

Extended Model of Code Orchestration and Deployment Platform

Todor Ivanov, Nikola Valchanov

Plovdiv University, Faculty of mathematics & Informatics, 236, Bulgaria Blvd., Plovdiv, Bulgaria

Abstract – This paper is focused on the process of continuous integration and respective orchestration tooling. It provides a summary on existing tooling with feature analysis and applications. The research explores techniques, processes, and solutions for code orchestration. It includes a comparison of the modern platforms and discusses the topic of extendibility of such products by presenting an architectural model that supports the integration of general-purpose extensions.

Keywords – continuous integration, continuous delivery, notifier, pipeline, infrastructure.

1. Introduction

Nowadays, information technologies are growing and developing rapidly. Almost every existing problem can be either solved or significantly simplified by software applications. In order for those products to be accessible, they need to be deployed either publicly or within a corporate infrastructure. In most cases that is managed by dedicated teams that are costly both in terms of invested time and resources. The process of deployment itself is very complicated and strictly specific for each product. It is often custom-tailored and constantly accommodated to the needs of the product as it evolves.

DOI: 10.18421/TEM112-45

<https://doi.org/10.18421/TEM112-45>

Corresponding author: Todor Ivanov,
Todor Ivanov, Plovdiv University, Faculty of mathematics & Informatics, 236, Bulgaria Blvd., Plovdiv, Bulgaria.


Email: ivanov.todor1@gmail.com

Received: 15 April 2022.

Revised: 16 May 2022.

Accepted: 21 May 2022.

Published: 27 May 2022.

 © 2022 Todor Ivanov & Nikola Valchanov; published by UIKTEN. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDeriv 4.0 License.

The article is published with Open Access at <https://www.temjournal.com/>

Until a few years ago it was common for the deployment process to be a manual effort. The process involved both developers and business analysts to follow a checklist of actions in order to deploy and ensure the quality of the deployment. Those operations included preparing the source code, building the product and making it available. In addition to that, every developer was responsible for his own local development environment which sometimes is a pretty difficult task and requires a lot of resources. Even though this has to be done only once, there are still a lot of repetitive operations that have to be executed since the product never really stops growing and developing.

Because of that, companies started to search for an automated solution. Its main goal of this automation was to speed up and ease the development process by automatically performing all of the repetitive steps that otherwise have to be executed manually [5]. Nowadays this automation process is common practice and is called continuous integration (CI) [1], [2], [4]. It does not only solve the mentioned problem, but it also gives additional opportunities for improvement of the quality and flexibility of the product.

2. Analysis of Existing Solutions for Code Orchestration

There are a few existing solutions that provide us with some tools that may help in solving the problem. Below are three of the most widely used code orchestration tools in modern days.

The first solution that we will consider is Jenkins [6], [7]. It is an open-source automation tool written in Java and built for Continuous Integration purposes [18]. Typically, it is being run as a standalone application in its own process with the built-in Java Servlet called Jetty. However, if the user wants to, Jenkins can be run as a servlet on Apache Tomcat or other Java Servlet containers. This tool is often used to automate the development workflow by executing tasks such as building projects, running tests, doing code analysis, and running deployments.

Fortunately, Jenkins is distributed as a WAR archive, installer package, Homebrew package,

Docker image and as a source code and that allows the users to run it on almost any platform and OS. It produces a web user interface and accepts calls to its REST API. When one runs Jenkins for the first time, it creates an administrative user with a long random password, which we can paste into its initial webpage to unlock the installation. Once installed, one can either choose to get plugins or continue with the native product.

In order to achieve the process of Continuous Integration (CI) in Jenkins, we need to create pipelines [6]. The pipelines are a set of commands that are executed by any order, and they automate certain tasks. These pipelines are configurable files that are handled by the Jenkins Server. It depends on the project architecture, but sometimes one might need several different environments to test his builds. This cannot be done by a single Jenkins server. Also, if larger and heavier projects get built on a regular basis then a single Jenkins server cannot simply handle the entire load. In that case, one will need to take advantage of the Jenkins distributed architecture.

In the distributed architecture [12], [17], Jenkins uses Master-Slave instances that communicate through TCP/IP to manage the build processes. The master instance is the Main Jenkins server, and its job is to handle the main tasks like:

- Scheduling build jobs;
- Monitoring slave instances;
- Dispatching build tasks to the slave instances for actual execution;
- Recording and presenting build results;
- Sometimes, if needed, executing build jobs directly.

That being said, the slave instances are responsible for just doing the job. They are Java executables that run on a remote machine and listen for master commands through the communication channel. Jenkins Slaves usually:

- can run on different operating systems and machines;
- Listen for requests from the master instance;
- Execute build jobs that are dispatched by the master instance.

In addition, a configuration can be done in which we can specify a slave instance to always run tasks for a certain project.

In order to execute any build or testing job, Jenkins takes advantage of Webhooks [8]. The Jenkins server checks the code repository at certain intervals and if any changes are made to the source code, it triggers a job. This job is executed by a predefined pipeline and that allows us to know which change is getting tested, which change is sitting in the queue or when

the build is broken. In the build pipeline the build as a whole is broken down into sections, such as the unit tests, acceptance tests, packaging, reporting and deployment phases. The pipeline phases can be executed in series or in parallel, and if one phase is successful, it automatically moves on to the next phase (hence the relevance of the name “pipeline”).

While one uses the web UI to create pipelines [15], the current best practice is to create a Jenkinsfile and place it in the repository. These files can be declarative or scripted. The simpler of the two, the declarative one, uses Groovy-like syntax. It consists of a “pipeline” block, which in turn, has nested “stage” blocks that describe executable steps.

When it comes to pricing, Jenkins is free. Comparing it to TeamCity or Travis [9], [10], which requires us to pay a subscription after the free trial; here we will only need to pay for the machine that one is going to use to host it. That, and the fact that it is basically community driven and has thousands of free plugins, makes it a great choice as a continuous integration tool.

TeamCity [13], [14] is considered the best alternative to Jenkins. It is secure out of the box and offers extremely stable plugins. It also got handy integrations with xUnit and other code coverage tools. Like Jenkins, this tool is often used for Java and .NET projects.

Installing and configuring TeamCity is easy as it only involves downloading the appropriate TeamCity Server installer package and executing the installation instructions. It provides a few pre-built options that can run on different environments like Windows, Linux, macOS and via Docker. All of them come with a bundled Tomcat in some way so the user does not need to install it manually. Unfortunately, TeamCity is only available on premise. If one would like to host it himself, he will have to set up a proxy server like Nginx.

By installing TeamCity Server, we get a web application that is responsible for the core functionality of the product. It gives a rich user interface that helps distribute the jobs (builds) to TeamCity agents and aggregate their results. Using the UI, the user can configure, setup and manage the entire build and all of its steps [11]. This makes it easy to use as it does not require any additional experience or domain-specific knowledge.

To run any jobs in TeamCity, one needs agents. These agents can be different depending on the job that they will execute. A fresh TeamCity server has one registered build agent by default that runs on the same machine. Additionally, TeamCity will add a version control system (VCS) trigger automatically when creating a project or building a configuration from a repository URL. If such change occurs, a build is being queued.

Whenever a build is placed in the queue, the first free agent takes it up and tries to execute it. Since the build itself is separated into steps, each step is executed consecutively after the previous one finishes, and the result can be either a simple status or an artifact. Usually, artifacts are published to the TeamCity Server and can be seen or downloaded. The main purpose of these artifacts is to allow the builds to be connected into a build chain called pipelines. They are often designed to compile, test, and deploy a certain project, but one can create them for any other goal. Furthermore, if the desired building or testing framework is not supported, TeamCity provides an option for the end user to create custom scripts that can extend the capabilities of the agents. These scripts are implemented with service messages, which are specially constructed pieces of text that pass commands or information about the build from the script to the TeamCity server. However, in order for the custom scripts to be executed by the TeamCity agent, they need to be explicitly written to the standard output stream of the process.

Unfortunately, TeamCity does not provide its full capabilities for free. Taking into consideration that it is only available on premise, it comes with a standard payment subscription package. We can use the free version of the tool with some limited functionalities, which in my opinion is a good option when you try to explore the possibilities of the tool or you can purchase an additional license for the full set of the instruments.

TravisCI [16] is another widespread alternative for continuous integration. It allows developers to quickly and easily build, test, and deploy code. The tool is developed mainly in Ruby and currently it is maintained by the Travis CI Community.

The good thing about Travis is that we don't really need to install it. It is available on premise as a SaaS and the only thing a user needs in order to use it, is an account. Travis is usually integrated with projects that are hosted in Version Control Systems like Github, Bitbucket and Gitlab. It supports the most popular programming languages and can easily run builds and tests on your code.

In order to use TravisCI in our product, all we need to do is to provide a configuration file. This file has to be located in the repository that will take advantage of the CI and has to follow the YAML syntax. The user will need to approve TravisCI's contribution request to the repository and after that the functionalities will be available for every open merge request.

TravisCI differs from Jenkins and TeamCity because it cannot really execute pipelines out of the box. It is possible, but in order to achieve that, the user has to do additional configurations out of the

VCS. These configurations have to be applied directly to the Travis infrastructure.

Usually, the tool comes with VCS webhooks integration out of the box. This allows the jobs to be executed every time a new commit or a merge request arrives. Some of the reasons that people depend on this CI are running tests, generating test reports, analyzing the results and providing code quality analysis. Each job is responsible for checking whether the code can compile and follow the predefined coding standards.

The configuration of Travis itself is placed in the Travis YAML file, which is part of the repository. There are predefined steps that can be used as hooks and the user can execute different commands in them. After the job has completed, a detailed report is generated and is available for the maintainer of the repository.

Unfortunately, Travis does not provide any free version. In order to explore the functionalities of the tool, the users can take advantage of the free trial version for 30 days and then they should either purchase a license or cancel their subscription.

We can see from the research above that all of the platforms provide a set of the same features which we can label as core functionalities. All of these features are illustrated below in the Table 1.

Table 1. Feature comparison

	TeamCity	TravisCI	Jenkins
Web Hooks	✓	✓	✓
Git Operations	✓	✓	✓
Use CLI	✓	✓	✓
Create Downloadable artifacts	✓	✓	✓
Run SSH remote command	✓	✓	✓
Works with microservice architecture	✓	X	✓
Has REST API	✓	X	✓
Support master-slave workers or agents	✓	X	✓
Has web UI	✓	✓	✓
Can be self-hosted	✓	✓	✓

By reviewing the table [Table 1] above, we can come to the conclusion that most of the systems implement the following features out of the box:

- Webhooks
- Git operations
- Downloadable artifacts
- SSH commands
- Web UI
- Self-hosting

3. Extended Model of Code Orchestration Platform

In order to adapt the currently existing core functionalities and to expose an extendable architecture for continuous integration [2] and continuous delivery [3], we can take a look at the following diagram [Figure 1]:

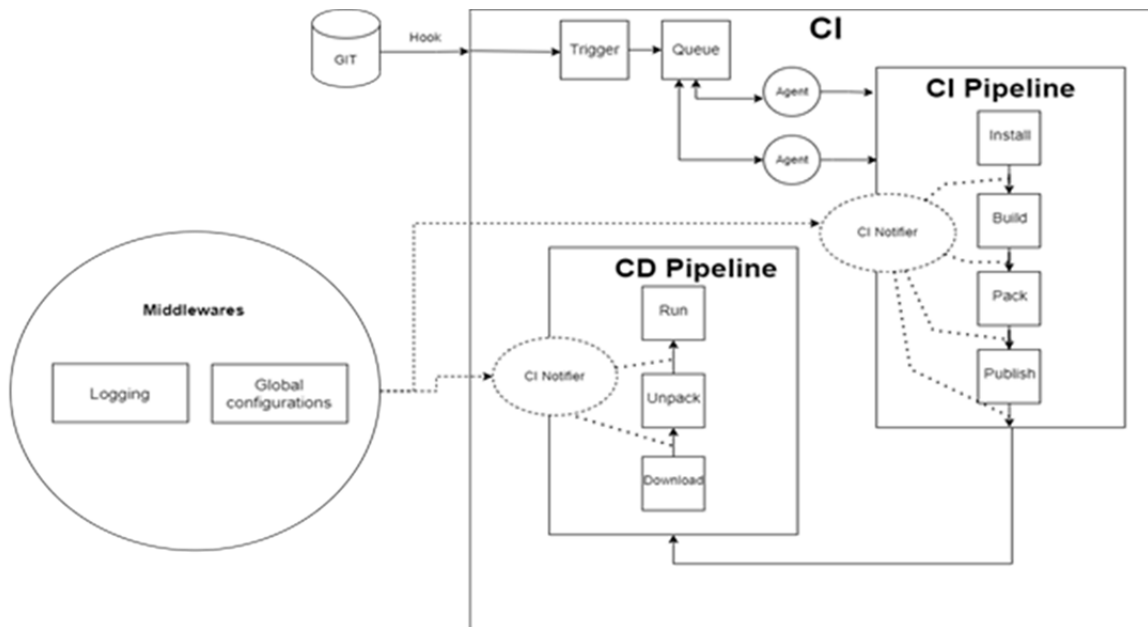


Figure 1. Extendable architecture for continuous integration

3.3. Agents

The agents are the job runners of the CI. Their responsibility is to execute already configured tasks that are chained in a pipeline. Each agent is responsible for a single job at a certain point of time and whenever the job completes, he claims the next one in the queue.

3.1. Git and Hooks

The process of CI/CD starts with a notification to the CI tool from the version control system. The hooks are responsible for providing a channel for communication between the source, which is located in the version control system and the job runner, which is the CI.

3.2. Queue

The queue is a part of the core functionality of the CI tool. It is responsible for handling pending jobs until an agent is free to claim them. The jobs are added and then assigned to an agent by the order they entered the queue (e.g. the oldest job is with the highest priority).

3.4. Pipeline

The pipeline is a sequence of jobs that needs to run consecutively. Each pipeline is specifically defined with a configuration that gives a detailed description of each step and how it should be executed. Whenever the pipeline finishes it can produce artifacts.

3.5. CI Notifier

The role of the CI Notifier is to expose events that can be fired after a specific action or step from the CI infrastructure has started or completed. These steps can be part of a pipeline, system events or any custom-made hooks. The notifier will allow external middlewares or plugins to be registered as listeners and to execute additional tasks without changing and affecting the existing pipelines. The number of events that the notifier will expose depends on the amount of stages that we define for a specific pipeline.

3.6. Extendable Middlewares

The extendable middleware concept has to handle different add-ons that may be registered as listeners to events exposed by the notifier. There should be an orchestrator that has to take care of the plugin's lifecycle and to allow adding and removing plugins without any additional changes to the main flow. The main purpose of these middlewares can be to add additional functionalities based on the artifacts that are generated from the execution steps.

3.7. Use Case Scenarios

With the example architecture above we have introduced a new concept of event handling by implementing a CI Notifier module. This provides an extendibility that was otherwise missing in the base set of features. We are going to provide an example that will show the use of this additional module.

We can say that we need to implement a reporting tool which will generate statistics based on the statuses of our CI operations. We have decided to store the raw data in ElasticSearch and we need to find a way to gather that data from the CI. In order to achieve this, we can add a CI Notifier to the CI implementation. We will then create a few events and hook them before and after every stage of our pipeline. They will listen for artifacts that are generated on success and on error of each step and will expose these artifacts to whoever needs them. By having these exposed events we can now attach a custom-made middleware that will contain instructions of how to process these artifacts and upload them to Amazon. It will be very easy to attach them because we do not need to modify the existing pipeline. We just need to have access to the already created artifacts. This means that we can plug and unplug these middlewares whenever we want.

4. Conclusion

In modern times, there is a problem in the process of software development because there are a lot of repetitive actions that need to be performed constantly. There are products that exist and provide a solution for that problem by taking advantage of continuous integration. The platforms that we reviewed in this article manage to solve that problem by adopting different approaches for automation depending on the context. They are straightforward and do not require any specific domain knowledge from the user. Unfortunately, not all of them are free to use it. This means that the companies have to decide whether they would like to invest in such a tool or not.

However, even though all of the platforms provide a good set of features, none of them are easily extendable. As a result of this research, we were able to provide a conceptual solution that will allow the CI platforms to be easily accessible by external middlewares. This concept will enrich the core infrastructure with additional opportunities without the need of any main changes.

References

- [1]. Arachchi, S. A. I. B. S., & Perera, I. (2018, May). Continuous integration and continuous delivery pipeline automation for agile software project management. In *2018 Moratuwa Engineering Research Conference (MERCon)* (pp. 156-161). IEEE.
- [2]. Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: improving software quality and reducing risk*. Pearson Education.
- [3]. Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.
- [4]. Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5, 3909-3943.
- [5]. Saidani, I., Ouni, A., Mkaouer, M. W., & Palomba, F. (2021). On the impact of Continuous Integration on refactoring practice: An exploratory study on TravisTorrent. *Information and Software Technology*, 138, 106618.
- [6]. Martin Heller. (2020). What is Jenkins? The CI server explained. Retrieved from: <https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html> [accessed: 13 February 2022].
- [7]. Saurabh, (2021), What is Jenkins? Jenkins For Continuous Integration, Retrieved from: <https://www.edureka.co/blog/what-is-jenkins/> [accessed: 13 February 2022].
- [8]. GitHub/Jenkins. (2019). Setting up Webhooks. Retrieved from: <https://gcube.wiki.gcube-system.org/gcube/GitHub/Jenkins: Setting up Webhooks> [accessed: 15 February 2022].
- [9]. Himanshu Sheth. (2020). Jenkins vs Travis | Which CI/CD Tool Is Best for You?, Retrieved from: <https://www.lambdatest.com/blog/travis-ci-vs-jenkins/> [accessed: 16 February 2022].
- [10]. Himanshu Sheth, (2020), TeamCity vs. Jenkins: Picking The Right CI/CD Tool. Retrieved from: <https://www.lambdatest.com/blog/teamcity-vs-jenkins-picking-the-right-ci-cd-tool/> [accessed: 17 February 2022].
- [11]. Nishant Sharma, (2021), How to set up a build pipeline on JetBrains TeamCity?, Retrieved from: <https://medium.com/testvagrant/how-to-set-up-a-build-pipeline-on-jetbrains-teamcity-41a1b0a67d76> [accessed: 17 February 2022].

- [12]. Aksakalli, I. K., Çelik, T., Can, A. B., & Tekinerdoğan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software, 180*, 111014.
- [13]. Watson, P. (2016). Continuous Integration with Teamcity. CreateSpace Independent Publishing Platform.
- [14]. Mahalingam, M. (2014). *Learning Continuous Integration with TeamCity*. Packt Publishing Ltd.
- [15]. Leszko, R. (2017). *Continuous Delivery with Docker and Jenkins*. Packt Publishing Ltd.
- [16]. Belmont, J. M. (2018). *Hands-On Continuous Integration and Delivery: Build and release quality software at scale with Jenkins, Travis CI, and CircleCI*. Packt Publishing Ltd.
- [17]. Laster, B. (2018). *Jenkins 2: Up and Running: Evolve Your Deployment Pipeline for Next Generation Automation*. "O'Reilly Media, Inc."
- [18]. Soni, M., & Berg, A. M. (2017). *Jenkins 2. x Continuous Integration Cookbook: Over 90 recipes to produce great results using pro-level practices, techniques, and solutions*. Packt Publishing Ltd.