Comparing Modern Build Automation Tools for an Insurance Company

A. Koschel, K.C. Tran, A. Grunewald,
M. Lange, A. Pakosch
Hochschule Hannover, Univ. of Applied Sciences and Arts
Hannover
Germany email: akoschel@acm.org
I. Astrova
Tallinn University of Technology
Tallinn
Estonia
email: irina@cs.ioc.ee

June 13, 2023

Abstract

In this paper we describe the selection of a modern build automation tool for an industry research partner of ours, namely an insurance company. Build automation has become increasingly important over the years. Today, build automation became one of the central concepts in topics such as cloud native development based on microservices and DevOps. Since more and more products for build automation have entered the market and existing tools have changed their functional scope, there is nowadays a large number of tools on the market that differ greatly in their functional scope. Based on requirements from our partner company, a build server analysis was conducted. This paper presents our analysis requirements, a detailed look at one of the examined tools and a summarizes our comparison of all three tools from our final comparison round.

Keywords: build automation, build server, DevOps, tool evaluation, CI/CD

1 Introduction

Over time software became increasingly more complex and distributed. Proportionally, it became more and more difficult for developers to compile, link, package and test software projects with their dependencies manually. Especially with the rise of cloud native development using microservices (cf. CNCF: https://www.cncf.io/) and the large number of involved artifacts, build automation became nearly a 'must have'.

For this reason, build automation tools like Make, Ant, Maven, and Gradle have emerged. Based on this, build servers have been developed. A build server is a system that checks out the current project status from a version control system (VCS) at certain times or events and then builds the project using a build automation tool described above.

If the build server responds to each check-in to the VCS, it is called Continuous Integration (CI). Build servers usually perform other tasks than just building the software, such as code analysis or automated testing to detect execution or quality issues early and ensure the stability of the software. Some build servers can even publish the built software. A distinction is made between publishing in a staging environment (Continuous Delivery (CD)) and publishing in a productive environment (Continuous Deployment (CD)) [1].

With the emergence of topics such as microservices and DevOps, build automation and in particular CI/CD is experiencing a new upswing. Small deployment units that are constantly built, tested, integrated and deployed, help significantly to enable continuous releases of – often more stable – software. As a result, a large number of build servers are currently available, which differ widely in their functional scope [1].

This article is based on a cooperation with a research partner from the insurance industry. While they operate successfully a medium scale service-oriented architecture (SOA), as well as mainframe and SAP applications, for newer demands – such as intelligent insurance risk predictions, and spontaneous insurances – they aim at microservices based solutions. In previous work [18] we had a look at microservices for such purposes. We also aim at working towards a 'Microservice Reference Architecture for Insurance Companies (RaMicsV)' jointly with our insurance industry partners [19], which gives an overall context for our work here.

However, especially with microservices a strong demand for a modern build system with good CI/CD support arrises. (Also) for that purpose the research partner is currently modernizing its build system, but is confronted with the large variety of tools. As part of our project work, a comparison of modern build servers was conducted. The results of this comparison, with – due to space limitations – an in depth look at (only) one evaluated build server, are the main contributions of this article (see [20] for a second in depth build tool evaluation of ours, which of course builds on the same evaluation process and uses the same evaluation criteria as here in this article).

The remainder of this article is organized as follows: Section 2 briefly looks at related work. Next Section 3 discusses our approach and shows how we have selected and compared different build server solutions. Section 4 evaluates in depth one particular build server, namely Travis CI, using the criteria from Section 3. Section 5 explicitly compares two evaluated build servers, and Section 6 gives a conclusion and some outlook to future work.

2 Related work

In this section we will have a brief look at some work related to ours, thus in particular at some general overview but also at particular tool evaluations for CI/CD.

Meyer gives in [21] an overview on CI and its tools', which is helpful for a general look at the topic. More strategically oriented, Jin and Servant discuss in [22] strategies to improve CI.

A systematic overview on continuous integration, delivery and deployment, which includes approaches, tools, challenges, and practices is shown by Shahin, Babar, and Zhu in [23]. Singh, Gaba, M. and B. Kaur provide a cloud platform oriented comparison of different CI/CD tools in [24]. Probably closest to our work, although from 2015, is [25] by Rai et al. They look at Jenkins with a comparative scrutiny of various software integration tools.

However, none of the articles follows the particular evaluation steps and criteria as they were emphasized by our partner company, which is a good example for (at least the German) insurance business. This, thus, is the major contribution of our work here, although might be generalizable and be of value for at least some other insurance companies as well.

3 Comparison – Our Approach

In order to find suitable solutions for the partner to modernize its build system, our comparison approach is divided into five steps. They are explained below in chronological order.

Step 1: Restricting the Subjects for Comparison After a training phase we have created a list of six build servers with a rough overview of each one. Afterwards we presented the list to the partner and in several discussions we reduced the number of subjects to three.

Jenkins [2,3,4,5], which is currently used by the partner, Travis CI [7,8,9, 10,11,12,13], and GitLab CI [14,15,16, 17] were selected for the comparison. Due to space limitations in this article we will only present our in depth evaluation results for Travis CI. Our in depth GitLab CI evaluation is presented in [20] and the in depth evaluation of Jenkins is presented [26].

Step 2: Analysis of the Research Subjects After the build servers were identified, we assigned a project participant to each tool to take a closer look.

Step 3: Development of Comparison Criteria After each project participant has become an expert with a specific build server, we worked out a number of possible criteria to compare the products. Together with the partner, we discussed and prioritized the criteria. We have identified the following four categories of comparison criteria by which the build servers are compared and analyzed:

- Internals: In this category we roughly look at the architecture of each system and we check, how the build process works internally.
- Pipelines: This category deals with the topic of pipelines. It is analyzed how a developer can model the build process in each tool, i.e. how pipelines can be created. Furthermore, the pipeline-relevant features (e.g. an artifact store, image store or caching in the pipeline) of each tool are discussed in this category.
- Additional Features: This category deals with the features that go beyond the functionality of a build server. An example could

be a built-in Wiki or a ticketing system.

- Platforms: This category analyzes supported platforms on which the build server can be run. We also investigate whether different versions of the software are available and how they differ.

- License and Pricing: The last category of comparison deals with the license and the costs associated with the operation of the software.

Step 4: Development of a Reference Scenario After analyzing the build servers according to the categories mentioned above, we developed a reference scenario in cooperation with the project partner. The scenario was created based on the partner's previous considerations for modernizing the build process. The result is a sequence of tasks which have to be implemented by the system using specific technologies. Each system must be able to map the following process to meet the partner's requirements:

1. Reacting to a commit on a git server.
2. Building the project with Maven.
3. Testing the project with Maven.
4. Create a Docker image with the executable jar file.
5. Push the created Docker image to a Docker registry.
6. Initiate image deployment on a Kubernetes cluster.

Step 6 is optional, because if the image is in a Docker registry, the deployment can be initiated manually without much effort and the implementation of Continuous Deployment is not necessary. Within this work the above workflow was implemented with the examined build servers. A 'Hello World' Spring Boot 2.0 web application served as an example.

Step 5: Concluding Comparison of the Analyzed Subjects Finally, all project participants came together and presented the results to each other. A final comparison was made and a recommendation for the partner was worked out.

4 CI/CD Tool Evaluation: Travis CI

Travis CI is a cloud-based CI/CD platform that supports open source and private projects hosted at GitHub. The integration of GitHub and Travis CI is easily carried out over GitHub Apps. Travis CI can build and test code changes automatically and also provide feedback on whether the changes led to working software [7]. Note: Travis CI formerly offered two different platforms for open source (https://travis-ci.org/) and private projects (https://travis-ci.com/). Nowadays, Travis CI recommends using the latter platform that now merges both kinds of projects. Besides the hosted platform, there is also the on-premises Travis CI Enterprise platform [8].

4.1 Internals

When running a build, either triggered by a check-in to GitHub or manually over the web-based user interface, Travis CI clones the GitHub repository in question into a newly created virtual environment. A series of tasks is then executed to build and test the code. The following list explains some of the terms relevant to the build process in Travis CI according to [7]:

- A phase is one step in a series of sequential steps of a job. For each programming language Travis CI provides a set of default phases. Some of these phases are optional, depend on the success of another phase or on the success of the build as a whole. Two important phases of a job are install for installing required dependencies and script where specified scripts are run. Some other phases a job can contain are listed below:
  - before_script or after_script
  - after_success or after_failure
  - before_deploy or after_deploy
  - deploy
- A job is an automated process which clones the repository of concern into a virtual environment and carries out a series of phases. Each job runs in a fresh virtual machine or container and does not share storage with other jobs (unless an external mechanism is used).
- A build is a group of jobs. All jobs have to finish in order for a build to finish.
- A stage is a group of jobs which run in parallel. If multiple stages are defined, the jobs in each stage run in parallel, but one stage after another (sequentially). With stages it is possible to make jobs only run, if other jobs from all previous stages have completed successfully. A use case, that the Travis CI documentation states, is to "run unit tests, deploy to staging, run smoke tests and just then deploy to production" [7]. In addition, it is possible to define conditional builds/stages/jobs and perform a so-called matrix expansion to get matrices of jobs.

The build process is executed by the so-called Travis Worker. Internally, Travis CI has many components; the main components and their interactions are depicted in Figure 1, deduced from descriptions in Travis CI's GitHub repository. The build process works as follows:

- First, the (Travis CI) worker gets a bash script generated from the provided .travis.yml file,
- spins up a compute instance (e.g., a VM, container etc.),
- uploads the bash script to the compute instance and
- finally runs the script.

The build environment (either Linux, macOS or Windows) can be defined within the .travis.yml file. When the hosted version of Travis CI is used, the number of workers and the number of jobs they can execute is managed transparently by the platform. On-premises Travis CI Enterprise shall scale to the amount of workers as needed and defines the amount of jobs to be executed concurrently on a worker [9].

4.2 Pipelines

The aforementioned jobs are just Travis CI's denomination of pipelines [11]. The terms from the previous section are a selection of keywords used to outline jobs declaratively in a .travis.yml. The YAML file has to be placed in the root folder of a project for Travis CI to work. Besides describing the jobs it is also possible to specify everything around them, e.g. environment variables with env or services like Docker with services. As mentioned in section IV-A it is possible to group jobs into stages, e.g. stages that are named "test" or "deploy", and define phases for each

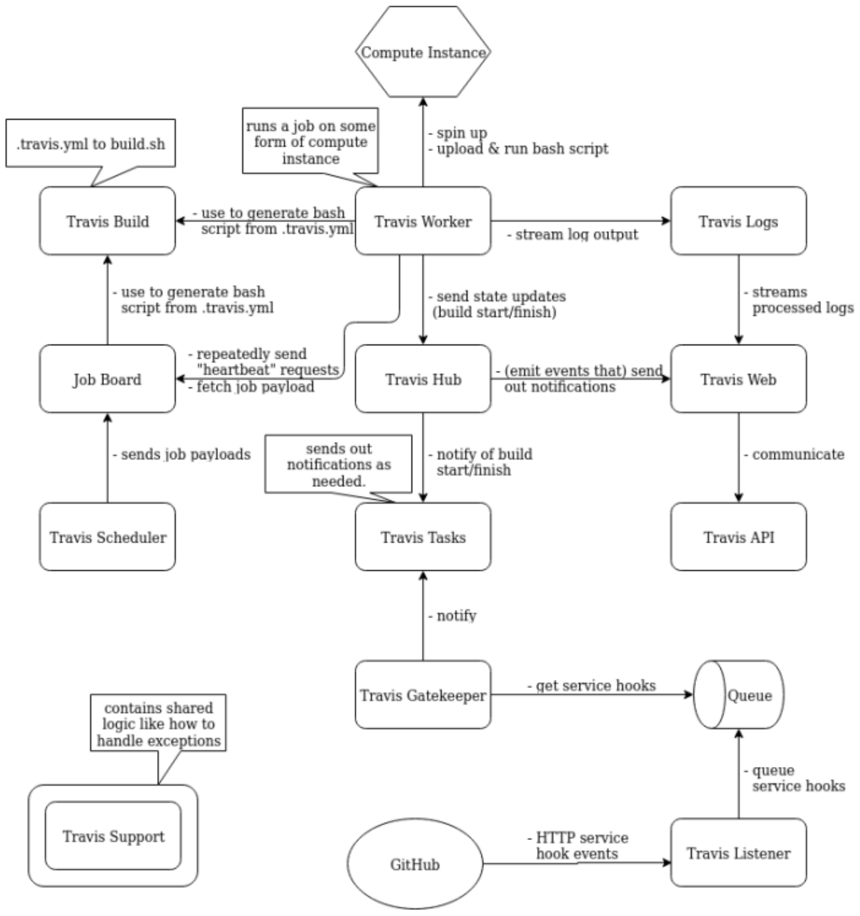

Compute Instance
- spin up
- upload & run bash script
runs a job on some form of compute instance
.travis.yml to build.sh
Travis Build
- use to generate bash script from .travis.yml
Travis Worker
- stream log output
Travis Logs
- use to generate bash script from .travis.yml
- send state updates (build start/finish)
- streams processed logs
- repeatedly send "heartbeat" requests
- fetch job payload
Job Board
Travis Hub
- (emit events that) send out notifications
Travis Web
- sends job payloads
sends out notifications as needed.
- notify of build start/finish
- communicate
Travis Scheduler
Travis Tasks
Travis API
- notify
Travis Gatekeeper
- get service hooks
Queue
contains shared logic like how to handle exceptions
- queue service hooks
Travis Support
GitHub
- HTTP service hook events
Travis Listener

Figure 1: The main Travis CI components deduced from [10].

job. A pipeline definition and the corresponding .travis.yml file vary in complexity as the requirements demand. Many examples for CI/CD pipelines can be found in Travis CI's user documentation or blog as well as in articles from other Travis CI users.

Next we use a .travis.yml file defining a simple pipeline for the reference scenario in II-D. The file specifies ...

- the language the project has and which version of the language the project should be build with.
- services to use: In this case Docker.
- environment variables. Credentials can be encrypted and added to the file.
- the content to cache for this job: In this case the local repository of Maven that contains all the project artifacts.
- commands to execute: mvn surefire:test -B.
- what to do after build success: Build a Docker image and push it to Docker Hub.
- where to deploy: Deploy to the Heroku platform.

The .travis.yml below shows how to cache content and how to push to an image store as pipeline relevant features.

```
language:java
jdk:openjdk8
services:
  - docker
env:
  global:
    - COMMIT=${TRAVIS_COMMIT::7}
    - secure : [ encrypted Docker user name ]
    - secure : [ encrypted Docker password ]
    - secure : [ encrypted Heroku API key ]
  cache :
    directories :
      - "$HOME/.m2/repository"
    script :
      - mvn surefire : test -B
    after success :
      - docker login -u $DOCKER_USER -p $DOCKER PASS
      - export TAG= ' if [ "$TRAVIS_BRANCH" == "master" ];
           then echo " latest " ;
           else echo $TRAVIS_BRANCH;
        fi '
      - export IMAGE NAME=example-repo/vis-travis-demo
      - docker build -t $IMAGE_NAME:$COMMIT .
      - docker tag $IMAGE_NAME:$COMMIT $IMAGE_NAME: $TAG
      - docker push $IMAGE_NAME
    deploy :
      provider : heroku
      apikey : "$HEROKU_API_KEY"
      app : exampleapp
```

In the above example is not shown, that it is possible to push to an artifact store. As with the image store (Docker Hub in the example), Travis CI does not provide an in-house solution, but allows to upload build artifacts to Amazon S3 at the end of a job [7].

4.3 Additional Features

Travis CI is a platform for CI(/CD) which itself does not offer many functionalities going beyond its original purpose. Nevertheless, it offers means to integrate services from other providers such as with the artifact or image stores like in the previous section. To notify someone of build results it is possible to send emails, but also send notifications to e.g. Slack channels or Campfire chat rooms or to webhooks [7]. With webhooks it is possible to use an app that allows Jira to get the build results [12]. Other integrations exist e.g. with SonarCloud to monitor the quality of source code, BrowserStack for interactive and automated testing or Atom Feeds to get updates on builds.

Furthermore, there are several tools to interact with Travis CI like command line tools or plugins for browsers. Many of these integrations can be defined in the .travis.yml file [7]. Features that would facilitate a DevOps workflow like a build-in wiki or ticketing system are not part of Travis CI, but since it is used with GitHub (Enterprise), these and other features might be found and utilized there.

4.4 Platforms

Travis CI is a ready-to-use Software as a Service (SaaS) product, but there is also Travis CI Enterprise, which can be operated on-premises. The underlying operating system of the hosted version is transparent to the users. On-premises the common setup for Travis CI Enterprise looks as follows: The Travis CI Enterprise platform and at least one Travis CI Enterprise worker run on physical machines or in virtual environments. Dedicated hosts or hypervisors (e.g. VMWare, OpenStack using KVM or EC2) should run at least Ubuntu 16.04; in the best case scenario Linux 3.16 is used and at least 16 GB of RAM and 8 CPUs are available [9].

4.5 License and Pricing

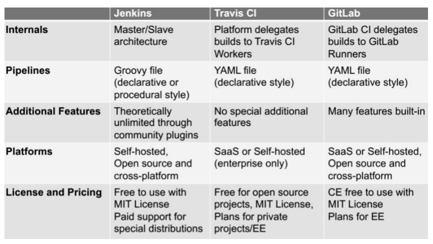
Travis CI's code is available at GitHub (open source) and usable under the MIT license [13]. The platform is free to use for open source projects, but for private projects only the first 100 builds are for free. It is possible to subscribe to different private project plans or to first start a free trial [8].

If certain customizations are wanted or Travis CI Enterprise is considered, the Travis CI team is to be contacted for consultancy and pricing [8].

5 Evaluation Summary

With this concluding article, all three selected build servers have been analyzed independently under the specified criteria from section 3. The present article in particular added the in depth analysis of Travis CI to our previous work [20,26].

To tie up the analysis and to compare the build servers, Figure 2 shows our summarized results. At first glance the internals of the build automations tools look differently, but they all use a similiar underlying master/worker architecture. While GitLab and Travis CI pipelines are defined declaratively using YAML files, Jenkins pipelines are defined declaratively or procedurally with Groovy via Jenkinsfiles. The procedural style gives Jenkins users more control over the internal flow of the pipeline, but requires more expertise.



	Jenkins	Travis CI	GitLab
Internals	Master/Slave architecture	Platform delegates builds to Travis CI Workers	GitLab CI delegates builds to GitLab Runners
Pipelines	Groovy file (declarative or procedural style)	YAML file (declarative style)	YAML file (declarative style)
Additional Features	Theoretically unlimited through community plugins	No special additional features	Many features built-in
Platforms	Self-hosted, Open source and cross-platform	SaaS or Self-hosted (enterprise only)	SaaS or Self-hosted, Open source and cross-platform
License and Pricing	Free to use with MIT License Paid support for special distributions	Free for open source projects, MIT License, Plans for private projects/EE	CE free to use with MIT License Plans for EE

Figure 2: Comparison of Build Automation Tools.

GitLab and Jenkins both support a full DevOps workflow. Jenkins through its plugin mechanism and GitLab through its many built-in tools. Travis CI, on the other hand, is a pure CI/CD tool, that doesn't offer many additional features. But it integrates smoothly with GitHub (Enterprise), which provides additional functionality. Travis CI can be compared to the GitLab CI component in GitLab.

There is no official SaaS solution for Jenkins. GitLab and Travis CI offer such a solution. Each examined tool can be run on premise. However, for Travis CI you have to pay for it. All tools offer versions that are available under the MIT license [6]. While Jenkins is completely open source and free to use, this is only partially true for GitLab and Travis CI.

6 Conclusion

Within the scope of this paper the market for build servers was examined. In cooperation with an industrial partner, three representatives were selected and a more in depth evaluation of them was undertaken. The detailed evalutaion of one of them and the overall brief evaluation results of all tools are presented in this article.

Comparison criteria were developed and grouped into categories. First the tools were examined independently. Then their similarities and differences were identified. To summarize, there are many tools that vary largely in their functionality. It is difficult to make a statement about which tool is the best, because different tools address different needs. However, beside the technical evaluation of the tools themselves, the comparison criteria and our general comparison approach might well be of value for others to undertake their own evaluation.

In [20] we presented the in depth evaluation of GitLab and in [26] we presented the in depth evaluation of Jenkins. Here we compared both to the Travis CI based solution in Section 5.

For our industry partner we recommend, that the functionality of the tools should be checked again against more detailed requirements. Jenkins as the currently used tool can meet all requirements known to us, but has to be equipped with a lot of plugins and configurations in order to do so. Beside GitLab and Jenkins, our partners also wanted to look in depth at the Travis CI build tool solution, which we presented here.

There is no clear winner in our comparison, since all tools have their pros and cons. At the moment we slightly favor GitLab however, because of it's DevOps workflow support, many useful features, open source availability, and it's big community.

References

[1] P. M. Duvall, S. Matyas, and A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk. Pearson Education, 2007.

[2] J. F. Smart, Jenkins: The Definitive Guide - Continuous Integration for the Masses. Sebastopol: O'Reilly Media, Inc., 2011.

[3] kohsuke. (2016) Use jenkins. [Online]. Available: https://wiki.jenkins.io/display/JENKINS/

[4] J. IO. Jenkins user documentation. [Online]. Available: https://jenkins.io/

[5] S. J. Bigelow. Jenkins-unterstuetzt-eine-umfassende-devopskultur. [Online]. Available: https://www.computerweekly.com/de/tipp/Jenkins-unterstuetzt-eine-umfassende-DevOps-Kultur

[6] Massachusetts Institute of Technology. The MIT License. [Online]. Available: https://github.com/jenkinsci/jenkins/blob/master/LICENSE.t

[7] Travis CI. (2019) Travis CI user documentation. [Online]. Available: https://docs.travis-ci.com/

[8] T. CI. (2019) Travis CI - test and deploy with confidence. [Online]. Available: https://travis-ci.com/plans

[9] Travis CI. (2019) Travis CI enterprise - Travis CI. [Online]. Available: https://docs.travis-ci.com/user/enterprise/

[10] T. CI. (2019) Free continuous integration platform for Github projects. [Online]. Available: https://github.com/travis-ci/travis-ci

[11] ——. (2019) The Travis CI blog: Setting up a CI/CD process on Github with Travis CI. [Online]. Available: https://blog.travis-ci.com/2019-05-30-setting-up-a-ci-cd-process-on-github

[12] Atlassian. (2019) Travis for Jira Jatlassian marketplace. [Online]. Available: https://marketplace.atlassian.com/apps/1220191/travis-for-jira

[13] T. CI. (2019) The Travis VI blog: Travis CI joins the Idera family. [Online]. Available: https://blog.travis-ci.com/2019-01-23-travis-ci-joins-idera-inc

[14] GitLab Inc. (2019) The first single application for the entire devops lifecycle. [Online]. Available: https://about.gitlab.com/

[15] ——. (2019) Gitlab CI/CD documentation. [Online]. Available: https://docs.gitlab.com/ee/ci/

[16] ——. (2019) Gitlab feature overview. [Online]. Available: https://about.gitlab.com/features/

[17] ——. (2019) Gitlab pricing. [Online]. Available: https://about.gitlab.com/pricing/

[18] C.v. Perbandt, M. Tyca, A. Koschel, I. Astrova, "Development Support for Intelligent Systems: Test, Evaluation, and Analysis of Microservices", in: SAI IntelliSys 2021. LNCS, vol 294. Springer, 2021.

[19] A. Koschel, A. Hausotter, R. Buchta, A. Grunewald, M. Lange, P. Niemann, "Towards a Microservice Reference Architecture for Insurance Companies", in IARIA SERVICE COMPUTATION 2021, The 13th Intl. Conf. on Advanced Service Computing, Online, Thinkmind, 2021.

[20] A. Grunewald, M. Lange, K.C. Tran, A. Koschel, and I. Astrova, "A Case for Modern Build Automation for Intelligent Systems", SAI Intelligent Systems (IntelliSys) 2022, vol. 1, Springer LNNS 542, 2022.

[21] M. Meyer, "Continuous Integration and Its Tools", in IEEE Software, vol. 31, no. 3, pp. 14-16, May-June 2014.

[22] X. Jin and F. Servant, "What Helped, and what did not? An Evaluation of the Strategies to Improve Continuous Integration", 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021, pp. 213-225.

[23] M. Shahin, M. Ali Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices", in IEEE Access, vol. 5, pp. 3909-3943, 2017.

[24] C. Singh, N. S. Gaba, M. Kaur and B. Kaur, "Comparison of Different CI/CD Tools Integrated with Cloud Platform", 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2019, pp. 7-12.

[25] P. Rai, Madhurima, S. Dhir, Madhulika and A. Garg, "A prologue of JENKINS with comparative scrutiny of various software integration tools", 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), 2015, pp. 201-205.

[26] M. Lange, K.C. Tran, A. Grunewald, A. Koschel, A. Pakosch, and I. Astrova, "Modern Build Automation for an Insurance Company Tool Selection", in: Proc. CENTERIS 2022, Elsevier Procedia CS, 2022.