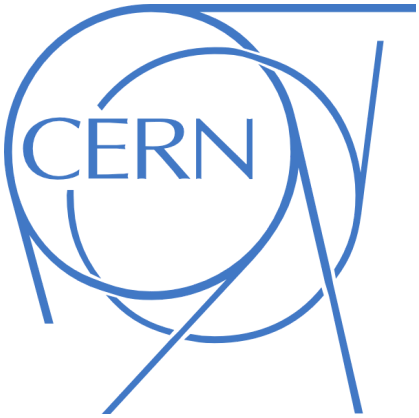
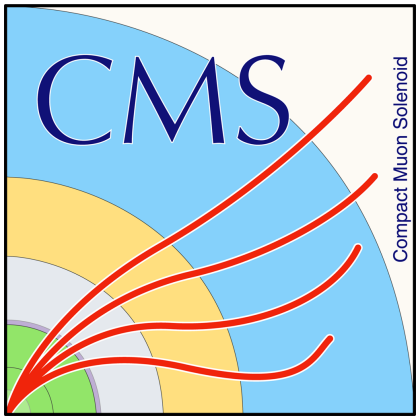




CMS - Automation of Unified building and testing

CERN Summer Online Educational Initiative Report
August 2020

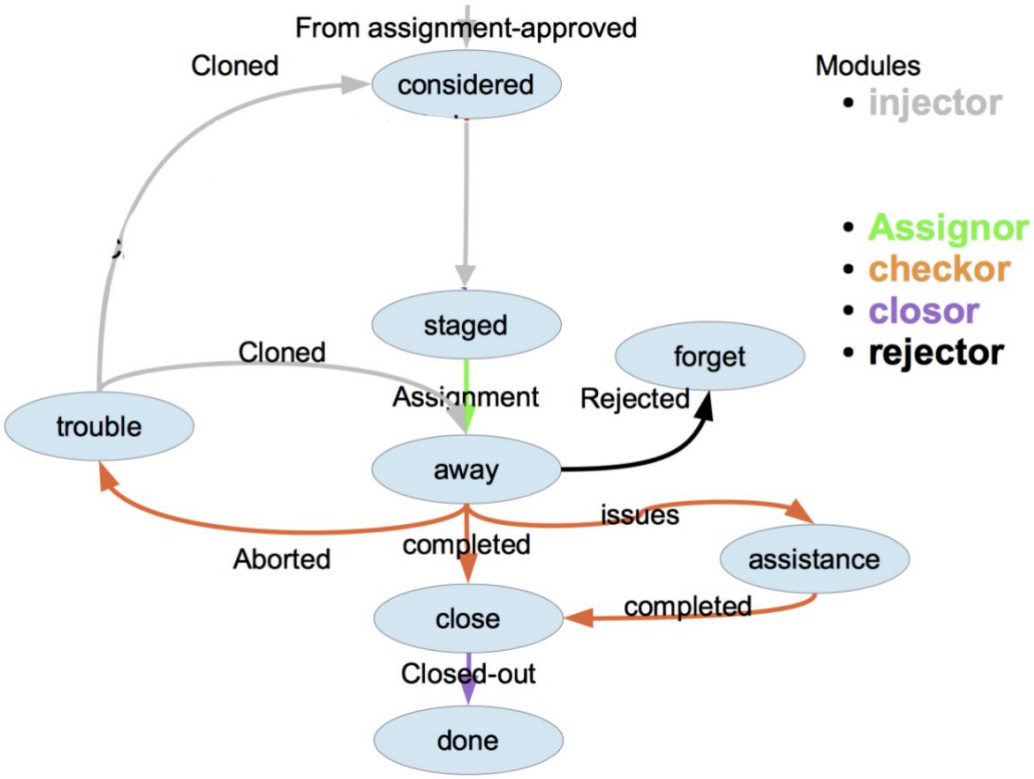
Author:
Ishan Rai Indian Institute of Technology, Roorkee

Supervisor:
Sharad Agarwal University of Wisconsin–Madison, CMS Experiment, CERN
Alan Malta Rodrigues University of Nebraska, CMS Experiment, CERN
Malik Shahzad Muzaffar CERN, Meyrin 1211, Geneve, CMS Experiment, CERN

Abstract

At any moment in time, the *CMS*[1] experiment is running hundreds of thousands of jobs on its distributed computing system, part of the *World Wide Computing Grid*[2]. Billions of collision events are simulated and reconstructed per week, lumped together in datasets for final analysis, published on commonly accessed catalogues. This task requires significant human effort. A possible way to increase the sustainability of computing operations is to develop and improve tools which allow automated tasks.

Unified[3] is part of such a category of software tools. It is a set of *python* components that contributes to the smooth automation of several operations relative to the *CMS*[1] central production. For example, the tool makes sure for the whole workflow management and needs significant integrations for various utilities. Testing *Unified*[3] is a fundamental task, and currently, the job requires manual work. This project aims to implement, via *Github Actions*[4], automation of the *Unified*[3] software integration process, including continuous integration and testing. The goal is to streamline the development, making it easier to integrate fixes and new features for developers and to obtain a fresh build for users.



Unified Workflow

The goal of the project

The primary aim of this project is to establish practices that drive development teams to implement small changes and check-in code to *version control repository*[5] frequently. Since most of the people working on *Unified*[3] develop code following different coding standards, the *CMS*[1] team needs a mechanism to unify, integrate and validate its changes.

The technical goal of this project is to establish a consistent and automated way to test *Unified* software. With the integration and testing process in place, *CMS*[1] offline computing team is likely to commit code changes more frequently, which would lead to better collaboration and code quality.

Most research scientists work with multiple environments other than the production, such as development and testing environments, and *Continuous Integration* ensures there is an automated way to push code changes to them.

CI/CD will require testing on every code before it is merged into the mainline branch because the objective is to deliver quality and error-free code. The final aim of this project is to perform continuous testing through a set of automated regression, performance, and other tests that will run in the *Github Action*[4] CI/CD pipeline.

Project Details

Implementing GitHub action workflows

GitHub Actions[4] helps to automate Unified development through a set of workflows. The files to automate the workflow are stored inside the *.github* folder in the same remote GitHub repository as Unified.
For Unified, this project implemented the following Github Actions workflows.:

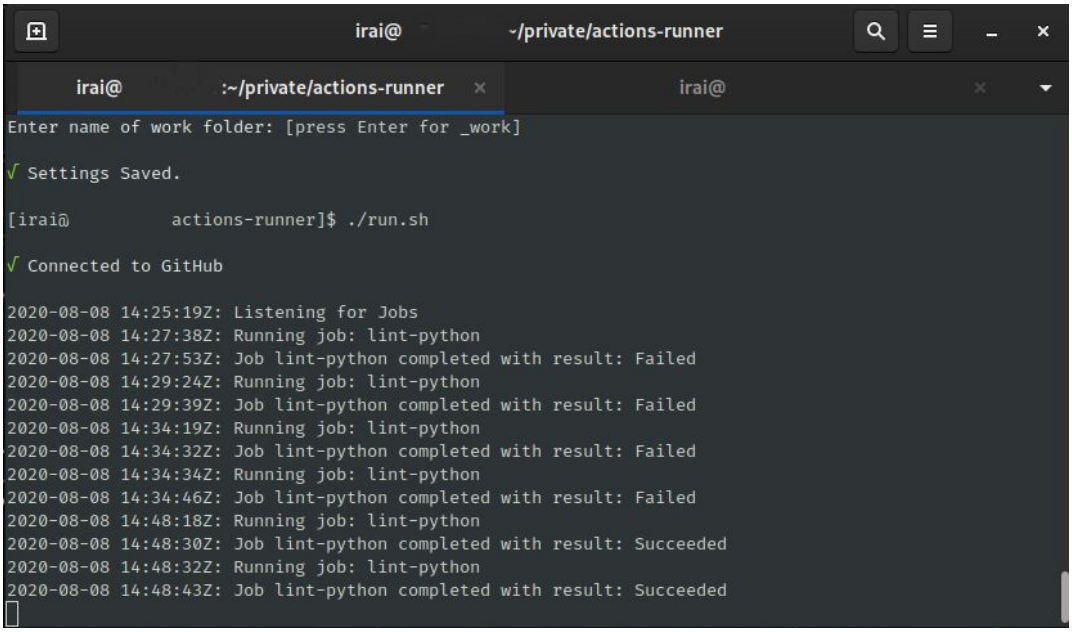
- Check Json format: This Github Action runs on every push and pull request, and checks for the syntax of the json files.
- PyLint Test: Runs on every push and pull request, and checks for any error in python code syntax. Also uses Github Action bot with Github APIs to post detailed pylint review scores as a comment on the pull request.

- ShellCheck: Runs on pull requests only. This GitHub Action workflow checks for syntax errors in shell scripts and uses ReviewDog to provide a line by line review on the pull request.
- ShellCheck/push: Runs on push only and checks for syntax errors in shell scripts.
- syntax check: Runs pylint test, shellcheck test and jsonlint test as *cron*[9] job on the entire repository
- Unit Test: Runs unit tests on a self hosted Linux machine inside CERN infrastructure

All of the above GitHub Action workflows for syntax checks run only on the files that have been changed and not on the entire code base. This greatly reduces the time for tests and shows

Setting up a remote virtual machine to run tests.

Although *GitHub Action*[4] offers remote machines to run tests, this was not possible for Unified because of the various security restrictions imposed by the CMS and CERN infrastructure. A lot of tests required setting up certificates for accessing the web server endpoints. Therefore, it was essential to set up a testing machine on the remote virtual machines inside CERN infrastructure.
The remote testing machine makes a *GET*[9] request to the remote repository after every short interval to figure out if any pull requests were to the remote repository. On encountering pull requests, it clones the repository, checks out to the developer's branch and runs all the tests.
We also added a Github Action workflow to run the entire test suite as part of *cron*[10] jobs every day at 0900 hrs UTC over the whole repository.



Remote virtual machine running the tests

Syntax checking and formatting

For any software testing, checking the syntax is the most fundamental requirement because it not only helps eliminate mistakes that could potentially break the code but also helps maintain a proper standard thus making it easier to debug the existing code and encouraging more contributions from developers. Since many research scientists are contributing to the Unified code base, it becomes essential to establish a set of standards to be followed while writing code.

Unified consists of code written in primarily three languages, three different syntax checkers were required, all of which were implemented using different GitHub Action workflows.

Shell Script

A majority of the unified code includes shell scripts that are used to run various python scripts. All the shell scripts syntax was corrected to meet a standard agreed by the Unified community members. Then, *shellcheck*[11] was used to check the syntax for the shell scripts. Finally, a bot was also integrated to help developers better understand where the syntax errors occurred.

GitHub bot to comment on the wrong syntax code for shell scripts

Python

For testing the python syntax, *pylint*[12] was used. A bot was also developed that gives a detailed report about the *pylint* test score.
For commenting the test report on the Pull Request, *GitHub APIs*[13] were used.

Github Actions Bot with a detailed report on the python syntax check

JSON

JSON files inside the Unified system are used for storing the various configuration settings for the python scripts. For syntax checking of all the JSON files, *jsonlint*[14] tool was used.

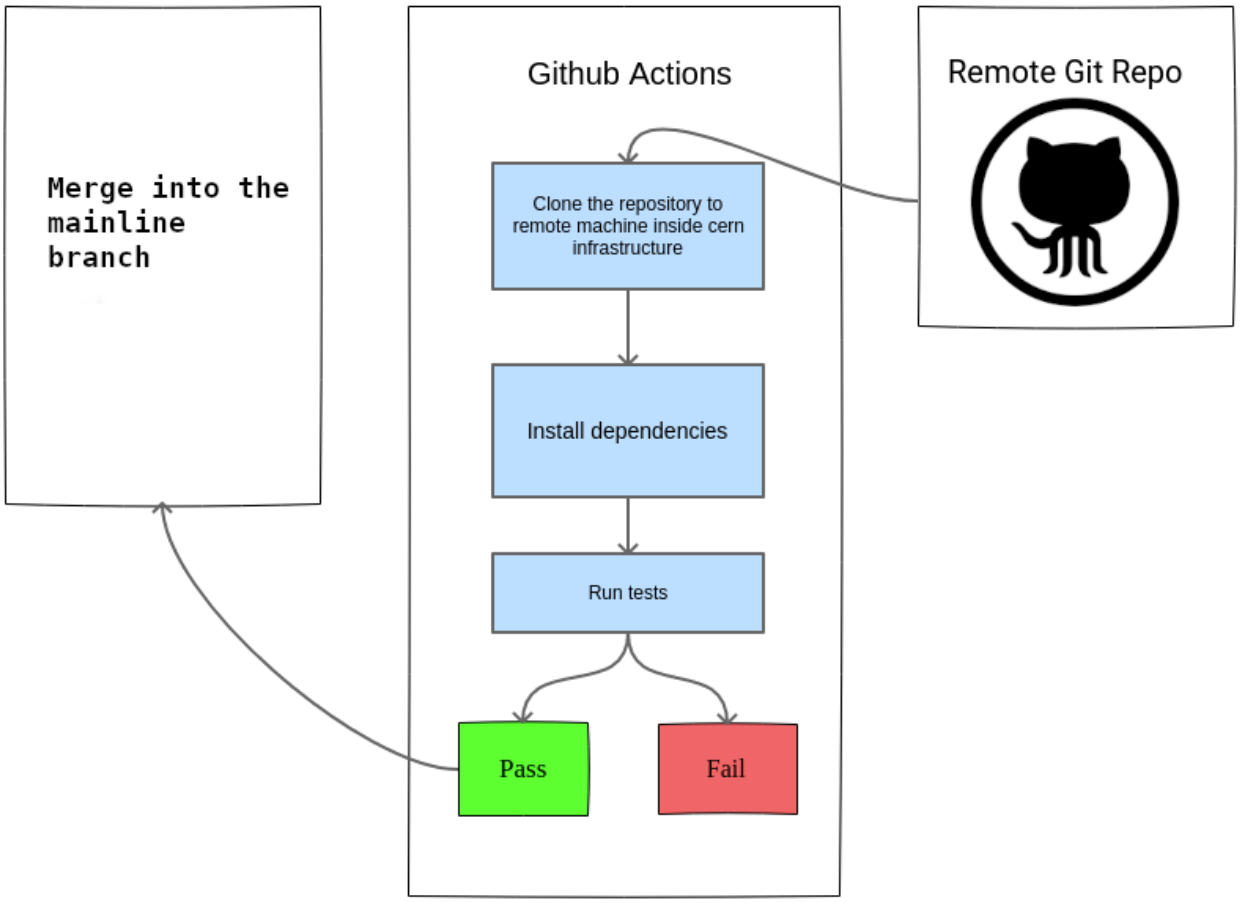
Unit tests for Unified functions

The unit tests for Unified run inside the remote virtual machines. All the unit tests were written in Python. There are primarily three kinds of unit tests: unit tests for API endpoints,
Unit tests for the database and unit tests for simple functions.
Following functions were covered with unit tests in the course of the project.

- `deep_update`
- `UnifiedConfiguration`
- `es_header`
- `url_encode_params`
- `GET`
- `getSubsciptions`
- `listRequests`
- `listCustodial`
- `listSubscirption`
- `pass_to_dynamo`
- `checkDownTime`
- `is_json`
- `read_file`
- `getWMStats`
- `checkTranferApproval`
- `getNodesId`
- `getDatasetFileLocations`
- `invalidateFiles`
- `getConfigurationFile`
- `getConfigurationLine`
- `getWorkflowCampaign`
- `agent_speed_draining`
- `getAllAgents`
- `batchinfo`
- `getNodesQueue`

Conclusion

Shellcheck[11], *pylint*[12] and *jsonlint*[14] are now integrated into the repositories of *Unified* and are being used every day to maintain a correct syntax and standard of code.
Unit tests for most of the crucial functions inside unified have been written successfully and are running on the remote virtual machine. The unit tests are run on every pull request and thus help maintain error-free code.
The next big step would be to do proper unit testing for the *Unified* modules.



General Structure of CI/CD pipeline set up for Unified

Resources

[1] CMS: Compact Muon Solenoid Experiment https://home.cern/science/experiments/cms

[2] World Wide Computing Grid: https://home.cern/science/computing/worldwide-lhc-computing-grid

[3] Unified: https://github.com/CMSCompOps/WmAgentScripts

[4] Github Actions: https://docs.github.com/en/actions/getting-started-with-github-actions/about-github-actions

[5] version control repository: https://en.wikipedia.org/wiki/Repository_(version_control)

[6] API: https://en.wikipedia.org/wiki/API

[7] pull requests: https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests

[8] issues https://docs.github.com/en/github/managing-your-work-on-github/about-issues

[9] GET: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.3

[10] CRON: https://en.wikipedia.org/wiki/Cron

[11] shellcheck: https://github.com/koalaman/shellcheck

[12] pylint: http://pylint.pycqa.org/

[13] GitHub APIs: https://developer.github.com/v3/

[14] JSONlint https://github.com/zaach/jsonlint