will be provided.

In default configuration server is listenning on localhost:12432 for backends and localhost:12434 for tasks. On POSIX compatible systems unix domain sockets are used for local connections by default.

beah-cmd-backend does not offer history or command line editing features (it is on TODO list) thus it is more convenient to use beah command line tool.

The commands supported are:

- ping [MESSAGE]: ping a controller, response is sent to issuer only.
- PING [MESSAGE]: ping a controller, response is broadcasted to all backends.
- run TASK (r TASK): run a task. TASK must be an executable file.
- kill: kill a controller.
- dump: instruct controller to print a diagnostics message on stdout.

- quit (q): close this backend.
- help(h): print this help message.

Controller's log is written to `[/tmp]/var/log/beah.log`.

3.4 Development and usage in a lab

The `lm-install.sh` script can be used to install harness from working copy on a lab machine. This requires either LABM env.variable to be defined or passing lab machine's FQDN as an argument

To change settings, change `lm-install-env.sh` file. As this file is tracked by VCS, if `lm-install-env.sh.tmp` exists in current directory it is used with higher priority.

3.4.1 Usage

On a lab machine:

```
$ mkdir -p /mnt/testarea/lm-install
```

This is the default. Change `LM_INSTALL_ROOT` in lm-install-env.sh.

On the machine where beaker/Harness tree exists:

```
edit lm-install-env.sh (or eventually lm-install-env.sh.tmp) file.
$ export LABM=x.ample.com
$ ./lm-install.sh
$ 'LABM=x.ample.com ./lm-install.sh'
```

Or, the following can be used instead of the last two steps:

```
$ './lm-install.sh x.ample.com'
```

On a lab machine:

```
$ cd /mnt/testarea/lm-install
$ . lm-package-*.sh
```

Be careful to choose the correct one to be used.

`. /mnt/testarea/lm-install/main.sh` can be used anytime to read environment and load functions. Run lm_main_help and lm_help for more help on available functions.

3.5 Writing a patch for Beah

Here is a brief overview of how you can submit a patch for Beah.

3.5.1 Clone Beah's repository

Clone beah: `git clone git://git.beaker-project.org/beah`

3.5.2 Create a local working branch

Create a branch (say, `myfeature`): `git checkout origin/develop -b myfeature`. Make your changes and once you are happy, commit the changes. If your patch fixes a bug, please include the Red Hat Bugzilla number as a footer line in your commit message. For example:

```
This commit fixes a minor glitch in how Beah handles
errors.

Bug: 134511
```

3.5.3 Submitting your patch

Beah and all other projects maintained as part of Beaker uses the Gerrit code review tool to manage patches. Push your local branch to the Beaker project's Gerrit instance for review:

```
git push git+ssh://gerrit.beaker-project.org:29418/beah  myfeature:refs/for/develop
```

CHAPTER 4

Releases

4.1 Beah-0.7.7

Changelog

- Beah systems units now use Wants= instead of Requires= for dependencies, so that they can be restarted independently of each other.
- The `beah` SELinux policy module is now built and installed on RHEL7.

4.2 Beah-0.7.6

Changelog

- Add before and conflicts on shutdown.target for beah systemd services. This will allow the Beah services to be shutdown cleanly.
- Currently Beah considers a Task runner exit as task completion. Starting this release, Beah will ignore a task exit when a system reboots via `rhts-reboot` and hence not mark it as "done". This fixes #908354.

4.3 Beah-0.7.5

Changelog

- Pass a valid Exception to errback()
- fix systemd dependencies for beah-srv.service
- don't rely on HOSTNAME env var
- SELinux policy module to allow beah to transition to unconfined
- Discard python-hashlib to enable FIPS mode on RHEL5

4.4 Beah-0.7.4

Changelog

- A new config option `IPV6_DISABLED` will cause Beah to avoid using IPv6 even when it is available.
- Beah now starts after systemd readahead collection is finished.

4.5 Beah-0.7.3

Changelog

- Backend needs to listen on all interfaces, not just loopback. This fixes a regression in Beah 0.7.2 where multi-host testing did not work because the other Beah processes in the recipe set were not reachable over the network. (Contributed by Dan Callaghan in #1067745.)

4.6 Beah-0.7.2

Changelog

- Brown paper bag release: fixed a typo in `start_task`, found by pylint.

4.7 Beah-0.7.1

Changelog

- Fixed missing conversion for RHTS_PORT, which was causing TypeError when the RHTS_PORT task parameter was set. (Contributed by Marian Csontos in #1063815.)
- Handles any combination of IPv6 and IPv4 being enabled, including absent IPv4 loopback address. (Contributed by Dan Callaghan in #1059479 and Amit Saha in #1062896.)

4.8 Beah-0.7.0

Changelog

- IPv6 support
- Remove dependency on 'python-simplejson' on RHEL 6+, Fedora

4.9 Beah-0.6.48-1

Changelog

- Add a release note generator
- ControlGroup configuration option no longer valid.
- pass exception instance instead of string to Failure

4.10 Beah-0.6.47-1

Changelog

- Changes to Documentation
- Add a version string.
- Add a new README and remove build.sh
- Documentation reorganization
- Add an error handler to simple_recipe
- fix RPM conditional on RHEL3 and RHEL4

CHAPTER 5

Discussions

The best way to interact with Beaker and the Beah developers and users is in the #beaker IRC channel on irc.freenode.net. The Beaker developers monitor this channel, and development discussions often happen there. Alternatively, you can post your question to the beaker-devel mailing list.

CHAPTER 6

Reporting a bug

If you've found a bug in Beah, please report it in Red Hat Bugzilla against the Beaker product using the test harness component.

Index