Why is testing important for ERP5

ERP5 is a complex framework which contains hundreds of thousands of lines of code of different languages (Python, XML, HTML, Javascript, etc). In order to guarantee it works and make sure no regressions can occur it's very important that we test every commit done in our repositories thus during years ERP5 has thousands of test suites which try to test nearly all possible ERP5 applications.

How did we do it before and why we needed a change

Before SlapOS and concept of a distributed cloud connected over a mesh IPv6 network called re6stnet we used to have manually setup few boxes that had all required components installed system wide by respective linux distribution package manager. Even though this approach worked it had few downsides:

- we used to notify test results by email only thus keeping no track of them anywhere else except mailing list - with new test node infrastructure we do save them inside our own ERP5 instance thus making them available for anybody on the web who can follow the quality of ERP5 code by simply visiting this link
- we were using system wide packages for ERP5 provided by Linux package managers for respective distro thus due to different way packages were made and hacked by respective distros we had no guarantee that ERP5's tests passing on OpenSuse will pass with Debian. Overall depending on system wide packages required quite a lot of efforts and manual system setup that was expensive in terms of engineering time. With SlapOS and automatic ERP5 deployment possibilities we can request a test node to be setuped in any Linux distro and it will compile all ERP5 packages required without depending on system ones which makes us OS independent.
- we had no nice user interface that would allow a developer to request that his / hers own ERP5 branch is tested. This was solved with erp5_test_results ERP5 application which is responsible for providing nice UI for developers and then latter on distributing test tasks to all available test node instances inside the SlapOS cloud.
- we used to develop before mostly against master ERP5 branch which required quite an attention so developer don't break it accidentally. With help of git and branching and test node we were able to develop inside own developer branches which can be tested separately and if tests passes safely merged to master. Logically this requires a fleet of test nodes instances and machines which have to be configured - now done automatically by SlapOS.

How it works now

The architecture of test node contains following components:

- test node which is a software setup inside a machine in SlapOS cloud. It is responsible for executing any test suite
- test suite - collection of unit and / or functional tests to be executed
- git repositories - where code to be tested is
- central test node distributor tool and user interface site - distributor itself is an ERP5 instance

Test nodes doesn't know in advance on which testsuite they are going to work on. Therefore every test node is defined with the url of a distributor. The test node will call "startTestSuite" on the distributor and it will get all needed parameters to work on one or many test suites.

The first time a test node calls startTestSuite, distributor is going to look if this test node already exists. If not, then it will be created under test node module.

From time to time an alarm at distributor looks at all defined test suites and available test nodes and distribute the work to
do. This alarm avoid moving test suite from a test node to another uselessly. In the same time, this alarm is checking for all test node that seems dead (10 hours without sending message) and invalidate them. Like this test suite allocated to a dead test node will be moved to another test node automatically.

Let's say we have 2 testing nodes daemon (A and B) running on two different computers. Each daemon is doing checkout or update of repositories. Since A and B can run with several minutes or even more of interval, distributor needs to make sure that both A and B are testing same revision. Therefore testnode A (assuming no test is already started) will do:

1. checkout/update repository 1, get revision x, checkout/update repository 2, get revision y
2. Ask distributor to start unit test for a particular test_suite title with following pair of repository and revisions:
   
   
   
   

3. Then runTestSuite is launched for [(repository 1',x), (repository 2',y)]

And then testnode B will do (running a little bit after testnode A):

1. checkout/update repository 1, get revision X, checkout/update repository 2, get revision Y
2. Ask distributor to start unit test for a particular test_suite title with following pair of repository and revisions:
   
   
   

3. And then distributor check if there is already running test, see there is one, so returns previous commit x, y:

4. Then runTestSuite reset git repositories and launch test for [(repository 1',x), (repository 2',y)]

Like this we are sure that all computers running the same test suite will be synchronized.

Conclusion

We manage to reduce a lot time needed to setup test node machines, we reused a lot of SlapOS code, principles and recipes to setup ERP5 inside a test node which improved general ERP5’s own setup recipes, we learned how to build and compile our own required packages in order to create isolated and safe environment inside any major Linux distribution. Last but not least we simplified a lot developers’ life, created procedures to follow and implied strict rules for contributing to ERP5: if your own branch do not pass test suites you are not allowed to merge your changes inside ERP5 master branch. This rule itself improved quality of ERP5 code.