ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

Review Article

Open Access

Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects

Naveen Muppa

10494 Red Stone Dr Collierville, Tennessee, USA

ABSTRACT

Lately, the software development industry is going through a slow but real transformation. Software is increasingly a part of everything, and, software developers, are trying to cope with this exploding demand through more automation. The pipelining technique of continuous integration (CI) and continuous delivery (CD) has developed considerably due to the overwhelming demand for the deployment and deliverability of new features and applications. As a result, DevOps approaches and Agile principles have been developed, in which developers collaborate closely with infrastructure engineers to guarantee that their applications are deployed quickly and reliably. Thanks to pipeline approach thinking, the efficiency of projects has greatly improved. Agile practices represent the introduction to the system of new features in each sprint delivery. Those practices may contain well-developed features or can contain bugs or failures which impact the delivery. The pipeline approach, depicted in this paper, overcomes the problems of delivery, improving the delivery timeline, the test load steps, and the benchmarking tasks. It decreases system interruption by integrating multiple test steps and adds stability and deliverability to the entire process. It provides standardization which means having an established, time-tested process to use, and can also decrease ambiguity and guesswork, guarantee quality and boost productivity.

***Corresponding author**
Naveen Muppa, 10494 Red Stone Dr Collierville, Tennessee, USA.

Received: May 07, 2023; **Accepted:**May 14, 2023; **Published:** May 22, 2023

Keywords: Agile Configuration Management, Containerization, Continuous Delivery, Continuous Integration, Git, Version Management

Introduction

The development of a product or service is achieved through iterations or rapid development cycles carried out in a shorter time. In other words, instead of a monolithic development strategy, agile practices involve working on chunks of projects at the same time, which makes changes and adjustments easy and manageable. Initially, the developers will analyze the whole journey a functionality is expected to take from the moment it is born, to how it is defined and prioritized over other functionalities, how the team that will implement it is chosen, how resources are allocated, and how the work is planned. Only after all these stages are complete, will the developers look at the implementation.

Continuous Integration

A widely used software development practice is one in which developers integrate code into a shared repository multiple times a day to quickly obtain feedback on the viability of that code. CI supports automated builds and tests, so teams can quickly collaborate on a single project. Also, CI enables software companies to have a frequent and shorter release cycle.

This strategy facilitates a quick and reliable launch of the program in production by utilizing the available practice sets. All of this is due to the regular merging of operational software copies, which eliminates and decreases software integration difficulties and, hence, expenses. Adherents of CI urge their development teams to create software in short iterations and to merge their functional code into the root code as quickly as is feasible.

Continuous Delivery (CD)

Continuous delivery is a software engineering practice in which teams design, build, test, and release software in short cycles. It relies on automation at each stage to ensure the cycle is both fast and reliable. It employs a set of practices and automatically deploys and delivers software to a production-like environment.

Basically, the CD is a software development strategy that automates the process by which changes made by an application developer are delivered to the code repository or the container registry, and it shows how the changes are automatically tested for errors. Thus, all the changes can be deployed in a live production environment by the operations team. By doing so, CD solves the limited visibility and communication issue between DevOps and business teams. To that end, the goal of CD is to guarantee that implementing new code requires as little work as possible.

Proposed Solution Architecture

The pipeline generator discovers build and deployment files in the git repositories and automatically creates pipeline jobs to execute them. To ensure that pipelines that inherit the CI/CD templates are always up to date, the pipeline will auto-regenerate on merge requests.

The generator creates an extra job that executes the regenerate pipeline script:

- the pipeline configuration files (i.e., *.gitlab-ci.yml* and the files from the *.ci/ folder*) are regenerated with the latest stable templates;
- if there are changes, the pipeline configuration files are pushed back to the remote HEAD, and the current pipeline is forced to terminate.

A new pipeline is triggered each time the pipeline configuration file is changed, but only for Merge Requests. Any changes made to the *.gitlab-ci.yml* or files from the *.ci/* folder other than those from the pipeline generator are ephemeral. Every time a Merge Request pipeline has been executed any changes to it will be lost. The only exception is the *.ci/custom.yml* file, which can contain any custom jobs, and which is persistent between pipeline regeneration.

The pipeline generator has two ways to regenerate: one if it is used with the default template jobs and the if it is used with custom jobs. The default jobs are basically jobs that are executed on most of the git repositories: code versioning, creating an AWS ECR repository, building Docker images, and deploying Helm charts on Kubernetes. These types of jobs are the basis of any pipeline and provide a centralized way to version, build, and deploy the code. While projects have much in common, there are still actions that are specific to each environment. Some projects may require uploading a file to an AWS S3 bucket, while others may need to run an extended test suite. To ensure the flexibility of this framework, the ability to extend the default pipeline was added. To include custom jobs in the pipeline, the need to add them to a file called *.ci/custom.yml* arises.

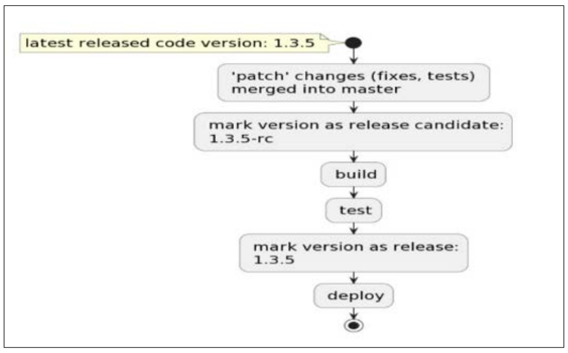
latest released code version: 1.3.5
'patch' changes (fixes, tests) merged into master
mark version as release candidate: 1.3.5-rc
build
test
mark version as release: 1.3.5
deploy

Figure 1: Solution Architecture

Versioning

This flow is performed first and is triggered when a git commit is merged into the master: the pipeline creates a lock for versioning so that other pipelines will not start until versioning is complete. If the commit contains deployable code changes, the commit is marked as a release candidate, then the pipeline versioning lock is released. Further, the test suites are executed, and if all the tests have passed, then the commit is marked as a stable release. Exemplifies the entire flow, while represents a code block from the versioning Bash script.

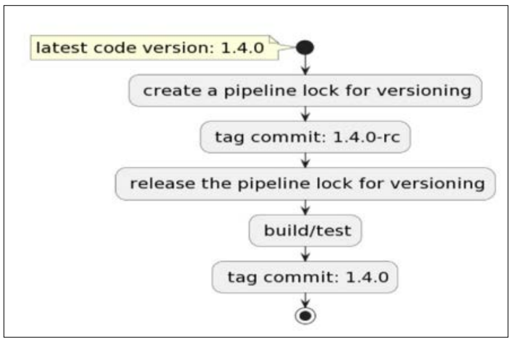
latest code version: 1.4.0
create a pipeline lock for versioning
tag commit: 1.4.0-rc
release the pipeline lock for versioning
build/test
tag commit: 1.4.0

Figure 2: Versioning

Build

In the following, the CI principles in just one pipeline are detailed. If there are any buildable code changes, the pipeline triggers a job that builds an image from the Docker file, and the build context as the root of the project. The image is tagged with two tags: a stable tag (the prefix—e.g., rc—and the branch name—e.g., rc-master), and a unique tag (the prefix—e.g., rc—and the branch name and the commit hash—e.g., rc-master-adc11dad). The image labels are updated in this way: the generic labels are appended to the image labels and, afterward, the image is published to the Container Repository: the AWS Elastic Container Registry. This flow expects to have one or more Docker files in the repository and will do the same steps for all of them.

An image is promoted only on the master branch when there are deployable code changes, by adding the semantic version tags to the latest image built on this branch and pushing the image to the Container Repository, which can be from different providers.

In below, an entire build step with its dependencies is exemplified. Is a pipeline diagram provided by GitLab, which describes more clearly the processes that are part of the pipeline. Shows an alternative way of describing the pipeline by also showing the jobs' dependencies.

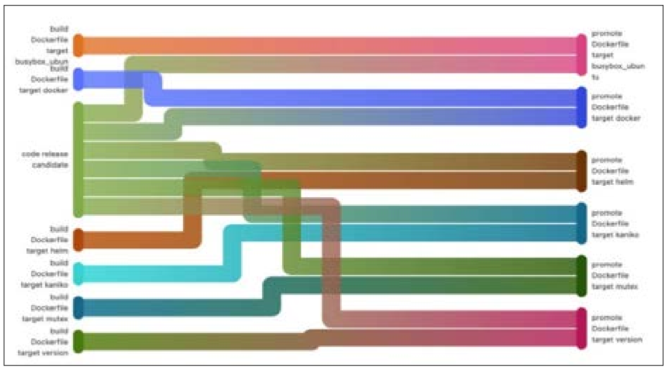
build Dockerfile target busybox_ubun
build Dockerfile target docker
code release candidate
build Dockerfile target helm
build Dockerfile target kaniko
build Dockerfile target mutex
build Dockerfile target version
promote Dockerfile target busybox_ubuntu
promote Dockerfile target docker
promote Dockerfile target helm
promote Dockerfile target kaniko
promote Dockerfile target mutex
promote Dockerfile target version

Figure 3: Build

Deploy

This subsection covers the CD principles in one pipeline. It verifies whether there are any object definitions or values files. The deployment pipeline is automatically triggered if it detects any such files. This pipeline contains the linting step, which is executed to test the chart with a specific values file, and contains the version update step, which updates the application and chart versions in Chart.yaml file.

If the deployment step fails, a rollback job is triggered to bring the application back to the stable variant.

This solution can be easily adapted to any type of preference or any type of customization. It is easy to use it because the developed code can be easily integrated with multiple third parties.

Figure represents a diagram flow, presenting the entire automated pipeline which was described in this section.

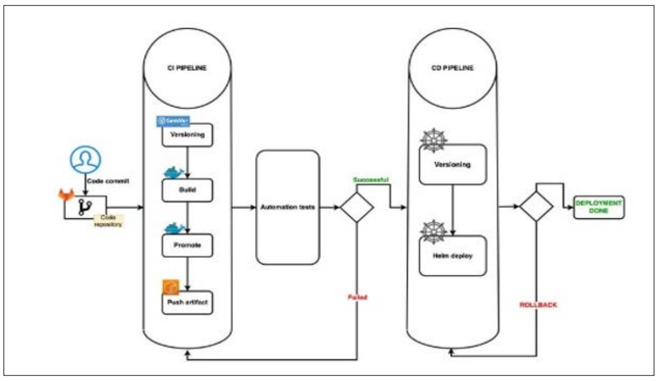

Figure 4: Deploy

Conclusion

Continuous integration systems help project members focus various resources on key issues, thereby reducing development time and improving software quality. The development team can spend more resources on software design; the double integration work is undertaken by machine. Continuous rapid feedback enables testers to be adequately tested. The continuous integration delivery system is a breakthrough for automated operation and maintenance. It helps to improve the maturity of software projects, implement continuous improvement of lean processes, promote the improvement of software service levels, and promote high-quality development of software systems through operation and maintenance mechanisms.

This paper presents a complex and automated pipeline generator with CI/CD principles for the deployment of multiple types of applications. The solution is based on Agile practices, which are responsible for the automatic integration, testing, and delivery of features for applications.

The proposed solution serves as a baseline for common CI/CD tasks and encapsulates the following specifics: all code must be versioned by semantic versioning standards; builds are created automatically by using Docker and the Docker layers are cached for later reuse; most deliverables are submitted to the Docker Container Registry; and most deliverables are deployed to a Kubernetes cluster via Helm. These practices ensure high availability with no downtime, fast and easy scalability, rolling back automatically to a stable version, scanning vulnerabilities in Docker, detecting any change in the application source code, and triggering an entire chain of actions and events based on what has been changed. If there are changes on the infrastructure manifests, but not on the application code, the process of building and testing pipelines will not be triggered, thereby the same artifacts' "pollution" is brought down. This feature, even if the same artifacts have distinct tag, leads to higher speed on pipeline duration [1-3].

References

1. Hodgson P (2024) Continuous Delivery in the Wild. O'Reilly Split https://www.split.io/continuous-delivery-in-the-wild/?utm_campaign=LP-OReilly-CD-in-the-Wild-PPC&utm_source=google&utm_medium=paid+search&utm_content=US_Deployment&utm_term=build%20and%20deployment%20automation&gclid=CjwKCAjwmrqzBhAoEiwAXVpgou9lP7_bBky519HvruSN5oMhbfGx56siCAKRCXxQ8AzsGg_IY25NERoC2XQQAvD_BwE&gad_source=1.
2. (2024) Agile 101: Using Agile project management methods to deliver customer value. ServiceNow https://www.servicenow.com/lpebk/agile-project-management.html?campid=107374&cid=p:spm:dg:nb:prsp:phr:Google_Other:ams:all&s_kwcid=AL!11692!3!648256817419!p!!g!!agile%20methodology%20process&ds_c=GOOG_AMS_All_EN_DEMANDGEN_SPM_PRSP_NonBrand_PHR_Other&cmcid=71700000102278031&ds_ag=Agile+Methodology_PHR&cmpid=58700008156364558&ds_kids=p79372254482&gad_1ACPaS9K3LEMDq7wH5LVPPbwXEw1MJGrOOT3jie1mQvN7RoCvEAQAvD_BwE&gclsrc=aw.ds.
3. Cheat Sheet: 8 Tips for Securing Your CI/CD Pipeline. snyk https://go.snyk.io/cicd-security-cheatsheet.html?utm_medium=paid-search&utm_source=google&utm_campaign=nb_lg_ci-cd-pipeline&utm_content=devsec&utm_term=devsecops%20ci%20cd&gad_source=1&gcli d=CjwKCAjwmrqzBhAoEiwAXVpgonbs9bBf83taG9QOfayZRhhvRfHCOcs9QiyFYcV GnFKKbAepp8m7PRoC8sIQAvD_BwE.