

Implementing Continuous Integration Pipelines

A Case Study for Mobile Applications using GitLab

Maha Asfour¹ and Alexandre Bragança²

¹ Institute of Engineering of Porto - Polytechnic of Porto (ISEP/IPP), Portugal
1181475@isep.ipp.pt

² Institute of Engineering of Porto - Polytechnic of Porto (ISEP/IPP), Portugal
Interdisciplinary Studies Research Center (ISRC), Portugal
atb@isep.ipp.pt

Abstract. Recently IT companies seek to automate the release lifecycle of software products, such as building, testing and deployment, to provide better quality and faster releases. To achieve this purpose, Continuous integration and continuous delivery tools were used.

This paper presents the results from the case where GitLab CI was used to improve a software company's existing continuous integration pipelines for mobile applications.

Keywords: DevOps, CI/CD pipeline, Metrics system, Mobile Applications.

1 Introduction

Before 2008, the software lifecycle suffered from slow progress and disorganized workflows, resulting in slow and the unreliable releases.

Then the DevOps concept (development and operation) was born to solve these problems by improving collaboration between all stakeholders throughout the process from planning to delivery. As a result, according to DevOps 2015 Status Report, "High-performance IT organizations deploy maximum 30 times with short lead times of up to 200 times; they have 60x less failure and 168x faster recovery" [1].

Some related practices also showed like Continuous Integration (CI), which automates the integration of code changes from multiple contributors into a single software project. Continuous Delivery (CD), which is an extension of Continuous Integration to ensure that the developer can quickly and sustainably deliver new changes to the customers quickly in a sustainable way.

This project addresses some of the problems of the CI/CD pipelines for the mobile applications, such as the long runtime of the pipelines, which means a delay in the release-lifecycle, and the lack of a metrics system to monitor the performance of the pipelines.

ISBN: 978-989-54758-6-5

The proposed solution is based on replacing Jenkins and Buddybuild with Gitlab CI and building new infrastructure with dockers, AWS services, and Kubernetes for the Android project, and maintaining the self-hosted infrastructure for the IOS project.

The structure of the paper includes an introduction describing the problem and the proposed solution, Related Work includes related projects and a comparison table for CI/CD tools, Context describes the project, and the goals, Continuous integration with Gitlab includes the CI/CD pipelines, and the proposed scenarios for implementing the solution, Evaluation to assess the final solution, and Conclusion then References.

2 Related Work

In this section, several related projects and ten CI/CD systems were examined to find the optimal solution depending on the requirements of the projects.

The Glintt web application project [2] used TFS and Jenkins, but the solution causes unclear problems when multiple developers merge their changes at the same time, resulting in delayed releases. Reducing the length of the release-lifecycle is the main goal of this paper.

In Shopping List application [3]. The main purpose was to create a CD pipeline for the ReactNative application, ReactNative being a framework for building IOS and Android Apps [4]. Circle CI was chosen to implement the solution, but it does not provide a self-hosted option for IOS projects[5] which is an important requirement in this paper, and also the non-use of the docker image means that time is wasted each time installing the environment requirements.

Piceasoft mobile applications [6] were missing the automated builds, tests, and releases.

To reduce the manual effort required, Gitlab CI was chosen to implement the CI / CD concepts, and as a result, the project became fully automated and the time required to integrate and release the application was reduced.

while this paper aims to improve the performance of the application's CI/CD pipeline without implementing continuous deployment.

The optimal tool for this project was found thank for the comparison table1 that contain the project's requirements and latest CI/CD tools in the market.

Although Teamcity successfully covered all necessary requirements, it is a complicated and hard to understand tool and requires a lot of settings to configure each project, which consumes the time of the teams. For this reason, Gitlab CI was chosen to work on the project.

Table 1. comparison table of CI/CD tools based on the company's requirements

	Jenkins	Buddy	TravisCI	Teamcity	CircleCI	BuddyBuild	Bitrise	Drone	Bamboo	Gitlab
Support Android projects	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓
Support IOS projects	limited infrastructure	✗	✓	✓	✓	✓	✓	✓	✓	✓
Support hosting on-premise	✓	✓	✓	✓	limited with IOS projects	✓	✓	✓	✓	✓
Git VCS	✓	✓	Just Github	✓	Not integrated with Gitlab	✓	✓	✓	✓	✓
AWS EC2 or EKS infrastructure	✓	✓	✓	✓	✓	✗	✗	Just ECS	✓	✓
Metrics system	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Jobs Parallelism	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Support Docker	✓	✓	Not supported Docker with Mac OS	✓	✓	✓	✓	✓	✓	✓
Friendly GUI	✗	✓	✓	✓	✓	✓	✗	✗	complicated	✓

3 Context

This work prepared during an internship for IT company works on developing and maintaining 11 mobile applications to Tourism company, and this project includes the automation of builds, quality assurance testing, unit testing and the release of the applications using CI/CD pipelines.

The main goal is to improve the performance of these pipelines using Gitlab CI and a new infrastructure, with the DevOps team interacting with the system by adding the CI/CD pipelines and improving their performance by developing them while the servers run these pipelines. As shown in Fig.1.

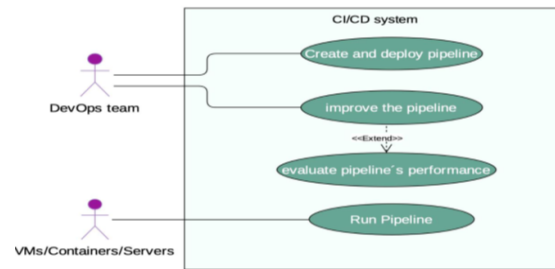


Fig. 1. Use case diagram for CI/CD system

4 Continuous Integration with GitLab

The Android pipeline starts with compiling and generating the APK package (JAR Android Package), then checks the code quality with three jobs running in parallel, Lint to scan the source code for potential bugs [7], Check style to check the Java code standards [8], and Detekt to analyze the Kotlin code [9], then the unit testing phase to check the logic of the units in the source code [10], then upload the application to Appcenter [11] so the QA team can test the application. As shown in Fig. 2

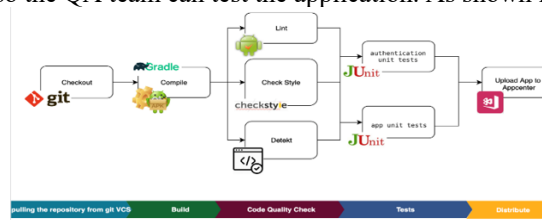


Fig. 2. CI/CD pipeline for the Android application

The IOS pipeline starts with two jobs in parallel, one of them to compile and test specific project of the 11 apps, and the other to compile and generate IPA package (iPhone Application Archive) for any of the 11 applications, then Code quality stage to check the maintainability, reliability, security and generate reports that define if the low quality causes failed cases [12], then uploading the application to Appcenter. As shown in Fig. 3

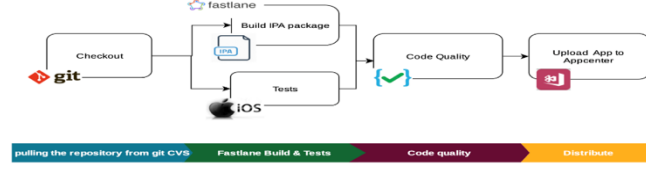


Fig. 3. CI/CD pipeline for the IOS application

Fig. 4 is the component diagram, which shows the interactions between the system components and the responsibilities of each component. For example, Gitlab is responsible for storing source code, pipeline script, artifacts, and analytics charts, Docker and AWS components run the Android pipeline, while Mac servers run the IOS pipeline and Appcenter stores beta versions.

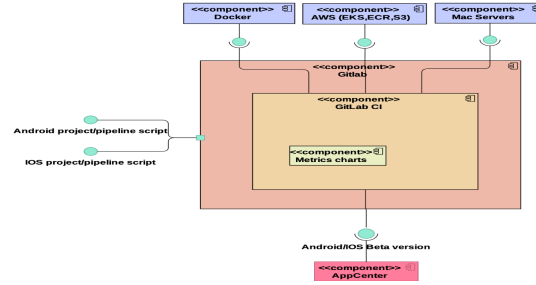


Fig. 4. Component Diagram for CI/CD tool

Gitlab uses runners to run pipelines, these runners are the infrastructure of the system. Three architectures were designed for the Android project, the first is based on Shared Runners provided by Gitlab CI, the second uses AWS EC2 instances as runners, and the third (adopted as final solution) is based on the Docker image, to prepare the Android environment and the Kubernetes cluster (AWS EKS service), where it is a group of runners that provides the ability to delete/create runners and increase/decrease memory on behalf of the user, which helps to reduce costs and avoid memory limitations. Meanwhile, the IOS project has only found an architecture that uses the available physical servers in the company as runners.

5 Evaluation

According to the Gitlab metrics system charts, the runtime of the Android project. For 10 builds per day with one commit, Gitlab needed about 144 minutes per day, while Jenkins needed about 310 minutes per day, and for the time of the IOS pipelines, Gitlab needed about 13 minutes for each build and test, while Jenkins needed about 18 minutes for each test, and BuddyBuild needed 31 minutes and 29 seconds for each build since the infrastructure of both projects with Gitlab is similar to the previous state with Jenkins and Buddybuild.

According to the results of a survey published among developers within the company, all team members think Gitlab, unlike BuddyBuild, has achieved usability and,

unlike Jenkins, has a friendly GUI, while most believe extensibility has been achieved and no one is against adopting it instead of the old tools.

6 Conclusion

This paper dealt with a project at an IT company for the maintenance of CI/CD pipelines for mobile applications, where the pipelines suffered from a long runtime and needed a metrics system to monitor their performance. Besides, the managers wanted to replace the current CI/CD tools with only one tool.

To achieve these goals, Gitlab CI was selected to implement the solution after a study of the latest CI/CD technologies. Considering the company's resources and server costs, EKS AWS & Docker Image solution was selected as infrastructure for the Android project, while the physical Mac servers were used for the IOS one.

To evaluate the speed performance of the pipelines, the Gitlab Metrics system showed the Pipelines times graph, where the Android pipeline in Gitlab is twice as fast as the pipeline in Jenkins and the IOS pipeline is so much faster than the old pipelines in Jenkins and BuddyBuild.

To evaluate more factors, a survey was published for the developers, in which they think that the new solution is easy to learn and use (usability), has a friendly GUI and offers the possibility to add more projects and runners (extensibility).

Although the positive results, this project can be developed later by using Terraform to have infrastructure as code and implementing the continuous deployment concept.

References

1. PuppetLabs. 2013. "2013 State of DevOps Report," no. July 2015: 1–12. <https://doi.org/10.13140/RG.2.1.3890.2645>.
2. Ramalho, Francisco. 2018. "Continuous Delivery: Aplicação Na Glintt." https://recipp.ipp.pt/bitstream/10400.22/12575/1/DM_FranciscoRamalho_2018_MEI.pdf (2018).
3. Ezugbaya, A. 2019. "Continuous Deployment for Cross-Platform Mobile Application," no. April. <https://www.theseus.fi/handle/10024/227482> (2019).
4. React Native, <https://reactnative.dev>. Last accessed 11 June 2020
5. Circleci.com, <https://circleci.com/docs/2.0/executor-types/#using-macos>. Last accessed 12 June 2020
6. Bitstream, <https://trepo.tuni.fi/bitstream/handle/123456789/27193/Salminen.pdf?sequence=4&isAllowed=y>. Last accessed 2 February 2019
7. Lint, <http://tools.android.com/tips/lint>. Last accessed 20 June 2020
8. Checkstyle, <https://checkstyle.org/index.html>. Last accessed 20 June 2020
9. Android, <https://mindorks.com/android/store/Static-Analysis-Tools/arturbosch/detekt>. Last accessed 20 June 2020
10. Unit-testing, <https://developer.android.com/training/testing/unit-testing>. Last accessed 20 June 2020
11. App-center, <https://visualstudio.microsoft.com/app-center/faq/>. Last accessed 20 June 2020
12. diving-into-code-quality-factors-affecting-code-quality, <https://www.disputesoft.com/diving-into-code-quality-factors-affecting-code-quality/>. Last accessed 20 June 2020