CERN Non-Member State Summer Studentship 2021

Report
on

CONSOLIDATION OF THE CERN ACCELERATOR BUILD INFRASTRUCTURE

Supervisor:
Brice COPY

Submitted by
Laman JALILOVA

August, 2021

1. Introduction

The below-presented report summarizes and describes separately various tools and plugins investigated, configured and tested during the 8 weeks of Summer Studentship as a part of Jenkins Clustering project.

1.1. Continuous Integration

Continuous Integration, also known as CI, is a set of practices implemented during the software development process for applying even the smallest changes in the code as frequently as possible to ensure the simultaneity and delivery of code changes reliably while working in a team.

CI helps in avoiding exceedingly many branches of the application which might conflict each other and encapsulates in itself the understanding of being built, tested and merged to a shared repository on regular basis, on every new commit.

1.2. Jenkins

Jenkins is an open source tool to automate software development processes and it is often referred as and automation server. Jenkins is the most widely used CI tool among others. Jenkins is written in Java which makes it reliable on any operating system that is supporting Java. Another reason for its popularity is the plugin support, which means that add-on extensions can be used with Jenkins.

1.3. Docker

Being a famous open-source containerization platform, Docker allows developers build, deploy, run, update and stop containers easily for a more convenient process of building, deployment and management of containers.

What primarily distinguishes virtual machines from Docker is that virtual machines have the so-called hypervisor, a layer that allows a number of operating systems to run at the same time, sharing the same physical computing resources, and the existence of guest operating systems above the host operating system makes virtual machines heavy enough. On the contrary, Docker containers can easily share the host operating system between numerous containers simultaneously which makes containers lightweight and enables users to run several applications over a single operating system kernel.

1.4. SonarQube

SonarQube which was frequently and vastly used during the Summer Studentship is an open-source platform created for the code inspection and static analysis for its quality enhancement by means of finding vulnerabilities, bugs and giving recommendations to a user about complexity, standards, code coverage and many much more that this for turning the code into a cleaner one. SonarQube supports more than around 20 programming languages and has supplementary plugins.

1.5. GitLab

SonarQube which was frequently and vastly used during the Summer Studentship is an open-source platform created for the code inspection and static analysis for its quality enhancement by means of finding vulnerabilities, bugs and giving recommendations to a user about complexity, standards, code coverage and many much more that this for turning the code into a cleaner one. SonarQube supports more than around 20 programming languages and has supplementary plugins.

2. Tools and Covered Concepts

2.1. Compatibility Matrix

Compatibility matrices are conveniently organized tools for identifying version combinations which are compatible or incompatible with each other. It us used for many purposes such as checking the browser compatibility, software compatibility, plug-ins compatibility, etc. The fundamental reason standing behind mentioning the concept of Compatibility Matrices in this report is that times to times incorrectly.

2.2. Quality Gates

When talking about quality gates in general terms, it is a specific checkpoint, a milestone that has priorly set criteria that have to be met in order to continue and proceed further to the next phase of the project. Quality gates are recognised as one of the most basic quality standards applied nowadays when working on software projects.

When integrating SonarQube with GitLab, it is highly desirable to make sure that there is a synchronization happening between the Quality Gate on GitLab and SonarQube, meaning, we expect the Quality Gate on the Gitlab to fail whenever the Quality Gate on the SonarQube fails. To put it in simple terms, we need to have a “signal” or “command” sent from SonarQube to GitLab every time we reach the Quality Gate.

To make the aforementioned happen, it is possible to apply changes directly to the *.gitlab-ci.yml* file by changing the boolean of *.sonar.qualitygate.wait* to *true*. However, this procedure itself is not the most convenient one for users since it is not a good practice doing it manually every single time. From the future perspective, it would be quite interesting and beneficial for the whole software developers community to create a plugin enabling the automatic integration of Quality Gates between SonarQube and GitLab.

2.3. CycloneDX

Programming till today has reached such a high level that the number of independent services, sub-components, etc. used for software engineering is unimaginably high. The above-mentioned services are it their turn also very often built from numerous libraries that are in some cases open source. Stem from that, as users of those libraries we can not be sure about their security and dangerousness for the project we are working on.

A bill of materials, shortly known as BOM, is simply a generated list of all pieces, parts, materials required for building or completing a certain product or action. In terms of software development, an understanding of SBOM - Software Bill Of Materials is the one with which we come across most frequently. SBOM fundamentally provides essential information about libraries used, notifies us about the ones which are out of date or deprecated, helps in finding vulnerable dependencies, and other components in a piece of software. To put it in another way, it analyses all parts of the software to warn us about components that could potentially cause damages or create obstacles in the process of software development.

One of such tools is CycloneDX which helps users define all complex and sophisticated components of a software. It's tools allow us to generate SBOMs for projects written in various programming languages. During Summer Student Programme 2021 I was responsible for investigating CycloneDX for Python which is one of the most frequently used programming languages. An input file for CycloneDX for Python was *requirements.txt* file which was generated automatically from `setup.py` file using `pipreqs` command. An output was `bom.xml` which indicated all components such as a publisher, name, version, description, hashes, licenses, purl, etc.

It goes without saying that manually inputing every single time a file to CycloneDX and going through all output files takes much time and energy of developers. Here comes the question regarding how to manage that output, how to keep track of evolution and monitor the progress.

The answer to all of the previous questions turned out to be a Jenkins plugin Dependency-Track which enables users and organizations to identify and reduce risk from the use of third-party and open source components that will have previously been produced by CycloneDX. Dependency-Track is in charge of continuous monitoring, producing real-time analysis and presenting it in a form of appealing and user-friendly charts and reports which makes the whole process of software development much easier and convenient.

2.4. CXX Plugin

Taking into account that SonarQube is mainly utilized for enhancing code quality, its CXX plugin serves for integrating C++ tools for warning about potential vulnerabilities. On a par with

Based on personal experience with Sonaq-cxx plugin, the following information may serve as a useful tool for properly configuring it. While setting the plugin up, many difficulties came up along the way and no documentation was useful for resolving the issue. Eventually, it turned out that the problem was old `.jar` files in `*extensions/plugins`. They have to be replaced with updated versions from `lib/bundled-plugins`: `docker run -ti -d -p:9000:9000 -v ``pwd``/sonar-cxx-plugin-2.0.4.2806.jar:/opt/sonarqube/lib/bundled-plugins/cxxplugin.jar sonarqube`

2.5. Authentication vs. Authorization

In simple terms, authentication is the process of verifying who a user is, his or her identity. On the contrary, authorization is known as the process of verifying what users have access to, not who they are, unlike authentication.

Usual procedures at the airport can be brought as the brightest example of the difference between authentication and authorization. The first thing one does when going to the airport is showing ID card so that one is recognized - it is authentication. Hereafter, one proceeds further towards the gates where one is requested to present his or her boarding pass depending on which one can either be let to enter the plane or sent back because of not having access - it is authorization.

Several technical differences are that authentication is normally done before authorization and authorization is done after successful authentication; authentication is generally transmitting information using an ID token, while authorization utilizes Access toke for these purposes.

3. Conclusion

During CERN Summer Student Programme 2021 numerous aforementioned tools have been investigated, tested and analyzed. It would be beneficial not only to CERN software engineers community for easing the development process, but for me as a student too, since the amount and quality of knowledge acquired during the past 8 weeks while working on a project, attending team meetings and lectures were unmatched by any other learning source.