Adopting CI/CD Technology for Internet and Mobile Banking Services Integrations in case of Bank of Abyssinia

Fisseha Jihad
HiLCoE Graduate Program, Ethiopia
fissehajihad@gmail.com

Abstract

Since the early 2000s Continuous Integration and Continuous Delivery (CI/CD) have become a part of the software development practices. However, this method is not well known and it is not being widely practiced in Ethiopia, let alone in Bank of Abyssinia. This is because the benefits are not well understood. This paper attempts to show the benefits of incorporating CI/CD in the development practices. The paper presents the comparative analysis made against four on-premise CI/CD tools that are being used worldwide. These are Jenkins, GoCD, Bamboo and TeamCity. And finally, the paper recommends a CI/CD tool out of the four that can be used in a development environment with a team such as the one at Bank of Abyssinia.

Keywords: continuous, integration, delivery, method, tools

1. Introduction

Bank of Abyssinia (BOA) is one of the preceding banks in Ethiopia. The Bank opened and started giving service to its customers in 1996. The bank currently has more than 10,000 employees. BOA used to be considered as one of the very late joiners in the technology sectors compared with the banks that had started along with it. However, in the past few years, it has emerged as one of the leading banks by introducing multiple pioneer technological products for its customers. BOA was the first bank to introduce Virtual Banking using ITMs, the first bank to acquire a payment gateway platform and the first bank to complete integrations telebirr. Currently BOA has finalized implementation of the first Digital banking platform in Ethiopia by acquiring a product called Temenos Infinity. BOA has more than 700 branches all over the country. And to increase its accessibility, BOA has deployed more than 1000 ATM machines and more than 2000 Point of Sale machines.

In Bank of Abyssinia a service integration with the Internet and Mobile banking solution is done very frequently. These integrations are mainly features that will be visible for customers. An Integration with DSTV, so that customers can pay their Bills on time or with EthSwitch, so that customers can send money to other member banks within the country instantly, or different kinds of billers, or merchants. But there are also constant bug fixes and enhancements that are done on a weekly basis. As the number of customers increase and the product is used frequently by a large sum of users, such developments of new features, enhancements and bug fixes have

also increased at a very rapid rate. And due to this, unlike before, more than one developer will be working on the code at the same time.

While multiple individuals are working on the code at a time, the current practice is teams take the latest code into their work station and they make all the necessary changes, such as add a new feature or fix a bug or make an enhancement without affecting any of the working functionalities that were previously done. Once the team working on the changes have finished applying their updates, the product will be built and the generated artifact is moved to the test servers for deployment. The team will then carry out the necessary tests on the new environment. If the expected result is achieved, the development team informs the business team about the new changes, so that they can perform the tests and approve the changes to be moved to the production server. After the business team approves the changes then the team makes the necessary changes to the package and will deliver the artifact to the infrastructure team for production deployment. However, when simultaneous works are done parallelly with other teams, the process will be quite different. Each team will proceed with their own updates to the code and test on their own environment, then they will prepare a separate build and test it on a different environment. All changes done by the team will be tested by the business teams separately. The business teams that will be testing the separated updates might be different as well. After the business team has finalized tests and the expected results are achieved, the process will be different, this is because all changes have to be compiled together. This is where the difficulty arises. One team must incorporate the changes done by the other team or teams and merge the changes together manually. This is a process of manually copying pieces of code from the other team or teams to their own. While doing this, usually the changes are affected and all teams must be involved, so that the issues on the features they have done will be troubleshooted and fixed. After the merge is completed, just like a new feature the steps will be repeated, internal tests by developers, move to the test environment and then let the business team carry out their tests and approve. Only then the changes can be delivered to the infrastructure team for production deployment.

This process is tedious and time taking. This way the team as a whole cannot be effective. In order to avoid the manual labor where teams merge their codes manually, the primary step will be to use some kind of git repository. By utilizing a git repository, the teams can work on their own branch and merge their changes all together. However, this will not solve the issues we face while merging the codes. Which is a merging conflict. If concurrent changes refer to the same file, we say that these changes are conflicting. As [1] states that, conflicts on a file can be automatically resolved or they need user intervention for their resolution. So, the best solution that is currently being practiced by developers all over the world is the continuous integration and continuous delivery. The idea is instead of compiling or merging the changes done by the teams after the delivery of a new feature, to merge the code very frequently and perhaps on a daily basis or even multiple times a day. The merging also will be monitored by a

CI/CD tool which senses a new merge and builds the package automatically or manually depending on the configuration to see if the code is okay. Also, the CI/CD tool will perform a series of tests that are preconfigured by the developers to make sure the code base is intact and functional.

By incorporating this process in the development practice, the teams can focus on their own features only and they do not have to worry much about manually integrating the changes made by other teams. This alone will remove a huge burden for the development team. The business team does not have to test the features separately and multiple times, because the delivery will be together. Multiple test environments do not have to be ready and configured in order to deploy multiple updates that are done at the same time. This way the overall process will be very efficient and effective. Hence, the objective of this paper is to showcase the list of available CI/CD tools and their unique features in order to help the BOA development teams in selecting a tool that will best fit their needs.

2. Related Work

According to [2] a Software Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation. [2] continues to explain that a number of software development life cycle (SDLC) models have been created: waterfall, spiral, V-Model, rapid prototyping, incremental, and synchronize and stabilize. As stated in [2] the oldest of these, and the best known, is the waterfall model: a sequence of stages in which the output of each stage becomes the input for the next.

However, it is nearly impossible to follow the waterfall methodology while working on a product where new features are required constantly and they have to be released frequently. Hence, for projects with such nature it will be highly advisable to follow the agile approach, and sometimes it is the only way. This is mainly because new features are required frequently, but also, the requirements are usually dynamic. A new proclamation will be declared by the National bank of Ethiopia to enforce some kind of regulation, and the changes might need a new development, or the management of the bank decides that the transfer to other banks should be temporarily disabled using the digital channels. These reasons will force the development team to be very agile and practice the methodology in their product delivery process.

According to [2] agile software development is a group of software development methodologies based on iterative and incremental development where requirements and solutions evolve through collaboration between self-organizing, cross functional teams. As [2] continues agile methodology has an adaptive team which is able to respond to the changing requirements. According to [4] the Agile software development process focuses mainly on fast delivery, and CI helps Agile in achieving that speed.

Continuous Integration (CI) is a software development practice, where developers frequently integrate their work with the project's Integration branch and create a build. It is

necessary to bring out issues encountered during the integration as early as possible. According to [3] CI/CD pipelines are the perfect complement to any team using agile methodologies, in ideal situations development teams are building features that are committed to source control often. As [4] states it helps because a build failure can occur due to either an improper code or a human error while doing a build (assuming that the tasks are done manually).

Continuous Delivery (CD) is the process of moving the code base into the production environment. Continuous delivery, according to [6], is the ability to get changes of all types – including new features, configuration changes, bug fixes, and experiments – into production, or into the hands of users, safely and quickly, in a sustainable way. Traditionally, what has become a trend is that, whenever a developer or a team of developers have finished their features, they will compile their works together, run a series of tests manually, move the changes to another environment where additional tests can be done by a testing team, and if all things go right, then move it to production. [5] notes that with Continuous Delivery, the artifacts that are produced by your CI server are deployed to the production server with a single button click. This can be achieved by automating the steps that are in between the development and the production deployment.

According to [4] a CI tool is at the center of the CI system, connected to the Version Control System, build tools, Binary Repository Manager tool, testing and production environments, quality analysis tool, test automation tool, and so on.

There are many advantages to incorporating the process of CI/CD in the software development practice.

CI/CD Helps to automate testing phases: The tool can automate tests; this is done by predefining the series of tests that should be performed before a product is said to be working properly. CI/CD tool, can compile and build a source code automatically when a commit to the code happens. But testing is a manual, time-consuming, and repetitive task, automating the testing process can significantly increase the speed of software delivery. However, automating the testing process is a bit more difficult than automating the build, release, and deployment processes. According to [4] it usually takes a lot of effort to automate nearly all the test cases used in a project. It is an activity that matures over time.

CI/CD Helps in working on small batches and avoid big bang: By committing or submitting the codes to the git continuously, we can avoid colossal setbacks. This is because the more frequently we submit, the less code base we change. And since the tests are done automatically, the CI/CD tool will inform us if there's any issue or incompatibility has occurred with the existing code and anything has been broken.

CI/CD Helps reduce errors and bugs in production environment: By committing the codes more frequently and carrying out automated tests more rapidly the team can catch error at a very early age and giving a fix before it affects any other functionality, this will help increase

the quality of the code, and will greatly reduce any errors or issues that will happen in the production environment, because we have a relatively more robust system.

CI/CD Makes easy the process of recovery from failure: Each commit is registered on the git repository, but without the CI/CD tool we can't be certain which code was working properly or not. With the help of the tool, we can easily rollback to the last working build instantly and easily.

CI/CD Increases the development productivity: development teams can focus and work on their own segment, and the time they spend solving merging conflicts will be much less.

CI/CD Creates Teams efficiency: not only the developers but the entire team involved in the product, from requirement to the release will have a smoother process.

CI/CD Accelerates speed to market: the time to release a product to the market will be much faster, since many time taking tests are automated, code delivery to environments can be automated, and integration process is quickened. Though it's not recommended, the production deployment can also be automated, but it is best to do that manually, because there might be a test that can't be automated and must be done manually.

Jenkins as stated in [6] is an open-source automation server written in Java, with very active community-based support and a huge number of plugins, it is one of the most popular tools for implementing continuous integration and continuous delivery processes.

A Jenkins Pipeline is a workflow with a group of events or jobs that are chained and integrated with each other in sequence. A continuous delivery (CD) pipeline as noted in [10] is an automated expression of your process for getting software from version control right through to your users and customers. [10] also notes that a Pipeline is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

Bamboo as stated in [9] is a continuous integration (CI) server that can be used to automate the release management for a software application, creating a continuous delivery pipeline. The product is supplied by Atlassian, which provides tools such as Jira, Trello, Confluence, Bitbucket and others. These products are used by many project managers and their respective teams due to their easy to use and rich interfaces. As a result, it can also be integrated with these products.

GoCD is one of the recently emerged competitors in software automation, and claims to have been designed for complex workflows that can be defined as code. GoCD integrates with many popular external tools and services via its extensible plugin architecture. [12] argues that GoCD helps to troubleshoot a broken pipeline by tracking every change from commit to deploy in real time. GoCD was originally released by ThoughtWorks in 2014, and it is based on the CruiseControl automation tool.

Volodymyr Melymuka [7] described TeamCity as a very light instrument, easy to install, integrate, and maintain. According to [7] TeamCity is a tool which helps to ensure that a software project not only compiles properly but can be assembled and (ideally) allowed to be delivered to operational destination production servers merely by glancing at the TeamCity welcome page. As claimed by [7] for distributed teams, it could give a priceless experience of having a reliable codebase free from some forgotten to be committed source files and resources.

For comparing these technologies, it is mandatory to select the parameters first. [13] has mentioned 7 criteria that are crucial while selecting a software. Functionality & Ease of Use, Vendor Viability, Technology, Cost, Support and Training, Industry Expertise and Implementation. Since we are currently taking an off the shelf product without any customization, we can ignore the vendor related criteria and focus on the others.

3. Methodology

I have selected four CI/CD technologies to be compared in this paper. These technologies are selected out of many because they can be configured and used on-premise. In a bank, usually the connectivity in the work stations with access to the banks servers are kept out of Internet access or with a very limited access to the Internet. This is due to security concerns and having an on-premise solution for such tasks is advantageous. After selecting the four CI/CD technologies five parameters were used to compare each of them, these are: Learning Curve, Initial Setup, Supported Plugins, Resource & Support Community and at last License & Cost. These parameters were measured by referring to the literature found, the contents found on the technology providers web pages and finally by experimenting on the technologies on my local machine. Some parameters are assigned ordinal variables. Learning curve, with rating variables, easy, medium and difficult based on the amount of time it will take the BOA team to adopt the technology and the complexity of the system. Supported Plugins, with rating variables, low and high based on the available numbers mentioned or indicated. Resource & Support Community, with rating variables low and high based on the literature found. All parameters are combined together and presented in a tabular form.

For selecting the final technology out of the four, License & Cost has been given the most priority and only the most rated technologies for this criterion are selected to be recommended. The final CI/CD technology selected shall make the development team at BOA more effective.

4. Comparative Analysis

4.1. Learning Curve

While adopting any new technology, we need to consider the learning curve of the product we are going to adopt. Currently many of the developers at Bank of Abyssinia do not yet understand the necessity of a CI/CD tool. Even if they are aware or became aware of the advantages it can give them, it means nothing if the product is complex to understand and that it requires long hours of training and dedication to make it useful for the team. The team will

definitely be very hesitant to adopt such technology and it is more likely that the product itself will be inefficiently used. Hence, we have to consider this before selecting any product and introducing it to the team.

All products selected here in this paper have more or less similar Interfaces and navigation and they appear to be somewhat easy to use from a glance. But for someone who is not familiar with the concept of CI/CD, it is very tricky to understand what the front-page supplies.

In Jenkins creating a series of tasks to make a pipeline can be easy if one is using a Freestyle project. And a freestyle job will be used to execute one specific task. However, in the Post-build actions. we can attach or link another Project to an existing one to address the subsequent actions. This could be a Unit test then an Integration Test, then a User Acceptance test and so on. However, this has its own downside, this is because of the UI. The UI based configuration in Jenkins is very limited, it presents predefined fields to be used based on the specific plugin selected. So to address this limitation, Jenkins uses Pipeline types of projects which is a scripting based project. Using Pipeline Jenkins introduced a much more flexible process. In order to use Pipeline, we need to learn the scripting language used. Pipelines are built with simple text scripts that use a Pipeline DSL (domain-specific language) based on the Groovy programming language. Groovy [14] is an object oriented and Java syntax compatible programming language built for the Java platform. This will be an easy process for the BOA team to adopt it, since the team is capable of writing java codes.

In bamboo, the process is more or less similar with that of Jenkins, however some of the terminologies are different. Here, we do not create a pipeline, rather we create a project. A project contains a plan and a plan contains jobs. Configuration of basic projects and running different builds are very easy. For a more complex configuration, Bamboo provides the scripting method called Bamboo specs. The specs can be written using Java or YAML. For a naive user the YAML seems to be very easy to understand and more preferable, but this too means the team needs to understand such scripting languages to make the best out of the product. However, since the team is capable of understanding and writing java code, it will be easy to adopt it.

GoCD gives a setup that is very similar to Jenkins. It uses a pipeline, while creating a new pipeline, the steps are very simple to follow. The setup is segregated by parts. Each part briefly explains what is expected. First a repository, then a pipeline name, a stage name and finally jobs & tasks. It presents an easy process for anyone who is well aware of what they are going to do, regardless of their experience on the GoCD platform. GoCD also provides the option Pipeline as a code, in which a more experienced user can utilize it for a rapid and complex setup. GoCD allows the configuration of Pipeline as a code using a declarative language rather than an imperative language, and for this it utilizes JSON, YAML and XML formats. The configuration could be part of the repository or it could also be separate from the code

repository. For the BOA team they work with both JSON and XML, however the tags used need to be understood, and it might take them time to get familiar with those. Hence the effort will not be as easy as writing java code.

The way TeamCity handles build configuration relates with that of Bamboo. We initially set up a project and the project configuration contains VCS (version control system) steps and a build step. When the code is fetched from the git repository, it automatically detects the possible build that can be configured. For Example: if the repository contains a java maven project, it senses the project type and provides build steps for the maven test. For a non-UI configuration, TeamCity provides configuration as code, but for this they use a full featured programming language called Kotlin. Even if using this full-fledged programming language makes it possible to create a configuration with complex requirements, it also adds an additional headache for anyone who will be administering the CI/CD tool. And for the team at BOA this is a new language that has not been used before, as a result it will be time taking to learn this new language.

4.2. Initial Setup

All tools presented here have a relatively very easy setup process with the exception to Bamboo. Jenkins installation is easier and it does not require any additional database for installation. This can also be a downside depending on the requirement we have. While setting up bamboo, the initial license key has to be requested from the Atlassian account, without the license no one can proceed and use the tool even for the free version available only for 30 days. Bamboo, GoCD and TeamCity have provided the options to install the tool using internal database H2 for GoCD and Bamboo, HSQLDB for TeamCity or using external DBs such as MySQL and PostgreSQL. In addition to these, bamboo supports MSSQL and Oracle as well. All tools support major operating systems, Multiple Windows and Linux Distributions and MacOS. On top of these, they all support docker setup as well. For initial installation all tools require Java 1.8 and above. The hardware specs differ for each tool. GoCD, with the least recommended requirement, suggests 2GB for server and 256MB for agent RAM, 2 cores(2GHz) for server and 1 core(2GHz) for agent CPU and 1GB minimum free space. Bamboo recommends (for individual and small teams) 4GB RAM and 20GB space should be allocated, though the installation size is 144MB. Jenkins recommendation is very generic with 4GB of RAM and 50GB of Storage. TeamCity is the most resource intensive compared to the listed tools, the size of the installer itself is more than 1.7GB. They note that for a minimal setup, 8GB Ram with Intel 3.2 GHz dual-core CPU and 1GB Network adapter can provide acceptable performance. All tools are web-based applications; hence they require a web browser to run the interfaces.

4.3. Supported Plugins

Jenkins, according to their official website, have more than 1800 plugins that are contributed by the open-source support community. The others even if they do not advertise their number

of plugins, compared to Jenkins they have a very lesser count. However, this doesn't mean we have a problem with building, testing and deploying a project. Unlike Jenkins, which requires a plugin for each task to be done, the others come with built-in plugins which allow the tools to do certain tasks. TeamCity has built-in capability for triggering builds automatically, or using specific versions of maven and gradle (as long as the gradle and maven versions are not newer than TeamCity release date). Bamboo has built-in plugins for all Atlassian project integrations, such as Jira, bitbucket and confluence. GoCD comes with bundled plugins for LDAP, YAML, JSON and password file authentication.

4.4. Resources and Support Community

As one of the oldest and most popular products, Jenkins has a lot of resources online. The website itself provides a well-documented guide for a beginner and there can be found numerous resources outside of the official website. According to [6] Jenkins is highly plugin-oriented, it has a great community that constantly extends it with new features. GoCD has sufficient documentation for a beginner, but [8] states that the quality of the official documentation was found to be uneven and it was sometimes lacking or confusing, which can be a hindrance. GoCD and TeamCity have an active forum where a participant can raise any question and receive any support directly. Unlike Jenkins, the resources found outside the forum and official website are somehow limited. Considering Bamboo and TeamCity, their documentation is very sufficient for anyone who is using them for their common features. But they are enterprise solutions and they provide SLA with the paid subscription

4.5. License & Cost

While Jenkins and GoCD are open-source tools they do not come with any license cost. The only cost they will incur is the resources to be used. However, TeamCity and Bamboo have license options. TeamCity for minimal operation the license is free, but it's limited to 100 build configurations, 3 agents and support via forum and issue tracker. But for dedicated technical support, unlimited build configuration and 3 agents there is a $1,999 payment for the first year and $999 for the second and onwards. The price increases as the number of agents increase. An agent according to [11] listens for the commands from the TeamCity server, it can be configured separately from the server. The main advantage of the separation is to set it up in different operating systems, so that we can test our codes using different system setups. But for the case of Bank of Abyssinia, 3 agents can be considered very sufficient. Bamboo provides the application free of charge for the first 30 days, which is a trial period, after that the license will expire unless it is purchased. The product is available for $1,200 with 1 agent, $3,200 with 5 agents. The 5 agents are more than enough for our use cases.

Finally, all parameters are combined together in Table 1 below.

Table 1: All Comparison parameters combined together

	Jenkins	**Bamboo**	**GoCD**	**TeamCity**
Learning Curve	Easy	Easy	Medium	Difficult
Initial Setup	RAM Size: 2GB for Server and 256MB for agent CPU: 2 cores (2GHz) for server and 1 core (2GHz) for agent Storage: minimum 1GB space	RAM: 4GB Storage: 20 GB	RAM: 4GB Storage: 50 GB	RAM: 8GB CPU: 3.2 GHz dual core Network Adapter: 1GB Installer: 1.72 GB
Supported Plugins	High	Low	Low	Low
Resource and Support Community	High	Low	Low	Low
License & Cost	Free	$1,200 with 1 agent $3,200 with 5 agents	Free	$1,999 - year 1 and $999 for subsequent years

4.6. Recommendation

The practice followed by the development team of Bank of Abyssinia working on Mobile and Internet banking products is currently outdated. The team is responsible for major product releases that directly affect the customer. This practice needs to be updated soon. The initial and major step to take is to have a centralized git repository. There are numerous git repository options, some provide an online service such as github, gitlab and many others and some could be configured on-premise. Once this is accomplished the subsequent step will be incorporating a CI/CD method to the process of the development. Selecting a tool that can facilitate this can be a difficult process, especially in an environment where the necessity of the method is not yet well understood. In order to select the best tool for the team and the bank, four well known CI/CD tools are selected. I have compared the tools based on their Learning curve, initial setup, supported plugins, resource & support community and finally their license & cost. As shown in Table 1.0, Jenkins has an easier learning curve, is relatively less resource intensive, has a high number of plugins and high resources available with a support community and most importantly it is free. Hence, Jenkins is the obvious choice to be selected based on the parameters defined.

5. Conclusion

CI/CD has been part of the software development life cycle since the early 2000s. The practice introduced many advantages for teams consisting of multiple individuals and with

frequent updates and maintenance are involved. The tools presented in this paper will help Bank of Abyssinia's development team to utilize their time efficiently, have a cleaner and maintainable Internet and Mobile Banking product and assist them greatly in the collaborative work they do within the team from build to test until the product is deployed and delivered to the customers. My final recommended tool is Jenkins, which is a free and open CI/CD technology. While utilizing a completely open-source product some unresolvable issues might occur, the support from the community might not be sufficient. During such times an enterprise solution comes in handy because they come with an SLA. But before going forward with such solutions, it is ideal to use free tools to get familiar with the process of CI/CD. The performance of the tools has not been measured and brought for analysis. This is because of the lack of the proper lab resources to conduct such tests. In the future, it will be ideal to incorporate such parameters in order to give a more reliable recommendation.

References

[1] Hoai Le Nguyen, Claudia-Lavinia Ignat, "An Analysis of Merge Conflicts and Resolutions in Git-based Open-Source Projects", retrieved from https://hal.archives-ouvertes.fr/hal-01917249/document, Last accessed on Nov 18, 2022Joni Virtanen "Comparing Different CI/CD", 2021

[2] S.Balaji and Dr.M.Sundararajan Murugaiyan, "Wateerfallvs V-Model Vs Agile: A Comparative Study On SDLC", June 2012

[3] Brandon Atkinson and Dallas Edwards, Generic Pipelines Using Docker: The DevOps Guide to Building Reusable, Platform Agnostic CI/CD Frameworks, Apress, 2018

[4] Nikhil Pathania, Learning Continuous Integration with Jenkins: A beginner's guide to implementing Continuous Integration and continuous Delivery Using Jenkins 2, Second Edition, Packt, 2017

[5] Sander Rossel, Continuous Integration, Delivery, and Deployment: Reliable and faster software releases with automating builds, tests, and deployment, Packt, 2017

[6] Rafał Leszko, Continuous Delivery with Docker and Jenkins: Create secure applications by building complete CI/CD pipeline, Third Edition, Packt, 2022

[7] Volodymyr Melymuka, TeamCity 7 Continuous Integration Essentials: A step-by-step introductory tutorial and intelligible practical guide to successfully applying Continuous Integration via TeamCity, Packt, 2012

[8] Johannes Heikkinen, "Investigating the suitability of GoCD for software automation use cases", Master's Thesis, Master of Engineering, Information Technology, Metropolia University of Applied Sciences, 2021

[9] Atlassian, "Understanding the Bamboo CI Server", retrieved from https://confluence.atlassian.com/bamboo/understanding-the-bamboo-ci-server-289277285.html, Last accessed Nov 18, 2022

[10] Jenkins, "Pipeline", retrieved from https://www.jenkins.io/doc/book/pipeline, Last Accessed Nov 7, 2022

[11] Jetbrains, "Build Agent", retrieved from https://www.jetbrains.com/help/teamcity/build-agent.html, Last accessed Nov 18, 2022

[12] GoCD, "Free & Open Source CI/CD Server", retrieved from https://www.gocd.org/, Last accessed Nov 19, 2022

[13] Software Alliances, “Software Selection Process and Criteria”, retrieved from https://blog.software-alliances.com/software-selection-process-and-criteria, Last accessed Jan 17, 2023
[14] Guru99, “Groovy Script Tutorial for Beginners”, https://www.guru99.com/groovy-tutorial.html, Last accessed Jan 17, 2023