Continuous Integration (&delivery) system (CI)

Goal: automation of software production.

Typical architecture example
Typical case of CI workflow: management of Pull Requests.

1. Pull Request (into master)
PR pending until the successful completion of the CI job.
The application is built including the modifications introduced by the PR.

1. Pull Request (into master)

2. trigger job

3. building system

The application is built including the modifications introduced by the PR.

Continuous Integration (CI)

Continuous Delivery (CD)
Integration tests are run...

1. Pull Request (into master)
2. trigger job
3. building system
4. testing engine

Continuous Integration (&delivery) system (CI)

Continuous Deployment system (CD)

CD job
and the PR is validated only if the tests are passed.
Then application is packaged...

1. Pull Request (into master)
2. trigger job
3. new build
4. PR validated
5. new release

Continuous Integration (&delivery) system (CI)

Continuous Deployment system (CD)

CD job
and uploaded so as to be available for every person concerned.
A successful CI job allows the PR to be merged into the code base.
CI job can optionally trigger a Continuous Deployment job...

1. Pull Request (into master)
2. trigger job
3. building system
4. testing engine
5. packaging system
6. repository system
7. allow merging

Continuous Integration system (CI)

Continuous Delivery system (CD)

version control system

deployment system

new build
PR validated
new release
new release available

CD job

1/4
Continuous Integration (CI)

so as to deploy the new version of the application.
The benefits of the CI (after some reasonable efforts):

- Prevent software regression: prevent faulty code from getting merged into the main repo.
- Less error prone: human out of the loop but everything is still manually operable.
- Improve reactive security: increase the frequency of releasing, from a CVE to a new release.
- Etc.
Continuous Integration (CI)

ESGF-test-suite & ESGF-jenkins role:

**ESGF-jenkins:** Continuous Integration (&delivery) system (CI)

1. **build system**
2. **ESGF-test-suite testing engine**
3. **packaging system**
4. **repository system**

**Continuous Deployment system (CD)**

1. **deployment system**

**version control system**
A CI example involving esgf-docker:

**ESGF-jenkins**: Continuous Integration (&delivery) system (CI)

- **github**: version control system
- **ESGF-test-suite**: testing engine
- **docker-compose**: building system
- **docker-compose**: packaging system
- **dockerhub**: repository system

Notes:

- **ESGF-test-suite** is not specific to Continuous Integration and ESGF-docker (e.g. can check your production ESGF nodes).

- **ESGF-jenkins** is not specific to ESGF-docker: it welcomes any ESGF projects.
Goals:
- Gather the integration tests of the ESGF stack.
- Developers of the ESGF stack are invited to implement/port their integration tests into ESGF-test-suite.

Architecture: a well organized set of Python 2.7 scripts based on Nose2 (also based on Unittest). Git repository: https://github.com/ESGF/esgf-test-suite. Author: Nicolas Carenton.

For the time being, only fundamental integration tests has been implemented:
- Pinging web pages (cog, thredds, stats, search, orp, etc.).
- Cog/django login.
- HTTP and gridftp downloading.

Features provided by esgf-test-suite:
- Myproxy authentication (credential) routines.
- High levels web crawling routines based on Selenium API, using Firefox.
- High levels Thredds catalog routines.
- Nice reporting (human readable, JSON, XML, etc.).
- Selection of tests thanks to Nose.

ESGF-test-suite is standalone application with its dependencies wrapped into a singularity image (container system):
- Install singularity, once.
- Run a interactive shell within the image of the dependencies of ESGF-test-suite.
- Run ESGF-test-suite.
Singularity esgf-test-suite_env.singularity.img:~/private/esgf-test-suite/esgf-test-suite> python2 esgf-test.py -v --nocapture --nologcapture --tc-file ../config_files/my_config_prod.ini -a '!compute,!cog_create_user' --with-id
#55 cog_root_login ... ok
#56 cog_user_login ... ok
#57 basic_ping (https://vesg.ipsl.upmc.fr/thredds) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-orp) ... ok
#58 dl_gridftp ... ok
#59 dl_http ... ok
#60 basic_ping (https://esgf-node.ipsl.upmc.fr/esgf-idp) ... ok
#61 basic_ping (https://esgf-node.ipsl.upmc.fr/) ... ok
  basic_ping (https://esgf-node.ipsl.upmc.fr/solr/#) ... ok
  basic_ping (https://esgf-node.ipsl.upmc.fr/esgf-search/search) ... ok
  basic_ping (https://esgf-node.ipsl.upmc.fr/esgf-orp) ... ok
#62 myproxy_get_credentials ... ok
#63 myproxy_get_trustroots ... ok
#65 slcs_django_admin_login ... ok
#64 basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/cmip5/stats-by-space/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/cmip5/stats-by-dataset/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/cmip5/stats-by-experiment/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/cmip5/stats-by-model/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/cmip5/stats-by-variable/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/obs4mips/stats-by-space/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/obs4mips/stats-by-dataset/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/obs4mips/stats-by-realm/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/obs4mips/stats-by-source/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/obs4mips/stats-by-variable/xml) ... ok
  basic_ping (https://vesg.ipsl.upmc.fr/esgf-stats-api/cross-project/stats-by-time/xml) ... ok

Ran 25 tests in 23.137s
OK
Adding new integration tests:

1. Create a new file prefixed “test_” in the tests directory

2. Implement a Python class (set of tests) and some methods (tests) with annotations (ids):

```python
class TestSlcs(AbstractBrowserBasedTest):
    def __init__(self):
        AbstractBrowserBasedTest.__init__(self)
        self.idp_node = config.get(config.NODES5_SECTION, config.IDP_NODE_KEY)
        self.username = config.get(config.SLCS_SECTION, config.ADMIN_USERNAME_KEY)
        self.password = config.get(config.SLCS_SECTION, config.ADMIN_PASSWORD_KEY)

    @attr('slcs_django_admin_login')
    def test_slcs_django_admin_login(self):
        # Always start with this method so as to dodge side effects.
        self.reset_browser()
```

ids of a set of tests

scope of a set of tests

unique id of a test

implementation of a test

inherits routine methods

using a routine method
ESGF-jenkins, a cluster of 3 Jenkins nodes:

- **Scalable** (by adding new worker nodes).
- Job failover (across the nodes of a cluster).
- Extensible (plugin system).
- Nice web interface (https://esgf-build.ipsl.upmc.fr/jenkins/).
- Point&Click configuration of Job.
- CI Job description language (Jenkins pipeline script named Jenkinsfile).
- GUI Job pipeline builder.
- Secure credential manager.
- Provides REST API.

Goals:
- Run CI jobs (provide computation resources).
- Ease the interactions between CI jobs and other applications (Version Control System, Continuous Deployment systems, etc.).

Working capacities:
- Every node can run 2 jobs at the same time.
- Worker nodes can run 1 job using Docker and Singularity at the same time.
- Provided job subsystems (Docker, Singularity, Git, Maven, Ant, Gradle, Bash, etc.).
Examples of interactions

- Github: allows merging PR, creates tags, etc.
- Dockerhub: pushes container images
- ESGF-jenkins: sends notifications (pushes, Pull Request creations, etc.)
- Slack #esgf-docker-ci: sends notifications

A complete and commented job script is available at:
https://github.com/ESGF/esgf-docker/blob/master/Jenkinsfile
Job status (failed, successful, presented like a weather forecast):

![esgf_docker_multibranch_pipeline]

ESGF software stack as Docker images

<table>
<thead>
<tr>
<th>Branches (2)</th>
<th>Pull Requests (10)</th>
<th>Tags (0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S W</td>
<td>Name ↓</td>
<td>Last Success</td>
</tr>
<tr>
<td>![blue circle] ![sunny weather]</td>
<td><strong>devel</strong></td>
<td>1 mo 7 days - #14</td>
</tr>
<tr>
<td>![orange circle] ![sunny weather]</td>
<td><strong>master</strong></td>
<td>2 mo 9 days - #2</td>
</tr>
</tbody>
</table>

Job stage view:

![Stage View]

Average stage times: (Average full run time: -15min 18s)
Exploring the log of a CI job:

<table>
<thead>
<tr>
<th>build</th>
<th>config</th>
<th>start</th>
<th>run test-suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>8min 58s</td>
<td>2s</td>
<td>1min 1s</td>
<td>7s</td>
</tr>
<tr>
<td>10min 0s</td>
<td>4s</td>
<td>4min 15s</td>
<td>37s</td>
</tr>
<tr>
<td>9min 31s</td>
<td>5s</td>
<td>50s</td>
<td>41ms</td>
</tr>
</tbody>
</table>

Failed with the following error(s):

Shell Script
script returned exit code 1

See stage logs for more detail.

15:55:22 warning: no previously-included files matching '*.py[co]' found anywhere in distribution
15:55:22 no previously-included directories found matching 'docs/_build'
15:55:22 File "build/bdist.linux-x86_64/egg/jinja2/asyncfilters.py", line 7
15:55:22 async def auto_to_seq(value):
15:55:22 ^
15:55:22 SyntaxError: invalid syntax
Builder of pipelines of jobs:

Build Pipeline

Pipeline

#19 esgf-node-manager_master

#18 esgf-dashboard_master

esgf-desktop_master

#16 esgf-getcert_master

Run
History
Configure
Add Step
Delete
Manage
Slack notifications (a PR successively failed then successfully tested):

Monday, October 22nd

jenkins 2:55 AM
ESGF-DOCKER PR-84#12: testing pull request PR-84 (branch devel)

jenkins 3:16 AM
ESGF-DOCKER PR-84#12: FAILURE

jenkins 1:27 PM
ESGF-DOCKER branch devel#14: testing commit 3eeb84b9dcd96cc390a8c9902b0ef4e4302e8f on branch devel

jenkins 1:59 PM
ESGF-DOCKER branch devel#14: SUCCESS

Words in blue are hyperlinks that point to the jobs.
Want an ESGF-jenkins account?

Need help to implement integration tests for ESGF-test-suite?

send me an email: sgardoll@ipsl.fr

Thank you for your attention