PhUSE 2017

Paper TT03

Catching Up: Continuous Integration Pipelines For Clinical Analysis

Andrew Karpow, Bayer AG, Berlin, Germany

ABSTRACT

Continuous Delivery (CD), Continuous Integration (CI) and test automation are mature state of the art topics for software development startups. Still, they are quite uncommon in current development environments for clinical analysis. We will present an implementation and example of a clinical analysis workflow using the popular Jenkins CI Software and GIT version control system. It has the capability to schedule SAS jobs, generate software quality reports and detect implausibility errors.

INTRODUCTION

What is continuous Integration?

Continuous Integration (CI) is a practice of merging all developer copies several times a day to a shared repository. For each change of code in this repository a centralized CI system is triggered, which fully automatically analyzes, verifies and builds the code. At the basis it is a principle to help developers to get fast feedback about the code quality on a regular basis. Developers can concentrate on the code and don't need to integrate all different kinds of checks and tools to ensure working and reproducible code. CI systems are typically centralized systems and therefore can reproduce results independently from, often customized, developer system. Ultimately, such systems can be used to continuously deploy (CD) programs based on their CI quality result in a validated or even productive environment.

Jenkins [1] is the leading industry CI system that can easily be integrated in closed environment with support of a variety of central repositories, programming languages and 3rd party tools like SAS®. We will present a new plugin for Jenkins that is able to run SAS® programs automatically within the Jenkins tool and parse output generated by SAS® to be interpreted by Jenkins according to quality requirements.

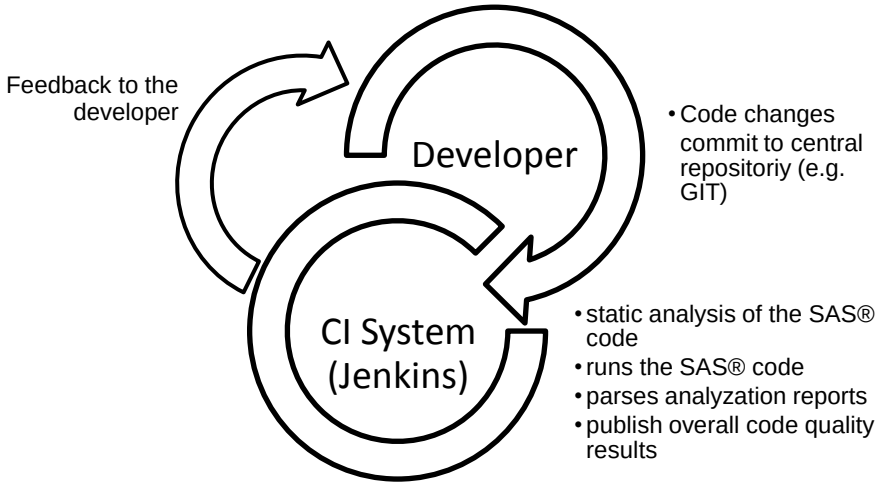

Figure 1: CI lifecycle

JENKINS CI

Jenkins is an open source platform independent automation server with a web based GUI. Originally developed by Oracle as Hudson, the open source fork Jenkins is the most popular CI software in use. It is a modular framework written in Java with a large amount of available plugins for different languages, code repositories, unit test frameworks, static analyzers and more.

Jenkins is able to process multiple projects at the same time, separate build stages in a pipeline and resolve build dependencies to ensure the correct build order. It is also able to scale Jenkins on a cluster farm to improve build times with the help of build agents.

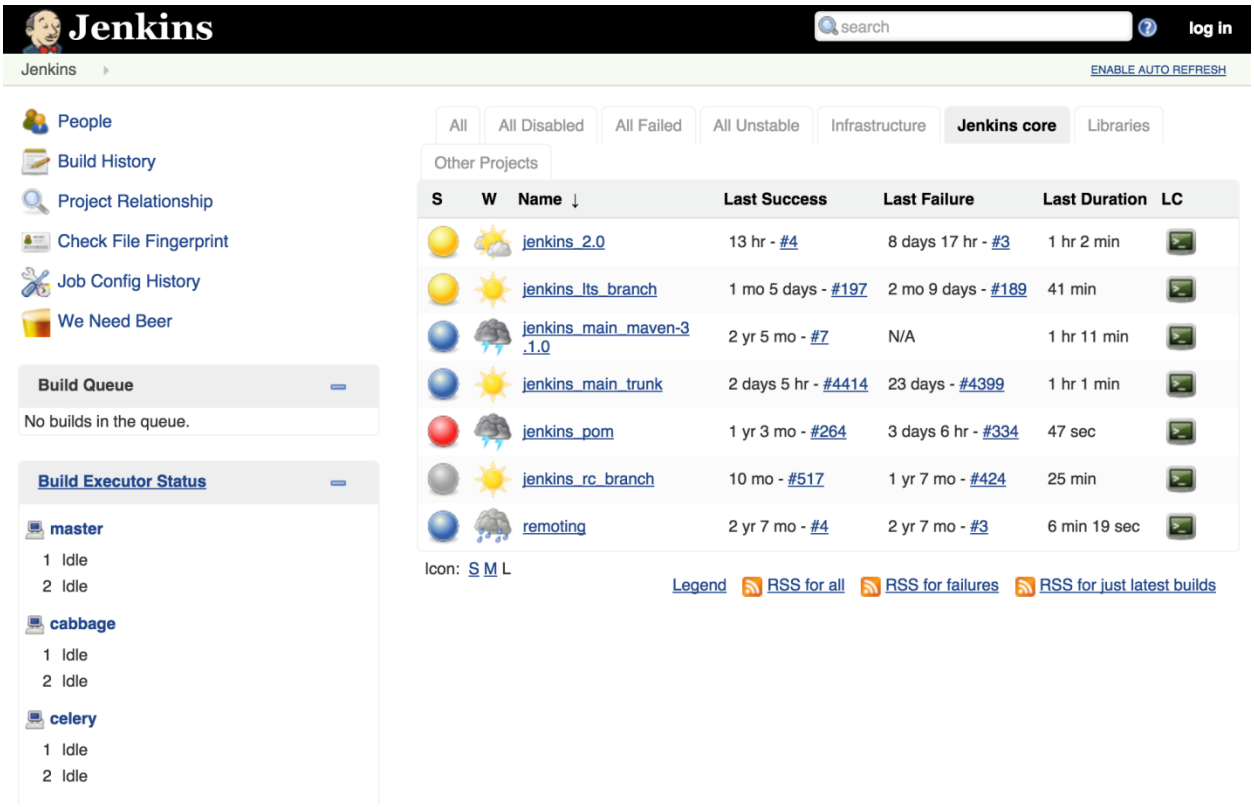

Figure 2: Jenkins dashboard with build status of different projects

MOTIVATION

Since the turn of the millennium, technology companies developed and implemented fast-paced, iterative methodologies such agile development, central repositories and build pipelines to push their work as quickly as possible to the consumer, drastically reducing the time to market. Meanwhile, nearly all software companies adopted the new paradigms, whereas in clinical analytics sharing the same challenge by developing software for calculating results of clinical studies with heavy timelines, adoption of new technologies is progressing quite slowly.

Our motivation is to adopt state of the art development models and automated processes to improve code quality, reduce development time and automate more tasks that are still common but tedious to do manually. The basic challenge to successfully deploy continuous integration in an existing development pipeline is to manage a single source code repository.

We encountered a lot of conservative development models where code is maintained by single persons or departments and co-existing without knowledge about other copies. Even though this paper doesn't address the peculiarities of a single repository system for larger teams in clinical analytics, it is a necessary requirement to efficiently use a CI system. How this is accomplished is an even tougher challenge in contrast of using CI itself.

IMPLEMENTATION DETAILS

We deployed a central repository on basis of GIT, an widespread open source concurrent version system. Also, Jenkins has a superb support of the GIT version system out of the house. Since Jenkins is a pure java container application, it is very easy to deploy standalone or as part as a java application server container.

To our knowledge, the only public available SAS® extension published for Jenkins is the SASUnit plugin [2] by HMS Analytical Software GmbH. SASUnit is a unit testing framework for SAS®-programs that consists purely of SAS® macros and some shell scripts. The SASUnit Jenkins plugin explicitly needs the SASUnit batch files and does only unit testing. We aimed for a more generic and universal implementation.

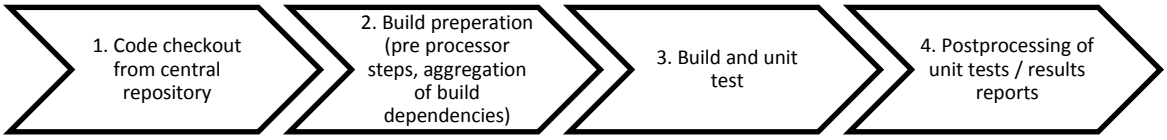

Figure 3: Steps of Jenkins build pipeline

SAS® RUN BUILD STEP

We decided to develop a Jenkins plugin written in Java to tightly integrate a SAS® program with Jenkins. Jenkins builds are typically separate in several build steps (Figure 3). The plugin allows specifying a SAS® program for the step "*3: Build and unit test*", that just executes the SAS® processor with the selected file. It parses the return code of the SAS® interpreter and exposes the build result to Jenkins.

The Jenkins dashboard also displays the command line results of the SAS® run that further helps to investigate build errors without messing with the SAS® command line program itself. An example output of the console output could look like this:

```
Started by user Testuser1
Building in workspace /var/run/Jenkins/workspace
Running SAS Jenkins Plugin
NOTE: Copyright (c) 2002-2008 by SAS Institute Inc., Cary, NC, USA.
…
NOTE: SAS initialization used:
      real time           0.00 seconds
      cpu time            0.00 seconds

1          proc print data=sashelp.air;
2          run;

NOTE: There were 144 observations read from the data set SASHELP.AIR.
NOTE: The PROCEDURE PRINT printed pages 1-3.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.02 seconds
SAS statistics of SAS Run #26 published successfully!
Finished: SUCCESS
```

After a SAS® run is finished, the result and build duration is displayed on the Jenkins dashboard, where a blue bubble indicates a successful run, a yellow one unit test failures, and a red bubble SAS® run failures due to error in the code or missing data (See Figure 4).

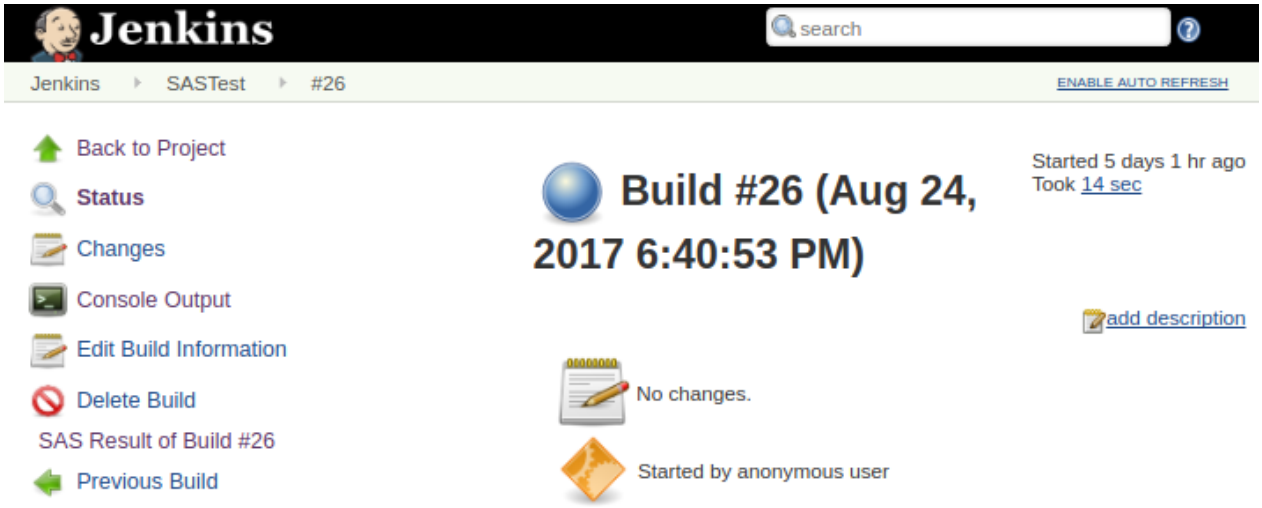

Figure 4: Jenkins Dashboard displaying a SAS run result

SCAPROC POSTPROCESSING

Furthermore, the plugin has a post processing option for step 4, where it parses outputs generated by SAS®, for now the SCAPROC procedure.

The SCAPROC procedure implements the SAS Code Analyzer, which captures information about input, output, and the use of macro symbols from a SAS job while it is running. The SAS Code Analyzer can write this information and the information that is in the original SAS file to a file that you specify. The SCAPROC procedure can also generate a grid-enabled job that can concurrently run independent pieces of the job. You can issue the SCAPROC procedure on your operating system's command line or in SAS code in the SAS Editor window. [3]

As the result file of the SCAPROC procedure contains a lot of interesting statistics about the runtime of a SAS® program and is machine readable, it is a good candidate to be processed by Jenkins. The SCAPROC result file is generated during step 3 and parsed by the plugin in step 4. After that, the Plugin generates a HTML report of the result file that can be accessed through the Jenkins dashboard by clicking on the "*SAS Result of Build #XX*" hyperlink (see Figure 5).

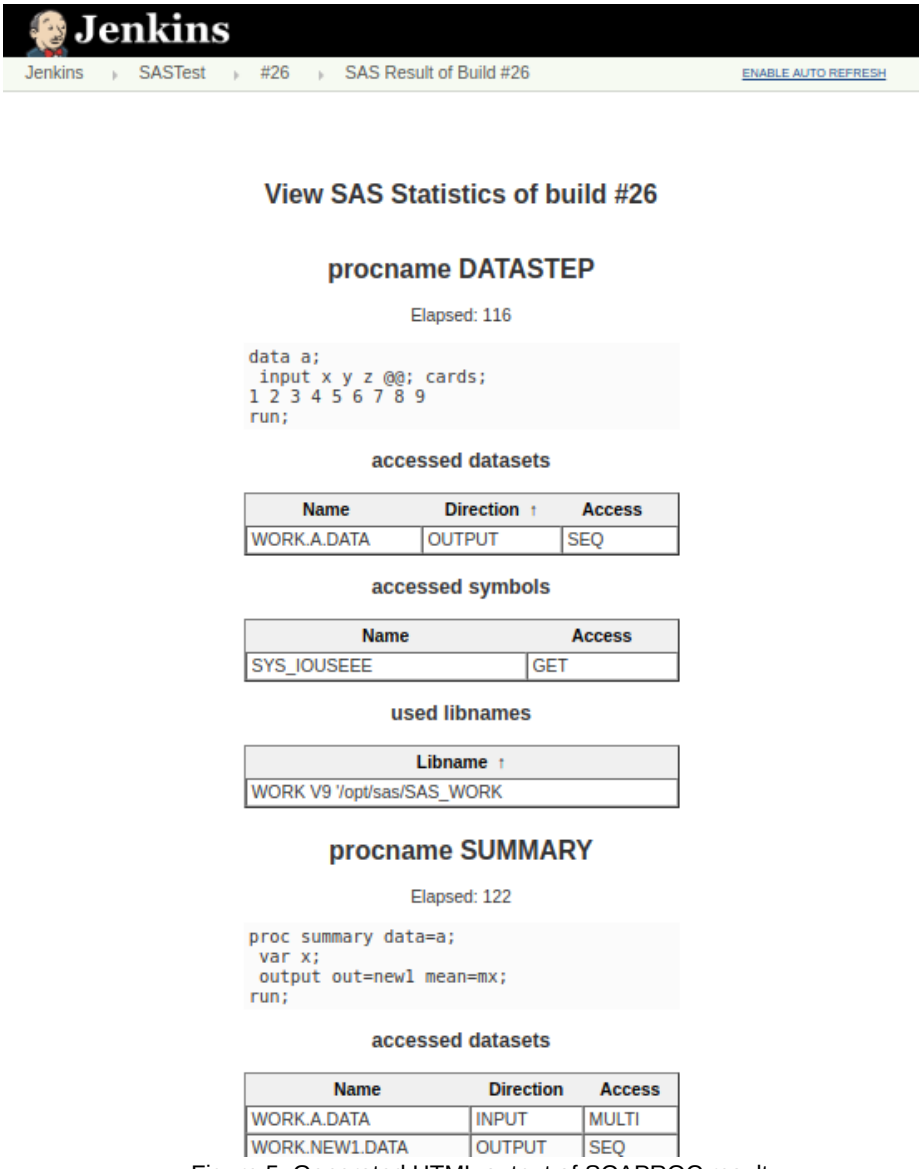

Figure 5: Generated HTML output of SCAPROC result

CONCLUSION

For now, our developed SAS® plugin is able to run and process results and presents the reports in an easy and clear way for SAS® developers. It shows that Jenkins is able to process SAS® jobs in a centralized way and gives, thanks to the self-hosting possibility, to be used in a clinical analysis environment. We also see a lot of improvement by integrating more unit test frameworks into the plugin and run more code quality checks like static code analysis or performance analysis.

REFERENCES

[1] Jenkins, "Jenkins CI," 07 2017. [Online]. Available: https://jenkins.io/.

[2] HMS Analytical Software GmbH, "GitHub Jenkins Plugin for SASUnit," 07 2017. [Online]. Available: https://github.com/jenkinsci/sasunit-plugin.

[3] SAS Institute Inc., "SCAPROC Procedure," 07 2017. [Online]. Available: http://support.sas.com/documentation/cdl/en/proc/70377/HTML/default/viewer.htm#p0sf63lx4fs2m5n14qv1bn8p863v.htm.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Andrew Karpow
Bayer AG
Müllerstr. 170-178
13342 Berlin
Work Phone: +49 173 8742483
Email: andrew.karpow@bayer.com