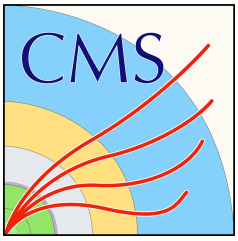
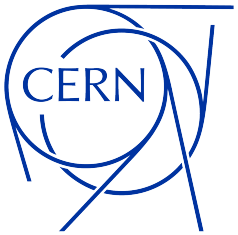
# Implementation of new DevOps practices in the CMS level-1 trigger online software





Chrisitan Winter (KIT)
christian.winter3@student.kit.edu

Supervisor:
Simone Bologna (CERN)
simone.bologna@cern.ch

July 5, 2021 – September 10, 2021

CERN – Summer Student Programme 2021

# Contents

# 1 Introduction

Nowadays, no large-scale physics experiment can be designed without automated data acquisition systems. These systems are controlled and monitored by software referred to as online software. The online software of the CMS level-1 trigger consists of a number of different projects, each one maintained by a separate team. With such a complex and diverse system it is not easy to successfully work collaboratively. Therefore, common libraries have been written so that only the individual settings differ. Following the efforts to make this system less error-prone, new procedures based on the DevOps methodology were introduced for the CMS level-1 trigger online software as part of the Summer Student Program 2021.

First, this report explains in section 2 some basic terms and introduces techniques and tools that have been used in this work. Then in section 3 the previous system is described and what the problems were. Then in section 4 the developed solutions are presented. Finally, in section 5 the difficulties in this project and also the open works for the future are discussed.

# 2 Base Knowledge

## 2.1 Trigger

The trigger is a fundamental component of the data acquisition of any modern physics experiment. The trigger receives data from the experimental apparatus to determine when an interesting event has detected and initiate the data acquisition system.

Triggering at the Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC) accelerator at CERN is a very difficult task. Here, there are different sensors and detector systems that can record different data, and, furthermore, the generated raw data volume is so large ( 50-100 TB/s) that it is impossible to store it all. Therefore, the trigger consists of a hardware based trigger followed by a software based trigger, which filters the events passing the first hardware based trigger more precisely.

The hardware based or level 1 trigger (L1T), which is relevant for this work, is organized into subsystems, which are each responsible for a detector system or which combine the signals of the different triggers. The trigger subsystems are shown in Figure 1.

To have a common platform and to be able to communicate with each other, the L1T online software projects are based on common libraries and are developed with the same framework. The software stack is shown in Figure 2. The software for trigger subsystem is called SWATCH cell and is managed in the CERN GitLab[1].

The hardware trigger boards communicate with the software via the normal network. This network is isolated from the outside to prevent external connection, and, for security reasons, can only be accessed via a proxy.

## 2.2 DevOps

DevOps is a group of techniques for developing and operating software. The goal is to improve the quality of the software. Particularly important is the continuous succession and automation of the steps. In Figure 3 is to be seen which steps a software must run through according to the DevOps approach.

For this project, two techniques in particular were used to automate some steps. One is continuous integration (CI) and the other is continuous deployment/delivery (CD).

[1] `https://gitlab.cern.ch/cms-cactus/projects`

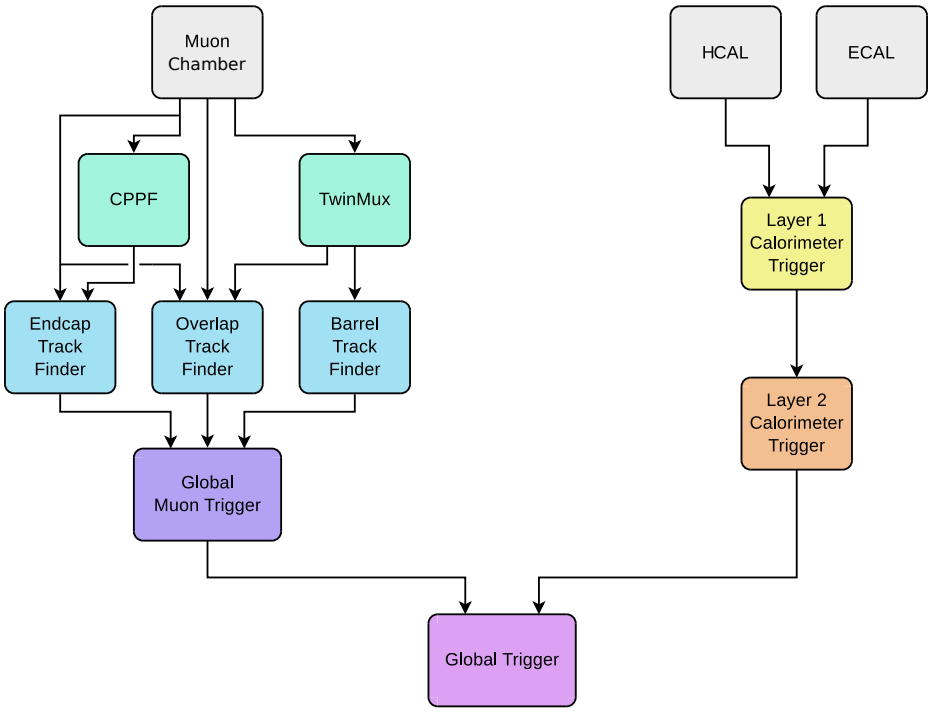



Figure 1: Shown here are the various trigger subsystems and how they are related. The left branch is responsible for the muon chambers and the right branch for the calorimeters.

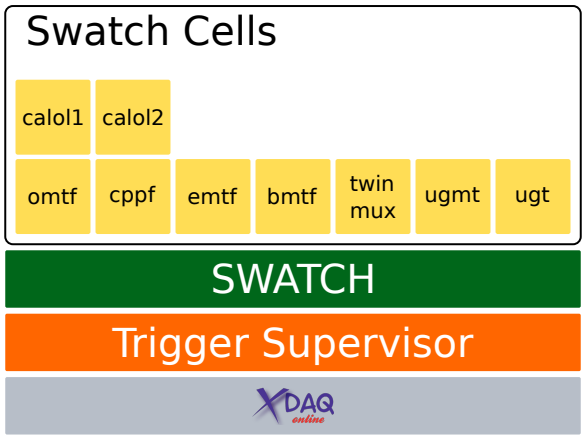



Figure 2: The software stack of all SWATCH cells. All are based on XDAQ which is a library of tools for developing data acquisition applications. With this, the Trigger Supervisor was developed which, among other things, enables additional communication channels between the SWATCH cells. On top of this comes the SWATCH software which provides a graphical user interface, accessible via the web browser, for the end user and also includes libraries for communication with the hardware boards.

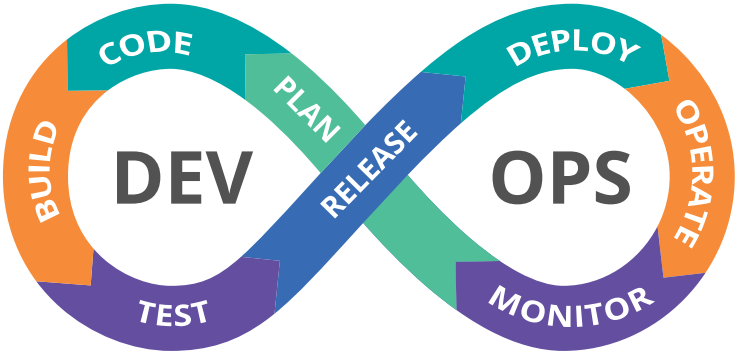



Figure 3: The DevOps circle describes the steps that software must go through according to DevOps philosophy. The more steps are automated, the better.

CI describes the automated building, i.e. compiling and linking, and testing of the software. That is, as soon as a code change has been committed, a pipeline should start, which first builds the software and then tests it. If a process in the pipeline fails, then the developer is notified, and he must reconsider his code change. Thus, many errors and bugs can be found immediately. Also, building the software is often tedious and can be done automatically in the background.

CD goes one step further and adds a step in the pipeline that publishes the built and tested software. This must not only mean publishing to the end user, but also publishing for test or demo purposes. For this to work, the pipeline must have access to a server or a server cluster. It is very convenient for this process to package your software in Docker containers and publish it using a cluster management program. This is described more in subsection 2.3 and subsection 2.4.

## 2.3 Containers and Docker

Since the last decade, more and more software developers have been packaging their software in so-called containers. Especially Docker containers are very popular, as they are a kind of industry standard. Containers have the great advantage that the software already comes with all the necessary libraries and the container only needs to be started with a container runtime service. The installation of additional dependencies becomes unnecessary. In addition, the programs within the container run in a sandbox. If a program crashes, then only the container crashes and not the entire operating system. Potential attackers from outside also have a harder time getting onto the computer if they first have to break through the container's sandbox. In this project, too, the software is packed into containers. So that the testing, deployment and operating of the software becomes easy

to automate.

## 2.4 Kubernetes

If a team of developers wants to publish their software, then usually not only on one but on several servers, which are located at different locations. This is then also called a server cluster. A cluster has several advantages, such as security against failures, load balancing, more performance and the possibility to seamlessly install updates without having to completely take the service down. Manually installing the programs on all servers in the cluster is of course very tedious and time-consuming, for this there is cluster management software like Kubernetes.

Kubernetes allows software packed in Docker containers to be deployed, managed and monitored simultaneously on a server cluster. Kubernetes was developed by Google to deploy and manage their own programs on their servers.

# 3 Previous Condition

As already mentioned in subsection 2.1 there is not only one trigger software but many so-called SWATCH cells which are developed by different teams. Since these teams are located internationally, it is not always easy to work together collaboratively, so that all programs work together afterwards. There are also developers for the SWATCH project, the Trigger Supervisor project or for XDAQ. In the past there were often made none or too little tests when one of the involved software changed.

The following scenario serves to identify where the problems occurred. For example, the SWATCH developers release a new version of SWATCH and ask the SWATCH cell teams to update it on the servers. The following steps are now necessary.

1. The software must be compiled with the new dependencies and packaged.
2. The packages must be copied to the server in the CMS network.
3. The packages must be installed on the server in the CMS network.
4. SWATCH software must be restarted.
5. Finally check if the software is running properly.

Now problems can occur at each step.

1. When compiling or packaging the software, parameters are not set or components are linked incorrectly. Libraries may also be missing.
2. The packages are forgotten to be copied, and old packages are installed again.
3. Not all or old packages are installed or errors occur because of missing libraries.
4. Forget to restart the software.
5. Not checking if the software is running as it should.

This sequence must be manually run every time a new deployment is performed. Since the procedure can be tedious, deployments may fail due to distraction. Since these steps are always similar, it is also possible to automate these steps. Techniques such as continuous integration (CI) or continuous deployment (CD), which are described in subsection 2.2, can be used for this. The GitLab platform, where the code is also stored and managed, offers

the possibility to create such CI/CD pipelines. Before the project, the L1T online software team had already started and added a step to the pipeline that builds and packages the project as soon as a code change is uploaded. This way the errors in point 1 can't happen anymore.

The goal and task of this project is to add additional steps to the CI/CD pipeline that automates, packages the software into Docker containers, tests and then publishes to a server cluster.

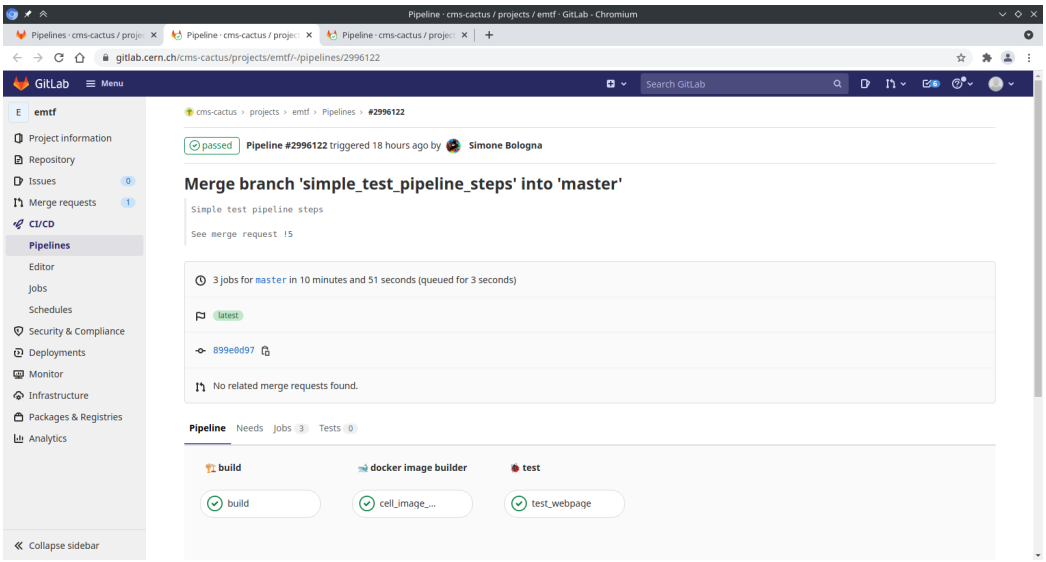



Figure 4: This screenshot shows how GitLab graphically represents a pipeline. In this pipeline there are three stages and each stage has one step. The stages are processed one after the other.

# 4 My work

As already explained in the previous section 3, the task in this project is to add additional steps to the CI/CD pipeline. The task is divided into two parts. One part that covers the CI steps and one part for the CD steps.

GitLab was used to create this pipeline, because it is also used to manage the code. The configuration of the pipeline works via YAML files, which can be found in each project under the name "`.gitlab-ci.yml`".

## 4.1 CI Pipeline

There was already a step that compiles and packs the projects. In this project, two additional steps were added. The first one automatically creates functioning Docker containers of the SWATCH cells after the software has been compiled and packaged. The second starts this Docker container and tests if the GUI is reachable and what HTTP status code is returned. If the pipeline terminates with an error message, the developer who uploaded the code changes is notified via email.

GitLab provides a graphical representation, shown in Figure 4, of such a pipeline, where

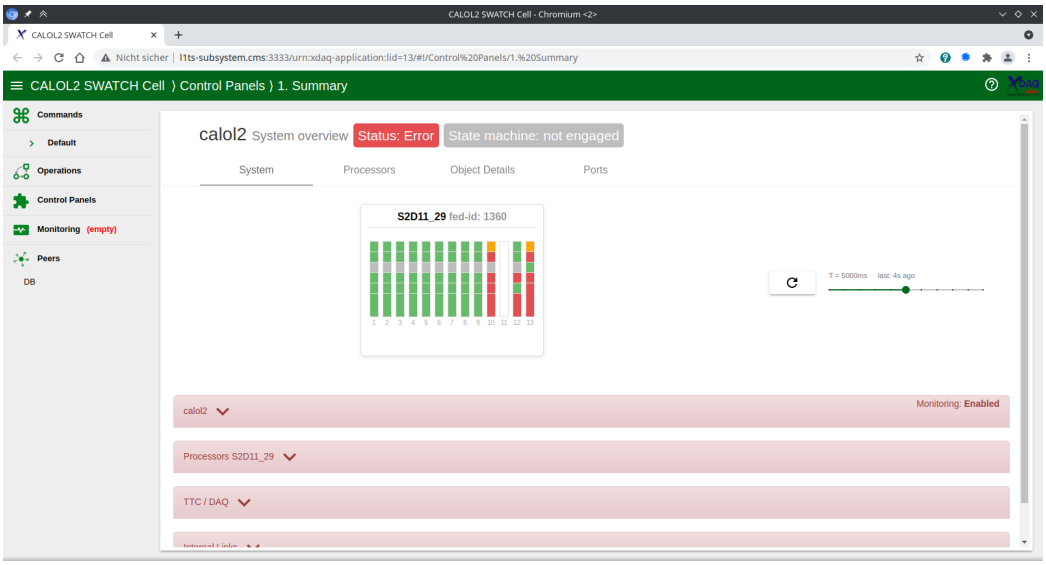



Figure 5: Here you can see the graphical user interface (GUI) of the SWATCH cell of the calorimeter layer-2 trigger subsystem. This GUI was provided by the SWATCH cell in a Docker container. The container was started on a server in the CMS network and was able to successfully connect to the hardware boards.

you can observe how the steps are processed one by one.

Normally, the SWATCH cell software would only run within the CMS network. Therefore, when creating the container, the configuration and profile files are modified slightly, so that the software also starts outside of CMS network. Of course, it can't connect to the hardware trigger boards and the display is very limited. Nevertheless, the test can be used to find many errors, which cause the software or the graphical user interface to not start correctly.

As a test, the container was started on a server in the CMS network and as shown in Figure 5, this was successful and the software in the container could connect to the hardware trigger boards without problems.

## 4.2 CD Pipeline

One advantage of containerized software is that it is easy to launch a container anywhere. Unfortunately, in this case it is not possible to do this in an automated way, as due to security reasons GitLab does not have access to the encapsulated CMS network where the containers should be started. Therefore, for this project a server cluster was created in the CERN network, which is similar to the one in the CMS network. The cluster is set up

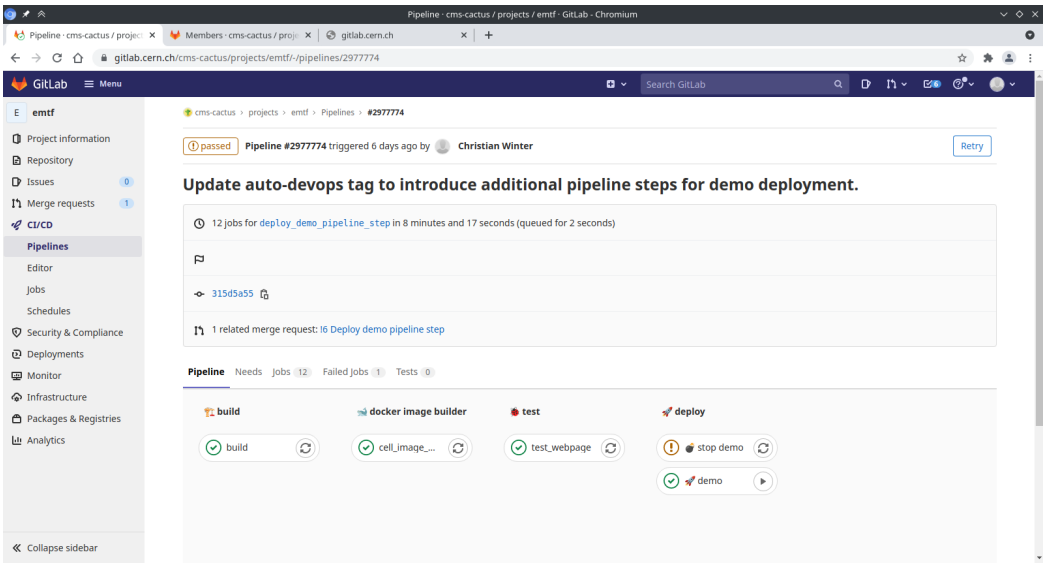



Figure 6: This screenshot shows the graphical representation of a GitLab CI/CD pipeline. The new deploy stage has two steps, which deploy the software to a Kubernetes cluster or stop the deployment. Both tasks must be triggered manually by pressing the button next to the step.

and managed with Kubernetes.

Now an additional step has been added to the pipeline, which allows to deploy the software to the Kubernetes cluster with only a click of a button in GitLab. In Figure 6 you can see the additional stage with the two added steps. One step deploys the software to the Kubernetes cluster, the other one stops the deployment. Both steps have to be started manually by pressing a button and therefore happen only semi-automatically.

Kubernetes decides under which URL the SWATCH cell website can be found. To allow developers to access the website without looking it up in the Kubernetes configurations, GitLab environments have been introduced. These extend the GitLab user interface with additional buttons and allow the user to open the corresponding website or to stop the deployment. The screenshot in Figure 7 shows the overview page of such an environment. The buttons "`View deployment`" and "`Stop`" are important here: the former is a link to the web interface of the newly-deployed SWATCH cell, the latter stops the application. Also, during a merge request you can now experience the program in operation before you merge it, because there are additional buttons now as well, as Figure 8 shows. This demonstrates that even if not published to the CMS network, the Kubernetes cluster is still very useful for testing the software.

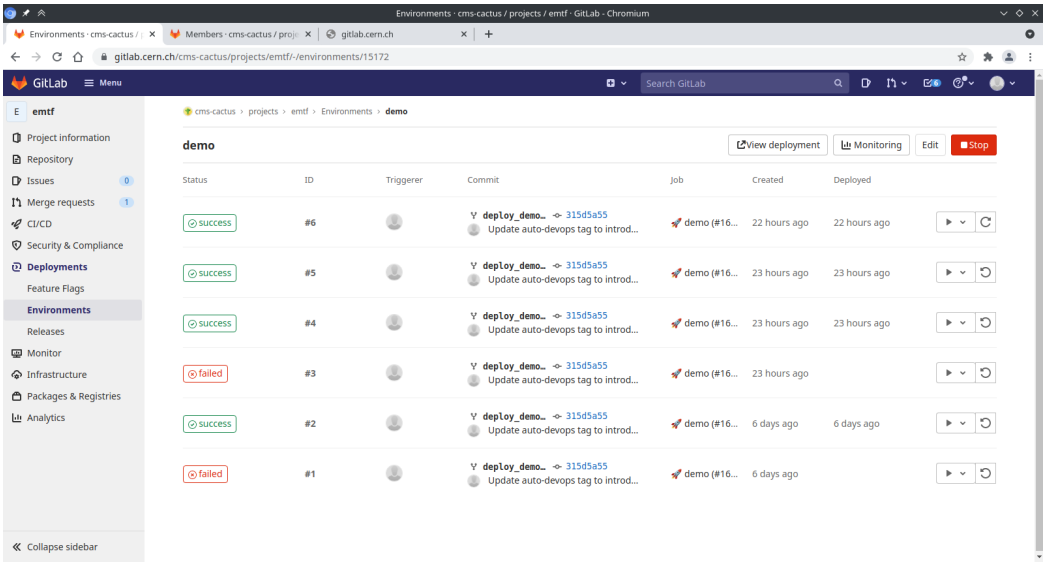



Figure 7: Here the GitLab overview page of an environment is shown. Important are the buttons on the top left which open the corresponding website or stop the deployed software. In the middle, the pipelines that led to a deployment in this environment are listed.

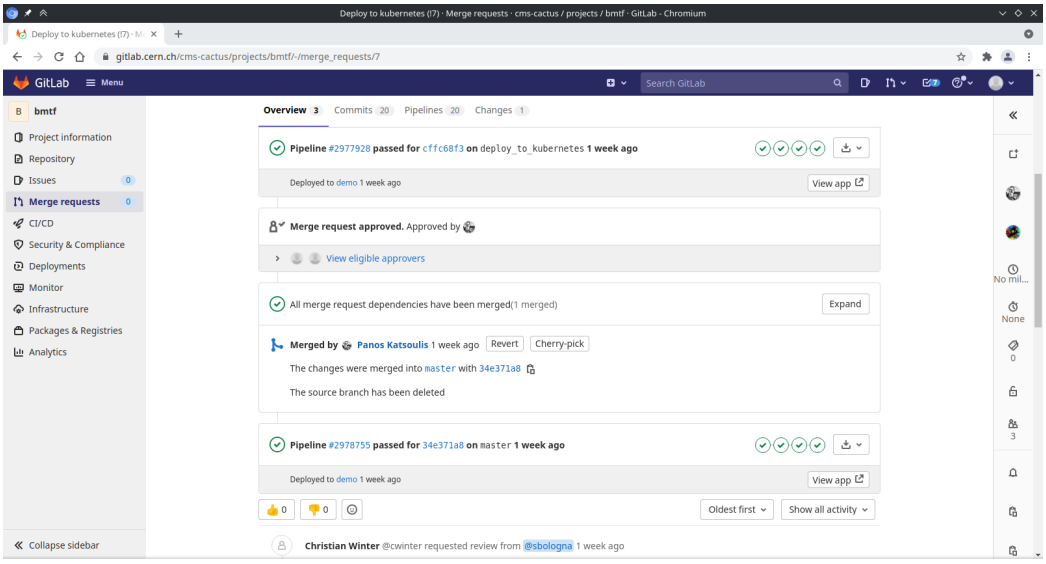



Figure 8: This GitLab merge request shows additional buttons "`View app`" before and after the merge procedure. This allows developers to directly see the changes they have made using the launched software.

# 5 Conclusions

With the steps presented, the software was made less error-prone. In review to the problems that can occur (see section 3) only the problems that are listed in point 1 and 3 could now be solved. The other problems can only be solved when there is a possibility to deploy into the CMS network in an automated way. For this however, a lot of preliminary work has been done, and the automated distribution works at least in a comparable Kubernetes cluster in the CERN network. In addition, new automated tests have been added, which also increase the code quality.

## 5.1 Difficulties

Of course, not everything went smoothly in the course of the project and I had some difficulties. The biggest problems I had at the beginning were with the software architecture and the structure of the GitLab CI YAML documents. These are distributed on several projects and are based on each other. At the same time you have two parallel structures, that of the software code and that of the GitLab CI documents. What would have helped me a lot here would be a diagram of how the projects are connected. Unfortunately, I have not yet managed to do this myself.

Later I had problems again when I wanted the Kubernetes (or OpenShift, which I wanted to use first) cluster to access private GitLab projects. Finally, I was able to solve this with Kubernetes and a manually created GitLab Deploy Token.

## 5.2 Outlook

As already mentioned, the work here is not done yet. It would be important that it is possible to start the created Docker containers in the CMS network automatically to prevent even more errors. This doesn't work from outside the CMS network, but it may work from the inside. With the help of regular checks within the CMS network to see if new containers have been built, the software can also be kept up to date.

What also needs to be changed is the way new containers are published. Currently, an old container on the cluster is terminated and deleted, and then the new one is published along with the associated Kubernetes infrastructure. However, Kubernetes offers the possibility to make the container change without a break by simply changing the source address of the container in the Kubernetes publication to the new address. Then Kubernetes

automatically pulls the new container and starts it without having to rebuild the whole infrastructure around it.

In addition, Kubernetes offers the possibility of automated tests in the published software. These can check, among other things, whether the website of the software is accessible. If this is not the case, Kubernetes automatically restarts the container or rolls back to an old container version. However, these tests can be used even more universally and can be adapted entirely according to the requirements. In general, as soon as the software can be published to the CMS network, more tests can be performed that also take the hardware or the connections to other subsystems into account.

## 5.3 Closing Thoughts and Acknowledgements

I am very satisfied with the project and what I have achieved. I learned a lot, and I am also strongly convinced that DevOps will become even more important in the future. Also, I had the rare opportunity to work with a large network and try out the programs on it.

I would also like to sincerely thank my supervisor Simone Bologna, who gave a lot of time and effort to me. He was always able to help me when I turned to him with my questions and problems. I would also like to thank all the other people involved in the CERN Summer Student Program, who provided this great opportunity for me. Thank you very much, I had a lot of fun.