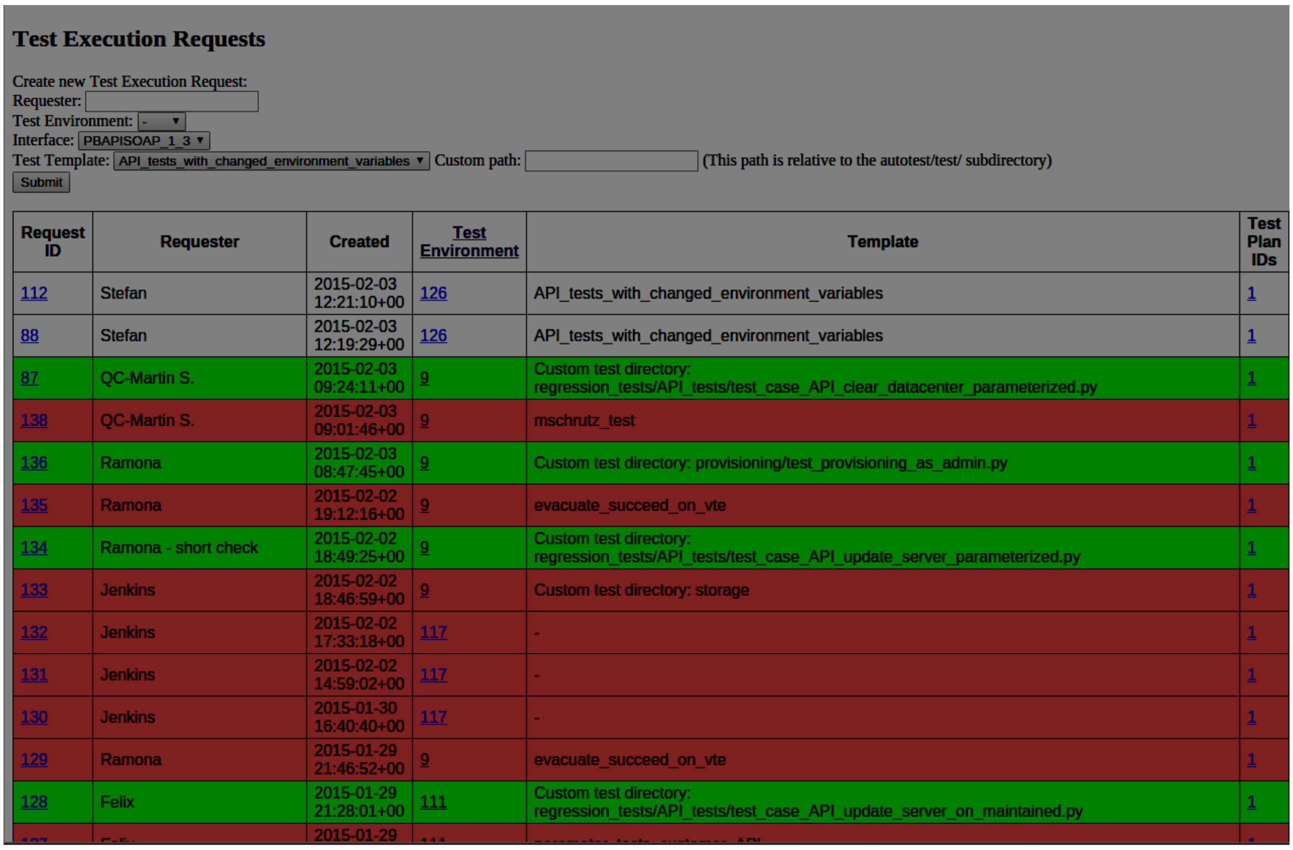
Python Online Test

Test executor web application

Implement a web application that provides a central place to run Python-based tests.
These tests will be run on a shared environment that provides limited hardware resources to developers.
To have some sample tests, just create dummy Python tests (some failing, some not, maybe add some delays for more realism).
You might also run the tests for your own application if you like.
The tests will be executed right on the machine that hosts the test executor application.

The application might look remotely like this:



Test Execution Requests

Create new Test Execution Request:
Requester:
Test Environment: -
Interface: PBAPISOAP_1_3
Test Template: API_tests_with_changed_environment_variables Custom path: (This path is relative to the autotest/test/ subdirectory)
Submit

Request ID	Requester	Created	Test Environment	Template	Test Plan IDs
112	Stefan	2015-02-03 12:21:10+00	126	API_tests_with_changed_environment_variables	1
88	Stefan	2015-02-03 12:19:29+00	126	API_tests_with_changed_environment_variables	1
87	QC-Martin S.	2015-02-03 09:24:11+00	9	Custom test directory: regression_tests/API_tests/test_case_API_clear_datacenter_parameterized.py	1
138	QC-Martin S.	2015-02-03 09:01:46+00	9	mschrutz_test	1
136	Ramona	2015-02-03 08:47:45+00	9	Custom test directory: provisioning/test_provisioning_as_admin.py	1
135	Ramona	2015-02-02 19:12:16+00	9	evacuate_succeed_on_vte	1
134	Ramona - short check	2015-02-02 18:49:25+00	9	Custom test directory: regression_tests/API_tests/test_case_API_update_server_parameterized.py	1
133	Jenkins	2015-02-02 18:46:59+00	9	Custom test directory: storage	1
132	Jenkins	2015-02-02 17:33:18+00	117	-	1
131	Jenkins	2015-02-02 14:59:02+00	117	-	1
130	Jenkins	2015-01-30 16:40:40+00	117	-	1
129	Ramona	2015-01-29 21:46:52+00	9	evacuate_succeed_on_vte	1
128	Felix	2015-01-29 21:28:01+00	111	Custom test directory: regression_tests/API_tests/test_case_API_update_server_on_maintained.py	1
[illegible]	[illegible]	2015-01-29	[illegible]	[illegible]	[illegible]

Features to be implemented

1. Create a new test run, providing:
   - user name
   - test environment ID (choose an ID between 1 and 100)
     - It should not be possible to run a test in an environment where there is already a test running!
   - choose one or more files to test, available tests are read from file system

    - the interface between the test executor web application and the Python test runner itself can be chosen freely

2. List all previous test runs as a table including their outcome (failed, success, running)
    - Bonus:
        - live-update of running/finished tests via Ajax
3. List details for one test run
    - Show all data provided in 1.
    - Show all logs that a test created
    - Bonus:
        - live-update the current state and the logs of this test run via Ajax

Requirements

Generally spoken, you are free to choose any library, tool and framework that helps fulfilling the task. The only hard requirement is to use Python (choose between 2.7 or 3.x) for the back end.

It is not essential to fulfill the task by 100%, it is more important that the resulting code is clean, concise, maintainable and (not overly) documented. Try to apply the KISS principle.

Backend

The backend of the web application should be implemented as a REST-service so it can be easily automated with external scripts.
To persist data, use any database (ORM or non-ORM, depending on what fits best in your opinion).

Frontend

For the frontend, choose any Javascript framework (or Coffeescript) you like.
Providing a highly responsive frontend using Ajax or Websockets is a bonus.

Tests Runner

Choose any test runner you like – unittest, py.test, nose or whatever.

Installation

The resulting application should be runnable on a standard Debian system running Wheezy.

Please provide a short documentation about the dependencies/requirements and how to run the final application.

Bonus task – profiling in production

You are running an arbitrary Python-based web application in production.
You discover that your web application has a memory leak: every 3-5 days, the memory usage of all Python processes spawned by your web application consumes all available memory of the server.

How would you approach this problem and find out the root cause for this memory leak?

Good luck!