kamailio-tests

A testing framework for Kamailio

Giacomo Vacca - Kamailio World 2018 Workshop

Why this Workshop

kamailio-tests aims to provide reduce the need for end-to-end tests. “Test units” target a specific core or module functionality.

It encourages contributors to the Kamailio project to build unit tests whenever a new feature is published, or a bug fixed.

It’s an open source project published by Daniel-Constantin Mierla, with small contributions from the presenter of this workshop.

About me

I design, build, maintain RTC platforms based on Open Source components.

User/Fan/Supporter of Kamailio for more than a decade now.

Truphone, Libon, Nexmo and more

I live in Sardinia.
(FAQ: No, that's Sicily; Sardinia is the one more West.)

Giacomo Vacca
@giavac

Agenda

The rationale behind this project

Project structure

The role of Docker

The current test scenarios

Future development

Q&A

https://github.com/kamailio/kamailio-tests

“Why do we always end up debugging?”

- Anonymous from the audience

Why kamailio-tests

https://github.com/kamailio/kamailio-tests

A “self-contained” environment to test:

- Modules
- Routing scripts
- DB access
- Build requirements (indirectly)

No need to setup permanent test environments

Allow for (automated) regression testing

Reduce the need for end-to-end testing

“Adding a feature or fixing a bug? Also add a test for it.”

Structure of test units

A script, *units/UNIT_NAME/UNIT_NAME.sh*

Optionally one or more custom routing scripts, *units/UNIT_NAME/kamailio-NAME.cfg*

A README, *units/UNIT_NAME/README.md*

Name of directory == Name of the unit

Unit name format: txxxxxnnnn

- xxxxx: 5 lowercase chars representing the test “group”
- nnnn: 4 digits to enumerate the unit in the group
- Example: **tcfgxx0001**

Structure of a test script

(Optionally) Configure the DB (add a subscriber, add an ACL, etc)

Run Kamailio with a given configuration (direct std out to a log file)

Trigger the test, by:

- Just starting Kamailio
- Creating a SIP INVITE (with or w/o auth)
- Creating a SIP OPTIONS to trigger a route

Stop Kamailio

Check the log for the expected items

Determine the test result (typically by grepping the log file)

```
#!/bin/bash

. ../../etc/config
. ../../libs/utils

echo "--- start kamailio -f ./kamailio-tasync0001.cfg"
${KAMBIN} -P ${KAMPID} -w ${KAMRUN} -Y ${KAMRUN} -f ./kamailio-tasync0001.cfg -a no -ddd -E 2>&1 | tee /tmp/kamailio-tasync0001.log &
ret=$?
sleep 1
sipsak -s sip:test@127.0.0.1
sleep 1
kill_pidfile ${KAMPID}
sleep 1
echo
echo "--- grep output"
echo
grep "request_route: var_loc is 1" /tmp/kamailio-tasync0001.log
ret=$?
if [ ! "$ret" -eq 0 ] ; then
    exit 1
fi
grep "request_route: var_inc is 1" /tmp/kamailio-tasync0001.log
ret=$?
if [ ! "$ret" -eq 0 ] ; then
    exit 1
fi
grep "async_route: var_loc is 1" /tmp/kamailio-tasync0001.log
ret=$?
if [ ! "$ret" -eq 0 ] ; then
    exit 1
fi
grep "async_route: var_inc is 1" /tmp/kamailio-tasync0001.log
ret=$?
if [ ! "$ret" -eq 0 ] ; then
    exit 1
fi
exit 0
```

Utilities and ktestsctl

etc/config: shell script with configuration items

VARIABLE = value

KAMBIN=${KAMBIN:-/usr/local/sbin/kamailio}

etc/utils: shell script with some utility functions

- kill_kamailio()
- kill_pidfile()

ktestsctl:

- Executes all the units
- Execute a specific unit
- Starts mysql (if needed)
- Quiet/Verbose mode

ktestsctl will ignore units listed in etc/excludeunits.txt

Current units

tasync0001: Verifies variable values when using asynchronous routes (async)

tauthx0001: Adds a subscriber to the DB and verifies authentication (with and without multidomain, auth_db)

tcfgxx0001: Test the default kamailio config file with various combinations of defines (with and w/o DB, with debug, etc)

tcfgxx0002: Test the default kamailio .cfg file without any special define (OPTIONS request)

tgeoip0001: Performs a GeoIP (geoip2) look up and verifies the result

tgroup0001: Adds user to an ACL group, and verifies it belongs to it (group)

tmodxx0001: Tries to run kamailio with all modules loaded

tphonu0001: Test number normalisation with phonenum module (phonenum)

tulocx0001: Registers a user and verifies it was added to the location entries (usrloc)

How to add a unit

Minimum work:

- Add a unit folder
- Add a unit script

Optionally:

- Add a custom kamailio .cfg

If a module is needed and is not built, the test will fail:

- Add dependencies for the module
- Remove module from the excluded list (exclude_modules)

Test, fail, change the test, rinse and repeat

Raise a PR

Consider supporting the new unit in all the OS distributions

Docker to the rescue

Easy to install in a variety of systems

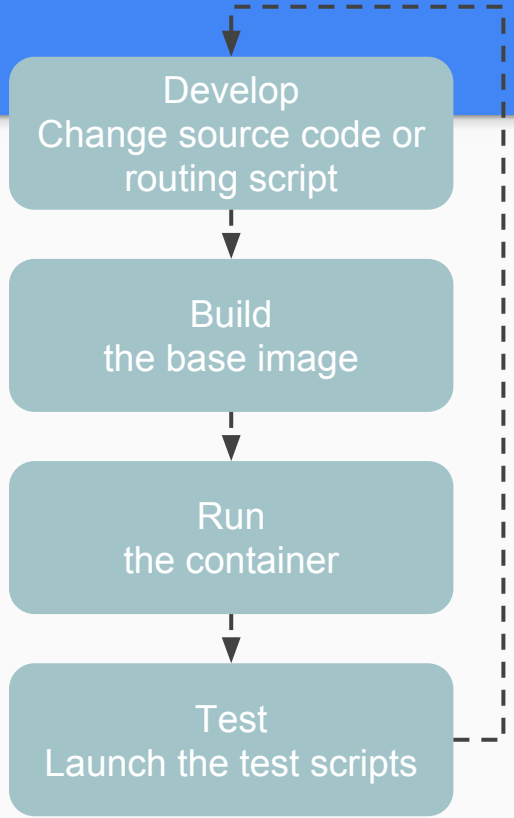
Allows testing against various OS distributions

Can run on a laptop, server or CI system

Intuitive workflow: 1. Develop, 2. Build a base image with latest code, 3. Run a container with that image, 4. Run the tests, 5. Rinse and repeat

Makes it easy to collaborate

(Also as: Build infrastructure for artefacts and Prototyping)



Dockerfiles

Install base libraries to build Kamailio

Install mysql (client and server)

Install sipsak, GeoLite2

Copy Kamailio source code (from branch or fork) into image, and Build Kamailio

Copy kamailio-tests units into image

Define an ENTRYPOINT and a (default) CMD

“A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.”
https://docs.docker.com/engine/reference/builder/

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Build the base image

The Dockerfiles are in kamailio-tests/docker/

Currently supported Debian 9 and CentOS 7 (but easy to extend)

The kamailio source code is copied inside the image (so you can test any branch or fork)

The unit tests are copies inside the image (so you can easily add new units or test a unit in isolation, and without re-building kamailio)

Change tag (-t TAG) to store various versions

$ docker build -t kamailio-tests-deb9x .

$ docker build -t kamailio-tests-deb9x -f kamailio-tests/docker/Dockerfiles.debian9 .

More about the Docker build

Docker images can be kept small with proper actions

- Perform all installations in the same command, to build a single layer
- Clean up yum/apt repos

Docker builds use caching: if a layer doesn't change than the existing one is used

- Explicitly invalidate the cache: --no-cache=true
- By changing an ENV variable at some point in the Dockerfile
- By changing the files that will be COPYed (or ADDed)

e.g. if you just change the test unit, Docker build will use the cache for the previous layers, and won't rebuild Kamailio

Run the container and launch the tests

The Dockerfiles define an ENTRYPOINT (default action that can be overridden at runtime)

The mysql DB is created optionally

You can run all the units or just one

Quiet/Verbose mode (based on ktestsctl)

$ docker run kamailio-tests-deb9x

$ docker run kamailio-tests-deb9x -q run tasync0001

```
=== unit tests execution report ===

running test units at: Mon May  7 11:17:11 UTC 2018

* test unit tasync0001: async routing - cfg variables tests
- test unit tasync0001: ok
* test unit tauthx0001: user authentication - basic tests
- test unit tauthx0001: ok
* test unit tcfgxx0001: default config file - basic load tests
- test unit tcfgxx0001: ok
* test unit tcfgxx0002: default config file - basic sip signaling tests
- test unit tcfgxx0002: ok
* test unit tgeoip0001: geoip2 module - basic tests
- test unit tgeoip0001: ok
* test unit tgroup0001: group module - group membership tests
- test unit tgroup0001: ok
* test unit tmodxx0001: load all modules - detect undefined symbols
- test unit tmodxx0001: ok
* test unit tphonu0001: phonenum module - basic tests
- test unit tphonu0001: failed
* test unit tulocx0001: user location - basic tests
- test unit tulocx0001: ok

=== unit tests execution end ===
```

Launch a specific unit

Handy during test development: launch a specific unit (instead of the full suite)

$ docker run kamailio-tests-deb9x run -q tasync0001

In this example, with -q flag for less verbose output.

Exclude a specific unit

There can be reasons to exclude a unit from the list, e.g.:

- Availability of the related module
- Maturity of the unit
- Libraries depending on OS

Add the unit (on a dedicated line) to etc/excludeunits.txt.DISTRIBUTION, eg.:

- etc/excludeunits.txt.debian9
- etc/excludeunits.txt.centos7

Future development

More unit tests

More complex scenarios (sipp and HTTP)

Integration with Jenkins and others

Additional OS support

<Your idea here>

Kamailio is a complex framework typically interacting with complex systems.

With **kamailio-tests** we can reduce the amount of end-to-end testing, by defining proper, isolated unit tests.

See also...

The work done in **evosip** (Presentations from Paolo and Alex in the next days)

My previous work in kamailio_ci project:
https://github.com/giavac/kamailio_ci

My workshop at Kamailio World 2016:
https://www.slideshare.net/GiacomoVacca/continuous-integration-and-kamailio

Thanks!

Giacomo Vacca
@giavac
https://github.com/giavac
https://www.linkedin.com/in/giacomovacca/

Q&A