Development, tests and deployment of web application in DACE

Julien Burnier[1], Fabien Alesina[2], and Nicolas Buchschacher[3]
[1]*University of Geneva, Geneva, Geneva, Switzerland;*
julien.burnier@unige.ch

[2]*University of Geneva, Geneva, Geneva, Switzerland*

[3]*University of Geneva, Geneva, Geneva, Switzerland*

Abstract.
The Data and Analysis Center for Exoplanets (DACE) is a web platform based at the University of Geneva (CH) dedicated to extrasolar planets data visualisation, exchange and analysis. This platform is based on web technologies using common programming languages like HTML and Javascript for the front-end and a Java REST API for the back-end. Over the last 12 months, the process to maintain, develop, test and deploy the applications has been dramatically improved to facilitate the maintenance and the integration of new features. The goal of such automation is to let more time to focus on development and reduce the duplicated work. To achieve this result, we migrated our Java application to the Maven software project management and added unit tests. We implemented a pipeline on GitLab which consists of executing the tests and deploy the application in a dev environment at every commit. The front-end side is then tested using the Selenium web browser automation to simulate the user - website interactions and compare the new results with the old ones. Once all the tests are validated, a manual action on the GitLab interface can be done to deploy the application on the official web site and we ensure the compatibility of the new features with the production version. We are currently working to have a very complete set of tests on both back and front end in order to remove the manual part of production deployment and to have a fully automated integration of our applications.

1. Introduction

About one year ago, the DACE platform suffered deployment issues and stability. Sometimes, our team was afraid to go on production because we didn't deploy new code since 2-3 months. Also a quick bug fix to deploy was complicated because the code was not updated since a long time. To facilitate development and deployment we naturally decided to implement continous integration.

2. Java code with maven and unit tests

The first thing to do was to encapsulate existing code to a project management tool which manage dependencies and simplify compilation, test and jar/war creation. We chose maven as it remains the reference on Java. Then we added unit test to existing code where possible. To do this we followed mvn convention and started using JUnit.

Finally, the idea for backend API for example, was to apply Test Driven Development when possible.

3. Gitlab - Continuous Integration tool

Then we needed a tool to do continuous integration. Fortunately, other project inside our university started to use Gitlab. We migrated our code from svn to git and pushed code inside gitlab. After that, a gitlab-runner must be installed where you want to run your build. And finally, we added a *.gitlab-ci* file into every project. You can see an example of our gitlab file (only with stage Test, Build and staging deployment) on figure 1. With this small configuration file you already have tests runned after each commit

```
stages:
  - build
  - staging
  - release
  - production

Maven Test and Build:
  stage: build
  script: mvn clean install
  # Need this config to pass the jar to next stages
  artifacts:
      paths:
        - target/*.war
      expire_in: 1 day

Deploy to Staging:
  stage: staging
  environment:
    name: Staging
  script:
    - export WEBAPP_FULL_NAME=$(basename target/*.war)
    - cp target/$WEBAPP_FULL_NAME /opt/tomcat/dace-webapps
    # ln -fsn will do the following : -f overrides if symlink already exists. -n will not follow previous symlink.
    # -s : standard option to have a symlink and not a hard link
    - export WEBAPP_NAME=$(echo $WEBAPP_FULL_NAME | sed 's/-.*//')
    - export WEBAPP_NAME="${WEBAPP_NAME}.war"
    - ln -fsn /opt/tomcat/dace-webapps/$WEBAPP_FULL_NAME /opt/tomcat/webapps/$WEBAPP_NAME
```

Figure 1. Example Gitlab code

and you'll receive an email in case of failure. The test and build script is simply *mvn clean install* and maven handle the rest. For deployment on staging (which is dev on our architecture), we do a cp of the war/jar and create a symbolic link to easily rollback in case of failure. For frontend part, the idea is to deploy via rsync on servers and then use Selenium.

4. Selenium

Before going to production, web tests should be done to ensure having no bugs and no regression (same as unit test but on visual part). To do this we use selenium. This tool simulates user interaction on a website. To facilitate selenium test development, we adopted Page Object Pattern. The Page object pattern let you split your selenium test code on each web page. For example, you can do a test on index.html and then click on about.html and create a test AboutPageTest.java to test this specific page. Example of home page test on figure 2

```java
package ch.unige.dace.pages.home;

import ...

/**
 * Created by julien on 20.10.17.
 * <p>
 * Test the home page. Use FunctionalTest to setup selenium environment
 */
public class HomePagePolygonsTest extends FunctionalTest {

    @Test
    public void shouldDisplayHomePage() throws InterruptedException {

        Navigator.openDaceWebsite();

        HomePagePolygons homePagePolygons = new HomePagePolygons();

        assertThat(homePagePolygons.isDisplayed()).isTrue();

        List<LocatorAndPage> polygons = homePagePolygons.getPolygons();

        for (LocatorAndPage polygon : polygons) {
            Page page = Navigator.goToPage(polygon);
            if (Navigator.isPageOpenOnNewTab()) {
                Navigator.switchTab();
                assertThat(page.isDisplayed()).isTrue();
                Navigator.closeTab();
            } else {
                assertThat(page.isDisplayed()).isTrue();
                Navigator.goBack();
            }
        }
    }
}
```

Figure 2. Example Selenium test code

5. Deployment

The deployment is also handled by gitlab. It needs to be defined in *.gitlab-ci.yml*. On our side, after every commit, we deploy backend and frontend on dev environment. Then if everything is ok, the app is released deployed manually on one production server and after some manual tests, deployed on the other production server. Architecture of DACE servers and deployment process can be seen on figure 3

Acknowledgments. This work has been carried out within the framework of the National Centre for Competence in Research PlanetS supported by the Swiss National Science Foundation. The authors acknowledge financial support from the SNSF. This publication makes use of DACE, a Data Analysis Center for Exoplanets, a platform of the Swiss National Centre of Competence in Research (NCCR) PlanetS, based at the University of Geneva (CH).

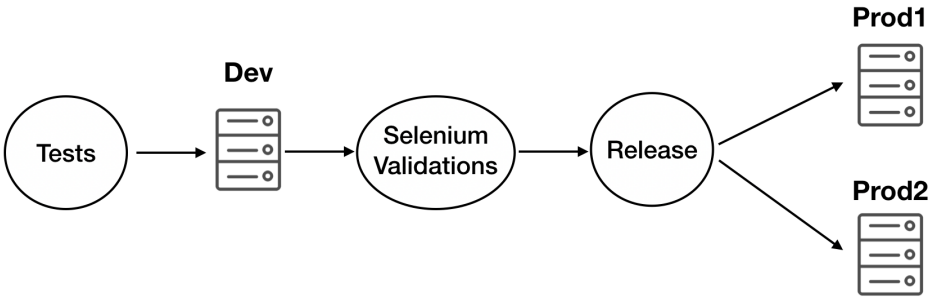

Figure 3. DACE continuous integration process