ISSN 2063-5346 *Section A-Research paper*

Pipeline-Generated Continuous Integration And Deployment Method For Agile Software Development

Arvind Kumar Bhardwaj, Sandeep Rangineni, Latha Thamma Reddi, Manoj Suryadevara, Krishnakumar Sivagnanam

[1]Senior Software Architect, Information Technology, Capgemini, Texas, USA, ORCID: 0009-0005-9682-6855

[2] Data Test Engineer, Information Technology, Pluto TV, California, USA, ORCID: 0009-0003-9623-4062

[3]Sr Product and Portfolio Manager, Information Technology, DXC Technology, Texas, USA, ORCID: 0009-0005-6338-7972

[4]Staff Product Manager, Information Technology, Walmart, Arkansas, USA, ORCID: 0009-0007-8738-2222

[5]Solutions Architect, Information Technology, Tech Mahindra (Americas) Inc, Virginia, USA, ORCID: 0009-0004-2843-182X

ABSTRACT

Because of this, DevOps and Agile concepts were established to ensure that applications can be delivered rapidly and reliably by fostering tight collaboration between developers and infrastructure engineers. The pipeline strategy has helped increase the effectiveness of initiatives. New features are added to the system with each sprint delivery, which is what agile approaches represent. These procedures might have fully formed features or they could have flaws or errors that have an effect on the delivery. This article presents a pipeline strategy for solving delivery issues, which speeds up the delivery process, simplifies testing, and enhances benchmarking. It increases stability and deliverability, reduces system disruption, and integrates numerous test phases. Because it is written in Bash, an interpreted language, this tool may be easily integrated with other systems. We show the value that our solution presently generates based on experimental data. This solution offers a fast and easy method for creating, managing, and automating automated pipelines for CI and CD projects based on the Agile methodology. Standard CI/CD processes may be built upon the suggested solution, stores reusable Docker layers in a cache, and uses Helm to deploy highly available outputs to a Kubernetes cluster. Altering the solution's fundamental concepts and bringing it to more platforms (windows) will be discussed later.

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

1.0 Introduction

The needs of current enterprises cannot be met by the methods of software development of the past. The agile technique appeals to software development firms because it facilitates a more responsive, effective, and quick software development life cycle. Experts and companies alike are so interested in developing and automating a unified solution for CI, CD, and CDT (“Continuous Integration, Continuous Delivery, and Continuous Deployment”).A product or service may be brought to market more quickly by using rapid development cycles, often known as iterations. That is to say, instead of tackling one huge project at a time, agile methods break down the tasks into smaller, more manageable chunks. Among the numerous advantages of the CI technique, the removal of maximum iterations per implementation and the decreased chance of unsuccessful implementations stand out. Businesses are more likely to invest in CD because it helps them get their products to market faster, improves quality, makes customers happier, help them avoid legal trouble, and increases efficiency and productivity. The bulk of software and mobile app development is currently hosted on infrastructure as a service (IaaS), making CI, CD, and CDT unquestionably important in the cloud [3].It is sometimes hard to revert to a previous version of a system after it has been deployed or after a migration has been completed, even if the system has been carefully developed [4]. The problem may be easily addressed by changing the size of the necessary systems due to the increased emphasis on achieving delivery deadlines through the usage of agile methods. There is also the possibility of including rollback into the pipeline, which would cause the system to revert to a previous state if the monitoring thresholds or basic checks fail [6].This article presents a blueprint for establishing CDT pipelines that are both automated and continuously integrated. It explains why it takes time and careful engineering to create delivery pipelines that are both flexible and high-quality [7].The remaining sections of the paper are organized as shown above.

2.0 REVIEW OF LITERATURE

To address product and business difficulties, the IT sector has increasingly adopted the CI/CDT/CD methodology for application delivery and deployment. Using Git-stored declarative configuration files, Argo CD can automatically deploy software applications to Kubernetes. This makes it possible for Git to be used for version control and auditing of the configuration. As per Dr.Naveen Prasadula Cluster-based tool whose primary function is to monitor an off-site Git repository and deploy new Kubernetes manifests as they become available. Because it integrates so many tools into a single package, it eliminates the need to adopt and set up separate programs to implement the various CI/CD best practices. This article addresses every facet of standard Agile organizations by proposing a solution that incorporates all of these features: a pipeline generator, version control, continuous integration, and continuous deployment.

2.1 Basic Ideas

CI allows for swift collaboration on a single project by automating the build and testing processes. Another benefit of CI is that it shortens and increases the frequency of software release cycles [14].

2.2 Constant Distribution (CD)

In software engineering, the term "continuous delivery" refers to a methodology in which software is developed, tested, and released at regular intervals. The speed and consistency of the cycle are guaranteed by the extensive use of automation throughout. It uses a series of procedures to deploy and deliver software to a staging area that mimics production [14].As an alternative to manual installation, an op-prem cloud solution might automatically deploy the packed program. Continuous Deployment refers to the procedure through which application updates are automatically sent out to live servers [20, 22].

2.3 CI/CD Pipeline

GitLab reduces development expenses and security concerns while speeding up product delivery from weeks to minutes. Some of Gitlab's tools are discussed below; using them makes this solution readily accessible and easy to use in any project or business [23].

2.4 Archiving and Repository Management

To keep track of and organize modifications made to software, a process known as "version control" or "source control" is used. Software engineers use version control systems, or VCSs, to keep track of past and present iterations of their code. Git is the most widely used VCS that support CI/CD workflows. A repository is a centralized database that stores and maintains large amounts of information in an orderly manner.

2.5 Robotic Tests

Unit, functional, and performance testing are all a part of a well-orchestrated test automation framework. A dependable code base, quicker reaction times, and less complicated decision-making are all advantages of the continuous testing method. The suggested solution [32] offers the possibility of adding automation testing.

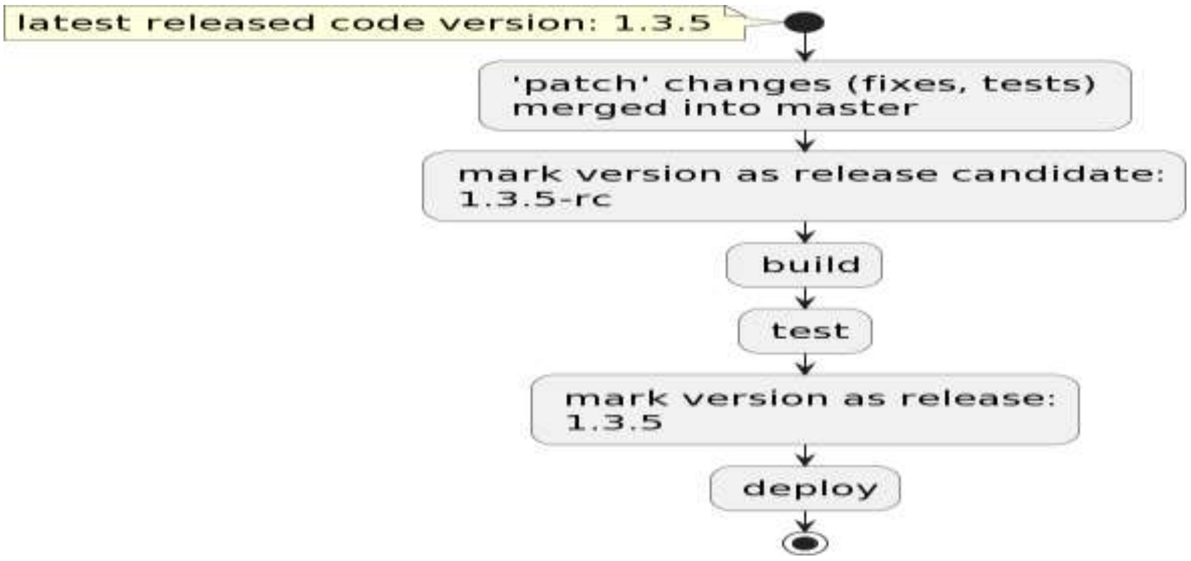
2.6 Methods of Deployment

Helm, which is built on Kubernetes, is the first application package management [34]. The application's architecture may be described using helm-charts, and it can be controlled using simple commands.

2.7 New Solution Structure Proposed

The primary phases of the proposed solution's pipeline are shown in a straightforward UML schematic

(Figure 1) below.


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

The suggested solution is shown in Figure 1 as a UML diagram.

Table 1 lists the available flows, and Figure 2 provides a more in-depth description of the pipeline generator's UML Diagram.

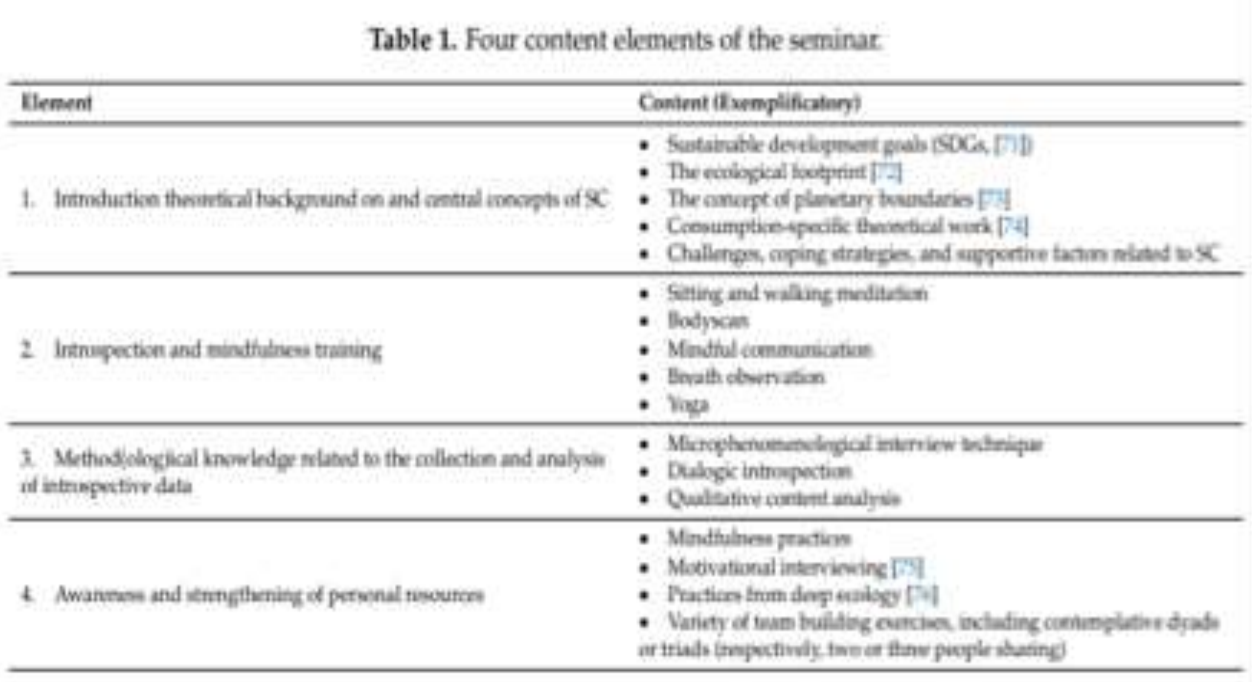
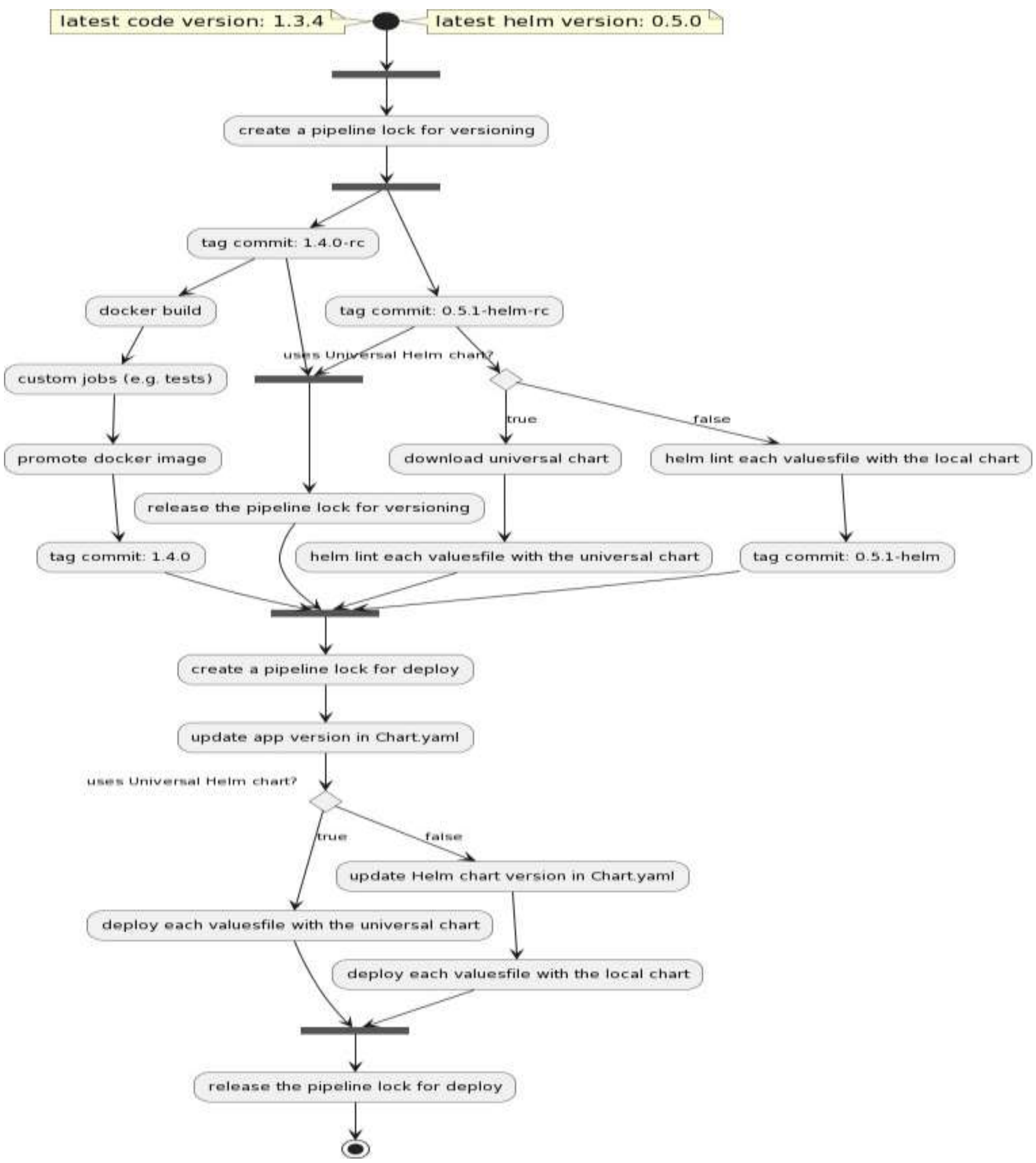


Table 1. Four content elements of the seminar.

Element	Content (Exemplificatory)
1. Introduction theoretical background on and central concepts of SC	• Sustainable development goals (SDGs, [71]) • The ecological footprint [72] • The concept of planetary boundaries [73] • Consumption-specific theoretical work [74] • Challenges, coping strategies, and supportive factors related to SC
2. Introspection and mindfulness training	• Sitting and walking meditation • Bodyscan • Mindful communication • Breath observation • Yoga
3. Methodological knowledge related to the collection and analysis of introspective data	• Microphenomenological interview technique • Dialogic introspection • Qualitative content analysis
4. Awareness and strengthening of personal resources	• Mindfulness practices • Motivational interviewing [75] • Practices from deep ecology [76] • Variety of team building exercises, including contemplative dyads or triads (respectively, two or three people sharing)


latest code version: 1.3.4
latest helm version: 0.5.0
create a pipeline lock for versioning
tag commit: 1.4.0-rc
docker build
tag commit: 0.5.1-helm-rc
uses Universal Helm chart?
custom jobs (e.g. tests)
true
false
promote docker image
download universal chart
helm lint each valuesfile with the local chart
release the pipeline lock for versioning
tag commit: 1.4.0
helm lint each valuesfile with the universal chart
tag commit: 0.5.1-helm
create a pipeline lock for deploy
update app version in Chart.yaml
uses Universal Helm chart?
true
false
update Helm chart version in Chart.yaml
deploy each valuesfile with the universal chart
deploy each valuesfile with the local chart
release the pipeline lock for deploy

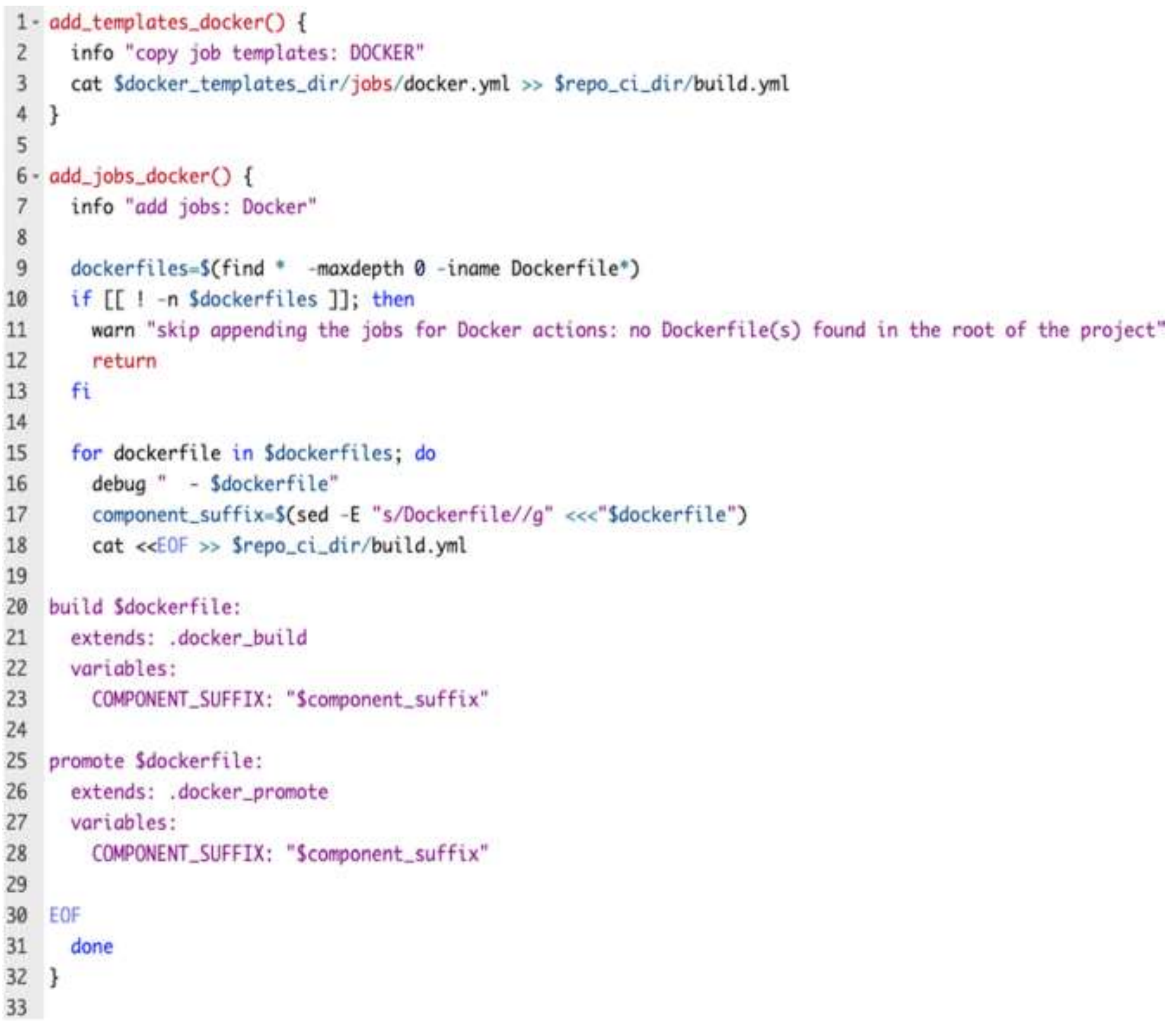
The solution is shown in a detailed UML diagram (Figure 2).

Detailed descriptions and explanations of the many proposed solution flows are provided below. The proposed application is a set of Bash scripts, one for each feasible procedure.

2.8 Synthesis of Pipelines

The pipeline generator can scan the git repositories for build and deployment files, and then generate the corresponding pipeline tasks. The.ci/custom.yml file stands alone; it may store any custom tasks and survives even after the pipeline is regenerated. These tasks form the backbone of any pipeline, since they allow for centralized management of code revisions, builds, and releases. While projects have many similarities, there are certain things that must be done in each setting. It's possible that your project calls for either running a comprehensive set of tests or uploading a file to an AWS S3 bucket. The framework's extensibility was ensured by using a custom pipeline extension mechanism. Adding custom tasks to the pipeline requires editing a file known as.ci/custom.yml.

For an example of the Bash script used to regenerate the pipeline, see Figure 3 below.



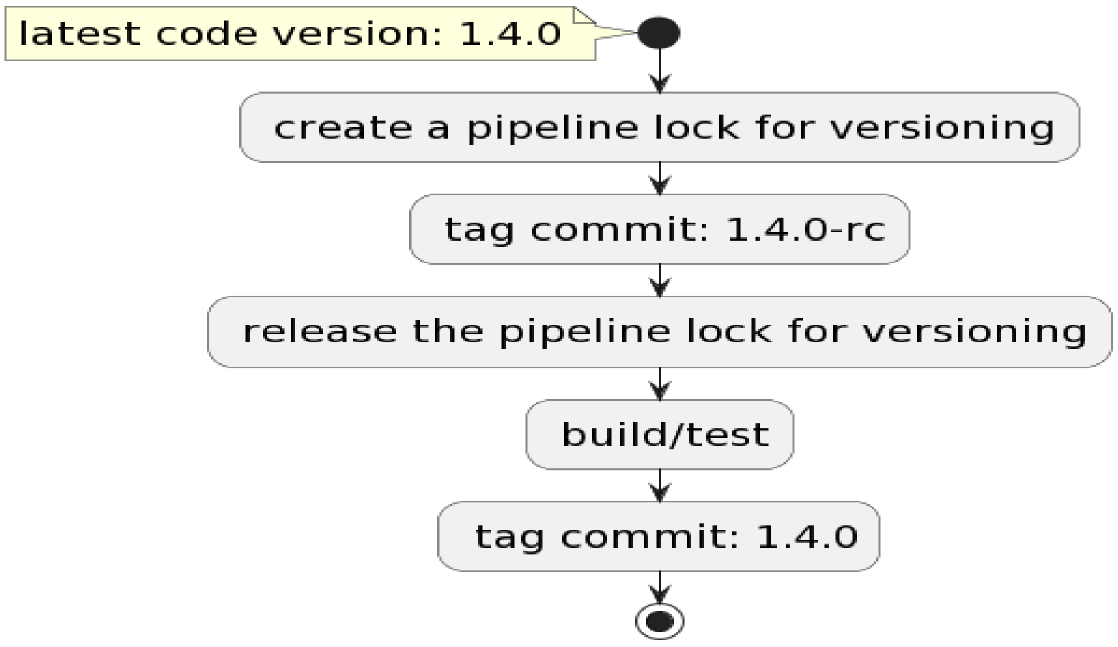
```
add_templates_docker() {
  info "copy job templates: DOCKER"
  cat $docker_templates_dir/jobs/docker.yml >> $repo_ci_dir/build.yml
}

add_jobs_docker() {
  info "add jobs: Docker"

  dockerfiles=$(find *  -maxdepth 0 -iname Dockerfile*)
  if [[ ! -n $dockerfiles ]]; then
    warn "skip appending the jobs for Docker actions: no Dockerfile(s) found in the root of the project"
    return
  fi

  for dockerfile in $dockerfiles; do
    debug "  - $dockerfile"
    component_suffix=$(sed -E "s/Dockerfile//g" <<<"$dockerfile")
    cat <<EOF >> $repo_ci_dir/build.yml

build $dockerfile:
  extends: .docker_build
  variables:
    COMPONENT_SUFFIX: "$component_suffix"

promote $dockerfile:
  extends: .docker_promote
  variables:
    COMPONENT_SUFFIX: "$component_suffix"

EOF
  done
}

```

Code Sample for a Pipeline Generator (Figure 3).

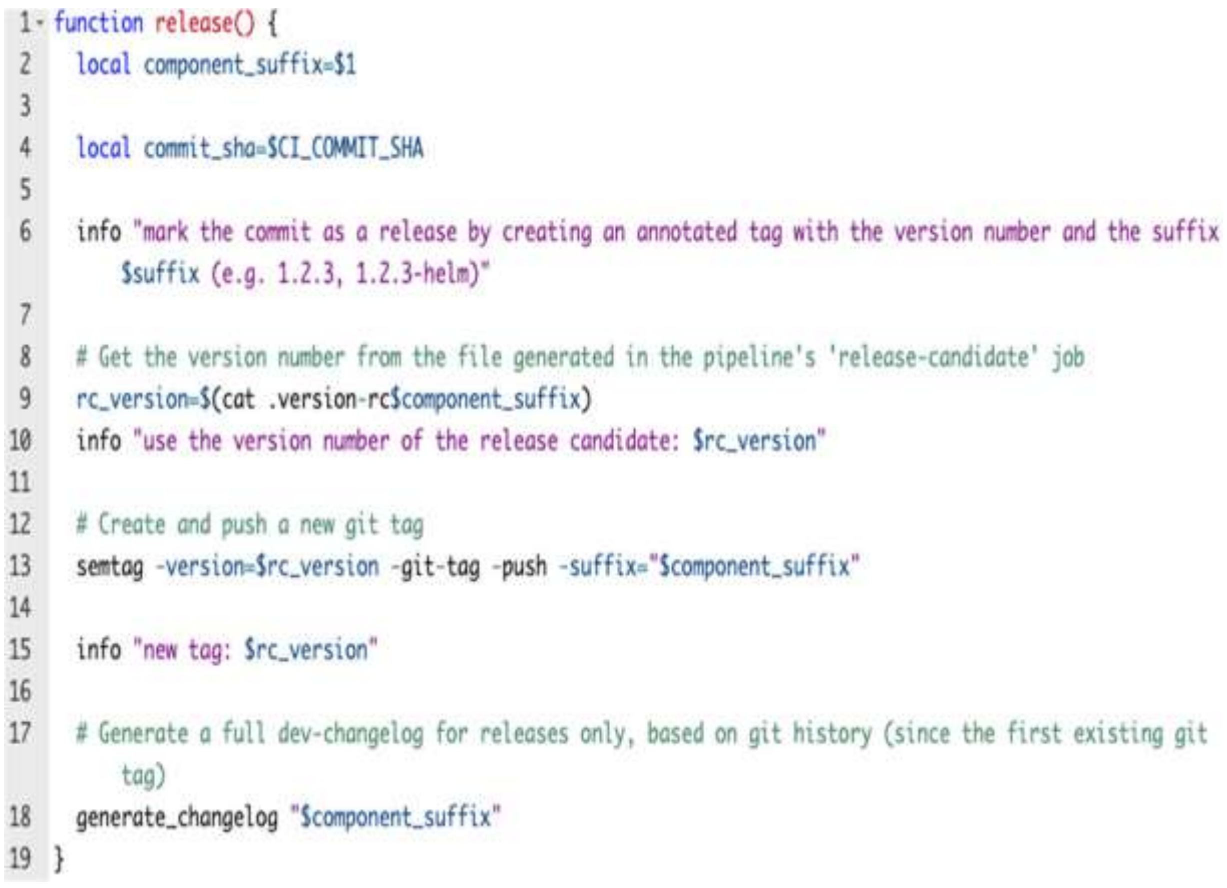
2.9 Versioning

The versioning pipeline is the first to execute once a git commit is merged into the master branch. Pipeline execution is blocked until versioning is complete because of the lock created by this pipeline. The pipeline versioning lock will be unlocked if the commit is a release candidate and includes deployable modifications. On top of that, test suites are run, and if they all pass, the commit is promoted to the stable branch.

Figure 4 depicts the whole procedure, whereas Figure 5 displays a fragment of the Bash script used for versioning.


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

The versioning process is shown in Figure 4.



```
function release() {
  local component_suffix=$1

  local commit_sha=$CI_COMMIT_SHA

  info "mark the commit as a release by creating an annotated tag with the version number and the suffix
     $suffix (e.g. 1.2.3, 1.2.3-helm)"

  # Get the version number from the file generated in the pipeline's 'release-candidate' job
  rc_version=$(cat .version-rc$component_suffix)
  info "use the version number of the release candidate: $rc_version"

  # Create and push a new git tag
  semtag -version=$rc_version -git-tag -push -suffix="$component_suffix"

  info "new tag: $rc_version"

  # Generate a full dev-changelog for releases only, based on git history (since the first existing git
     tag)
  generate_changelog "$component_suffix"
}
```

Figure 5 a fragment of the Bash script

The CI component of this solution is shown by the command shown in Figure 8 below, which is taken from the build Bash script.

The section describes the Docker caching layers, an additional feature of Docker images. The Dockerfile defines the collection of instructions and actions that make up each layer of a Docker image. Docker's support for "layers" makes it possible to break down a complex job into manageable chunks, so that just the layer corresponding to a given change in the code or program has to be updated. The proposed technique searches in order to cache the especially uncached levels and puts a premium on caching all the layers in order to speed up the construction process. By contrasting the times required to generate the same image with and without cache enabled, Figure 9 provides a thorough description of Docker caching.

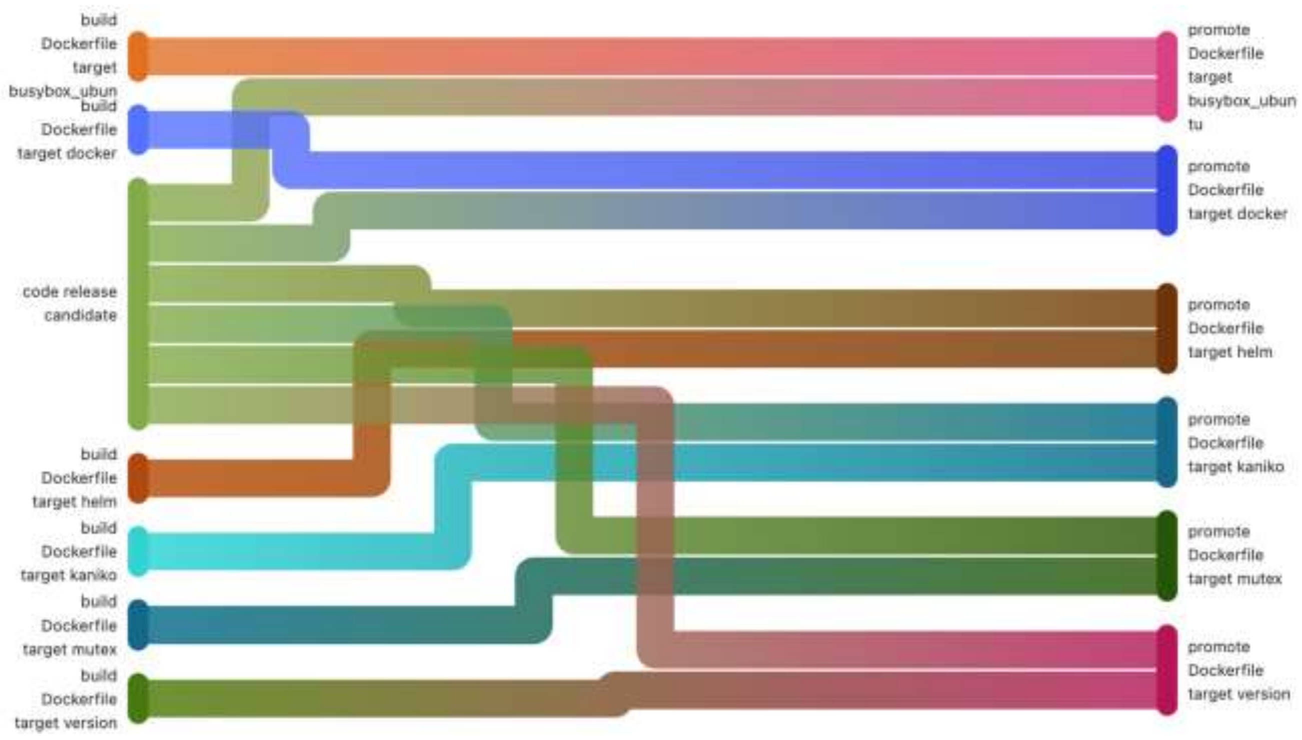
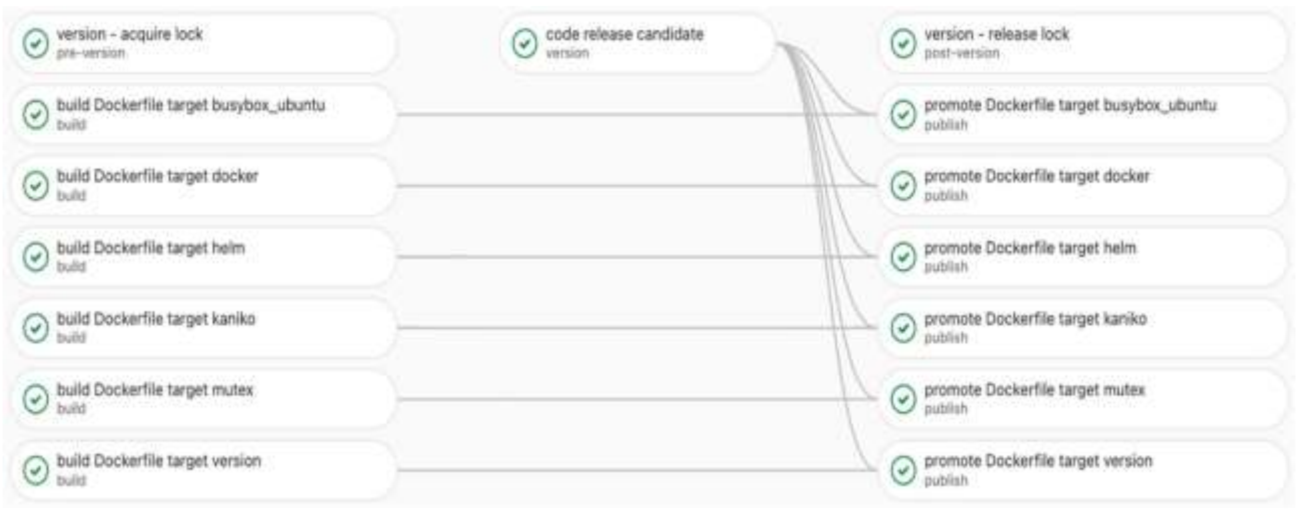

build Dockerfile target busybox_ubun
build Dockerfile target docker
code release candidate
build Dockerfile target helm
build Dockerfile target kaniko
build Dockerfile target mutex
build Dockerfile target version
promote Dockerfile target busybox_ubuntu
promote Dockerfile target docker
promote Dockerfile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

Figure 6: Flow of a pipeline schematic.


version - acquire lock
pre-version
build Dockerfile target busybox_ubuntu
build
build Dockerfile target docker
build
build Dockerfile target helm
build
build Dockerfile target kaniko
build
build Dockerfile target mutex
build
build Dockerfile target version
build
code release candidate
version
version - release lock
post-version
promote Dockerfile target busybox_ubuntu
publish
promote Dockerfile target docker
publish
promote Dockerfile target helm
publish
promote Dockerfile target kaniko
publish
promote Dockerfile target mutex
publish
promote Dockerfile target version
publish

Dependencies on flows in a pipeline diagram are shown in Figure 7.

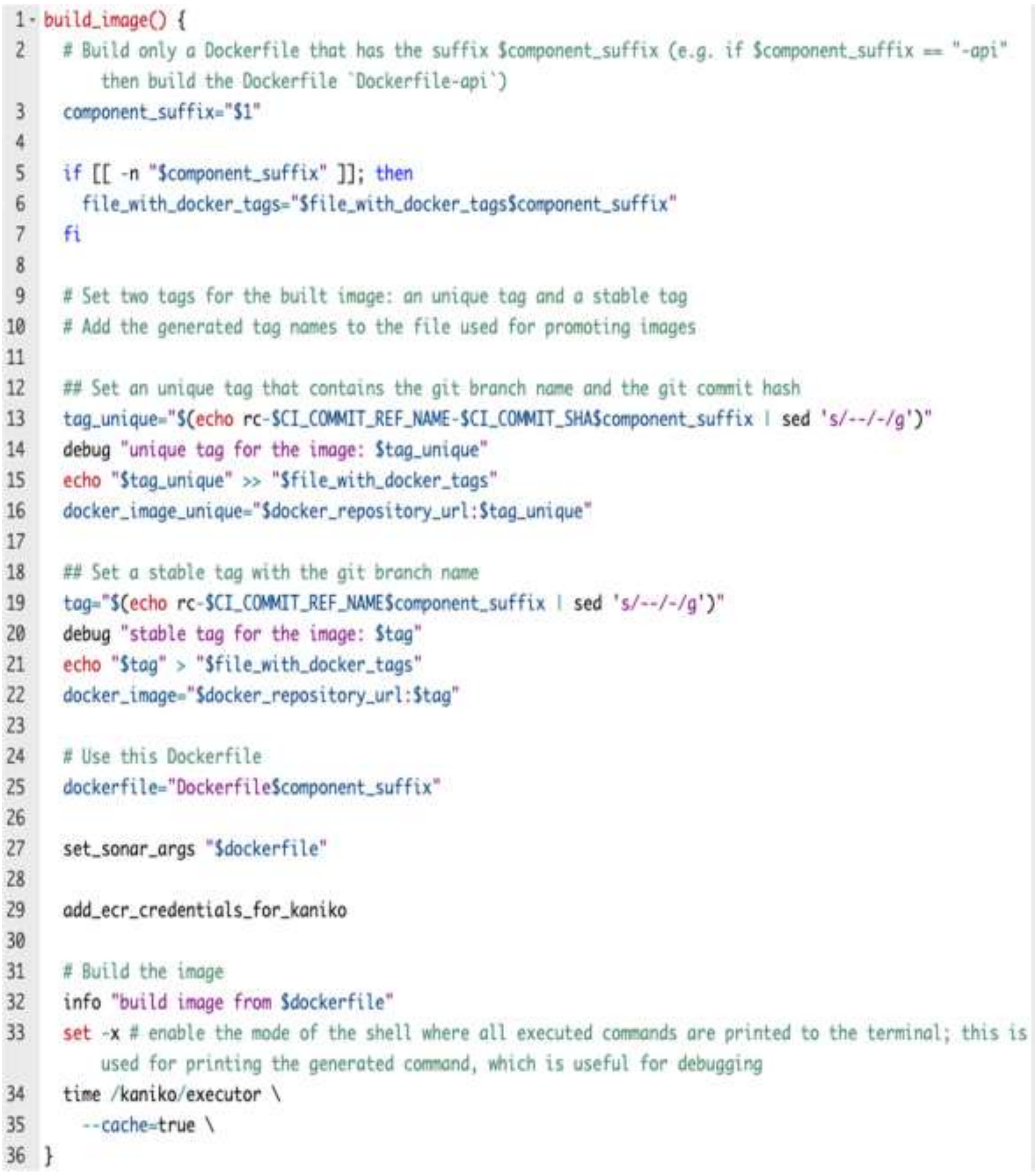
2.10 Deploy

The CD pipeline is discussed in this section in detail. It checks for the presence of files containing object definitions and values.

This method lends itself well to customization on a broad scale. The produced code is readily adaptable to the needs of many external entities, making its usage a breeze.

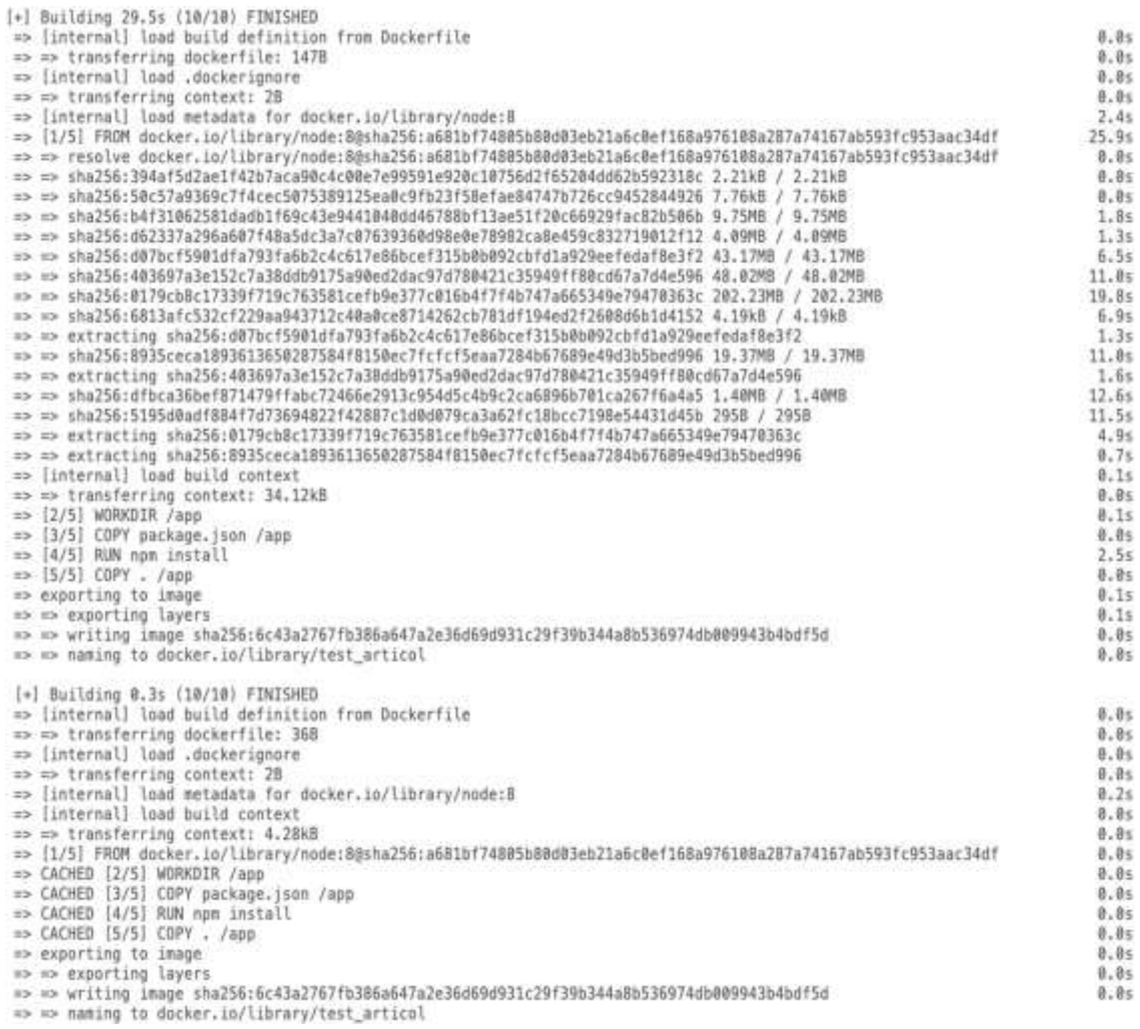
The full automated procedure discussed here is shown in flowchart form in Figure 10.

Figure 11 clearly depicts the deploy pipeline of the proposed solution, which includes the helm installation/upgrade procedure.



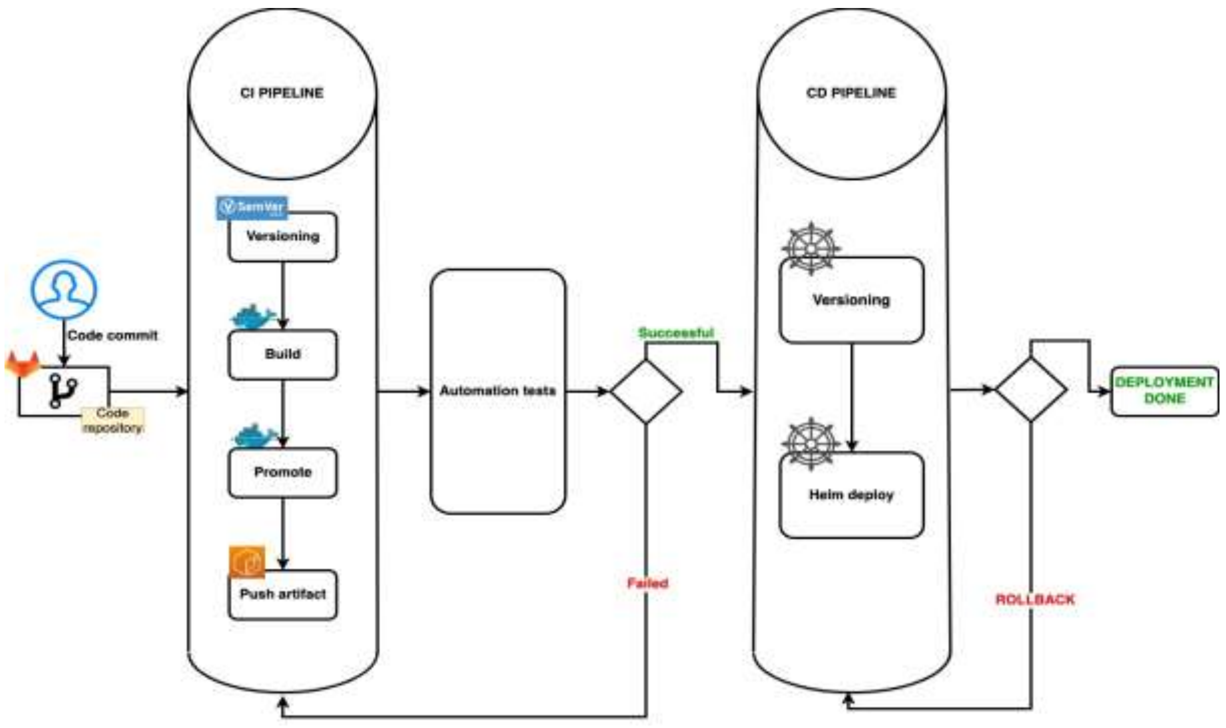
```
build_image() {
  # Build only a Dockerfile that has the suffix $component_suffix (e.g. if $component_suffix == "-api" then build the Dockerfile `Dockerfile-api`)
  component_suffix="$1"

  if [[ -n "$component_suffix" ]]; then
    file_with_docker_tags="$file_with_docker_tags$component_suffix"
  fi

  # Set two tags for the built image: an unique tag and a stable tag
  # Add the generated tag names to the file used for promoting images

  ## Set an unique tag that contains the git branch name and the git commit hash
  tag_unique="$(echo rc-$CI_COMMIT_REF_NAME-$CI_COMMIT_SHA$component_suffix | sed 's/--/-/g')"
  debug "unique tag for the image: $tag_unique"
  echo "$tag_unique" >> "$file_with_docker_tags"
  docker_image_unique="$docker_repository_url:$tag_unique"

  ## Set a stable tag with the git branch name
  tag="$(echo rc-$CI_COMMIT_REF_NAME$component_suffix | sed 's/--/-/g')"
  debug "stable tag for the image: $tag"
  echo "$tag" > "$file_with_docker_tags"
  docker_image="$docker_repository_url:$tag"

  # Use this Dockerfile
  dockerfile="Dockerfile$component_suffix"

  set_sonar_args "$dockerfile"

  add_ecr_credentials_for_kaniko

  # Build the image
  info "build image from $dockerfile"
  set -x # enable the mode of the shell where all executed commands are printed to the terminal; this is used for printing the generated command, which is useful for debugging
  time /kaniko/executor \
    --cache=true \
}
```

Image construction as seen in Figure 8.

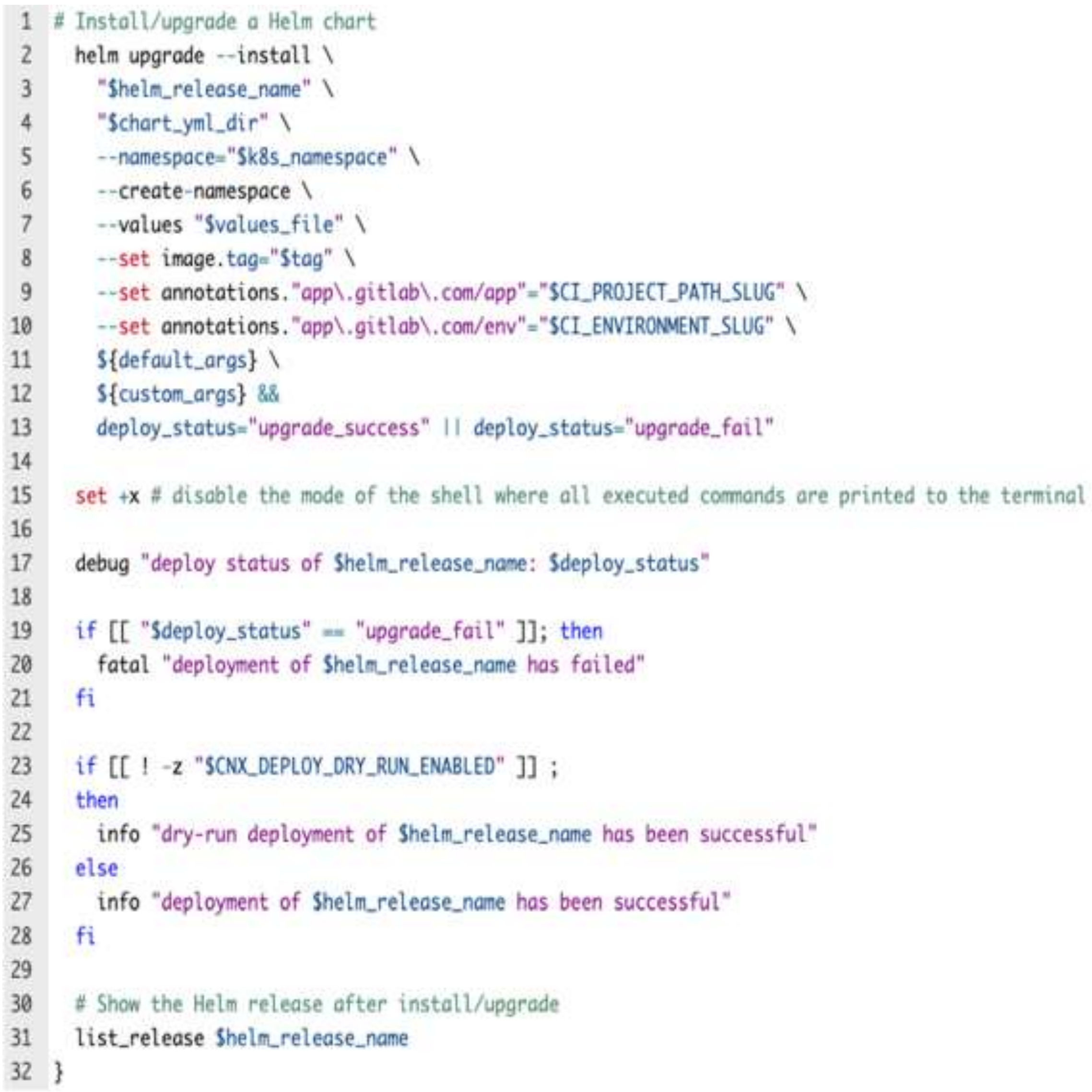


```
[+] Building 29.5s (10/10) FINISHED
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 147B                                                                                 0.0s
 => [internal] load .dockerignore                                                                                    0.0s
 => => transferring context: 2B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/node:8                                                            2.4s
 => [1/5] FROM docker.io/library/node:8@sha256:a681bf74805b80d03eb21a6c0ef168a976108a287a74167ab593fc953aac34df     25.9s
 => => resolve docker.io/library/node:8@sha256:a681bf74805b80d03eb21a6c0ef168a976108a287a74167ab593fc953aac34df      0.0s
 => => sha256:394af5d2ae1f42b7aca90c4c00e7e99591e920c10756d2f65204dd62b592318c 2.21kB / 2.21kB                       0.0s
 => => sha256:50c57a9369c7f4cec5075389125ea0c9fb23f58efae84747b726cc9452844926 7.76kB / 7.76kB                       0.0s
 => => sha256:b4f31062581dadb1f69c43e9441040dd46788bf13ae51f20c66929fac82b506b 9.75MB / 9.75MB                       1.8s
 => => sha256:d62337a296a607f48a5dc3a7c07639360d98e0e78982ca8e459c832719012f12 4.09MB / 4.09MB                       1.3s
 => => sha256:d07bcf5901dfa793fa6b2c4c617e86bcef315b0b092cbfd1a929eefedaf8e3f2 43.17MB / 43.17MB                     6.5s
 => => sha256:403697a3e152c7a38ddb9175a90ed2dac97d780421c35949ff80cd67a7d4e596 48.02MB / 48.02MB                    11.0s
 => => sha256:0179cb8c17339f719c763581cefb9e377c016b4f7f4b747a665349e79470363c 202.23MB / 202.23MB                  19.8s
 => => sha256:6813afc532cf229aa943712c40a0ce8714262cb781df194ed2f2608d6b1d4152 4.19kB / 4.19kB                       6.9s
 => => extracting sha256:d07bcf5901dfa793fa6b2c4c617e86bcef315b0b092cbfd1a929eefedaf8e3f2                            1.3s
 => => sha256:8935ceca1893613650287584f8150ec7fcfcf5eaa7284b67689e49d3b5bed996 19.37MB / 19.37MB                    11.0s
 => => extracting sha256:403697a3e152c7a38ddb9175a90ed2dac97d780421c35949ff80cd67a7d4e596                            1.6s
 => => sha256:dfbca36bef871479ffabc72466e2913c954d5c4b9c2ca6896b701ca267f6a4a5 1.40MB / 1.40MB                      12.6s
 => => sha256:5195d0adf884f7d73694822f42887c1d0d079ca3a62fc18bcc7198e54431d45b 295B / 295B                          11.5s
 => => extracting sha256:0179cb8c17339f719c763581cefb9e377c016b4f7f4b747a665349e79470363c                            4.9s
 => => extracting sha256:8935ceca1893613650287584f8150ec7fcfcf5eaa7284b67689e49d3b5bed996                            0.7s
 => [internal] load build context                                                                                    0.1s
 => => transferring context: 34.12kB                                                                                 0.0s
 => [2/5] WORKDIR /app                                                                                               0.1s
 => [3/5] COPY package.json /app                                                                                     0.0s
 => [4/5] RUN npm install                                                                                            2.5s
 => [5/5] COPY . /app                                                                                                0.0s
 => exporting to image                                                                                               0.1s
 => => exporting layers                                                                                              0.1s
 => => writing image sha256:6c43a2767fb386a647a2e36d69d931c29f39b344a8b536974db009943b4bdf5d                         0.0s
 => => naming to docker.io/library/test_articol                                                                      0.0s

[+] Building 0.3s (10/10) FINISHED
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 36B                                                                                  0.0s
 => [internal] load .dockerignore                                                                                    0.0s
 => => transferring context: 2B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/node:8                                                            0.2s
 => [internal] load build context                                                                                    0.0s
 => => transferring context: 4.28kB                                                                                  0.0s
 => [1/5] FROM docker.io/library/node:8@sha256:a681bf74805b80d03eb21a6c0ef168a976108a287a74167ab593fc953aac34df      0.0s
 => CACHED [2/5] WORKDIR /app                                                                                        0.0s
 => CACHED [3/5] COPY package.json /app                                                                              0.0s
 => CACHED [4/5] RUN npm install                                                                                     0.0s
 => CACHED [5/5] COPY . /app                                                                                         0.0s
 => exporting to image                                                                                               0.0s
 => => exporting layers                                                                                              0.0s
 => => writing image sha256:6c43a2767fb386a647a2e36d69d931c29f39b344a8b536974db009943b4bdf5d                         0.0s
 => => naming to docker.io/library/test_articol
```

Fig. 9. A dataset of Docker-cached layers.



A schematic representation of the suggested solution is shown in Figure 10.



```
# Install/upgrade a Helm chart
  helm upgrade --install \
    "$helm_release_name" \
    "$chart_yml_dir" \
    --namespace="$k8s_namespace" \
    --create-namespace \
    --values "$values_file" \
    --set image.tag="$tag" \
    --set annotations."app\.gitlab\.com/app"="$CI_PROJECT_PATH_SLUG" \
    --set annotations."app\.gitlab\.com/env"="$CI_ENVIRONMENT_SLUG" \
    ${default_args} \
    ${custom_args} &&
    deploy_status="upgrade_success" || deploy_status="upgrade_fail"

  set +x # disable the mode of the shell where all executed commands are printed to the terminal

  debug "deploy status of $helm_release_name: $deploy_status"

  if [[ "$deploy_status" == "upgrade_fail" ]]; then
    fatal "deployment of $helm_release_name has failed"
  fi

  if [[ ! -z "$CNX_DEPLOY_DRY_RUN_ENABLED" ]] ;
  then
    info "dry-run deployment of $helm_release_name has been successful"
  else
    info "deployment of $helm_release_name has been successful"
  fi

  # Show the Helm release after install/upgrade
  list_release $helm_release_name
}
```

The update or installation of Helm, seen in Figure 11.

2.11 Experimental Findings and Analysis of the Effects of the Application

Impact Study on an Application

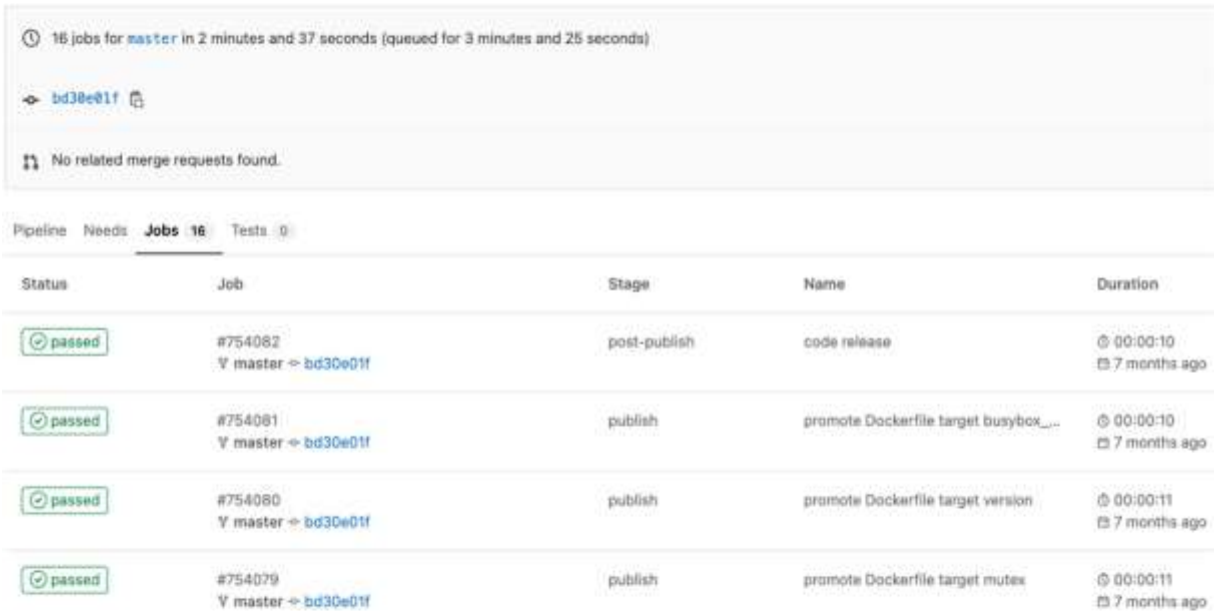
Second, each pipeline is distinct and basic, with no unnecessary or extraneous components, ensuring that code duplication is kept to a minimum across repositories.

The overall quality of information systems is enhanced via management transformation [36]. Operations and maintenance staff now have higher levels of professional competence. The change of O&M management, driven by technological advancements.

3.0 OUTCOMES OF EXPERIMENTS

The abstract objects in GitLab known as "runners" carry out the action of running a pipeline task. The first inference that can be drawn from this result is that the runners are SPOT instances, which are reusable and operate on AWS's spare capacity and hence cost far less than on-demand instances. About 80% of the calculated expenditures for each activated pipeline are avoided thanks to this suggested solution feature.It is also important to highlight the benefit brought about by the suggested solution, which is connected to the construction stage of the pipeline. By reusing existing layers wherever feasible, Docker-cache-layer integration speeds up the build process and reduces the amount of data sent. When a complete build phase is started again, the cached layers will be used instead of starting from scratch. This whole procedure lengthens the pipeline, cutting expenses while adjusting the effect on production.

Figure 12 displays the timespan of tasks from the sample pipeline shown in Figures 6 and 7. It demonstrates that 16 tasks (six Docker builds, six Docker promotes, and four versions) take a total of 157 seconds to complete throughout the full pipeline.



16 jobs for master in 2 minutes and 37 seconds (queued for 3 minutes and 25 seconds)

bd30e01f

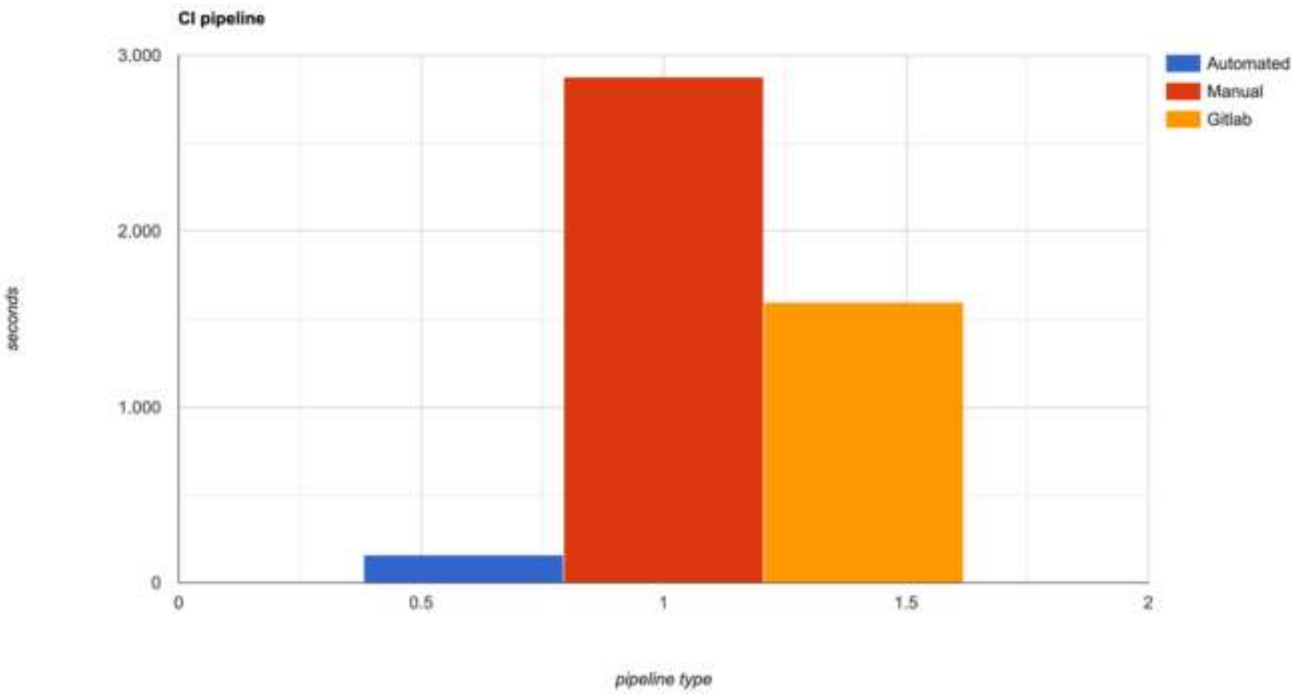
No related merge requests found.

Pipeline Needs **Jobs 16** Tests 0

Status	Job	Stage	Name	Duration
passed	#754082 master bd30e01f	post-publish	code release	00:00:10 7 months ago
passed	#754081 master bd30e01f	publish	promote Dockerfile target busybox_...	00:00:10 7 months ago
passed	#754080 master bd30e01f	publish	promote Dockerfile target version	00:00:11 7 months ago
passed	#754079 master bd30e01f	publish	promote Dockerfile target mutex	00:00:11 7 months ago

Time and steps in the pipeline diagram, seen in Figure 12.

Without the suggested solution's integration, manually triggering the tasks likewise took roughly 180 seconds. Time differences between the three pipeline options are seen in Figure 13.



Number of seconds depicting time disparity across pipeline types, seen in Figure 13.

The operating expenses of the pipelines are the focus of the brief discussion that follows as a direct consequence of the comparison. Collectively, this may help reduce CI/CD infrastructure expenses for businesses by as much as 80-85 percent compared to using a manual pipeline. When compared to the suggested method, the savings percentage for the third kind of pipeline—which can also perform processes in parallel but not on spot instances—is substantially lower, at roughly 10-15%.

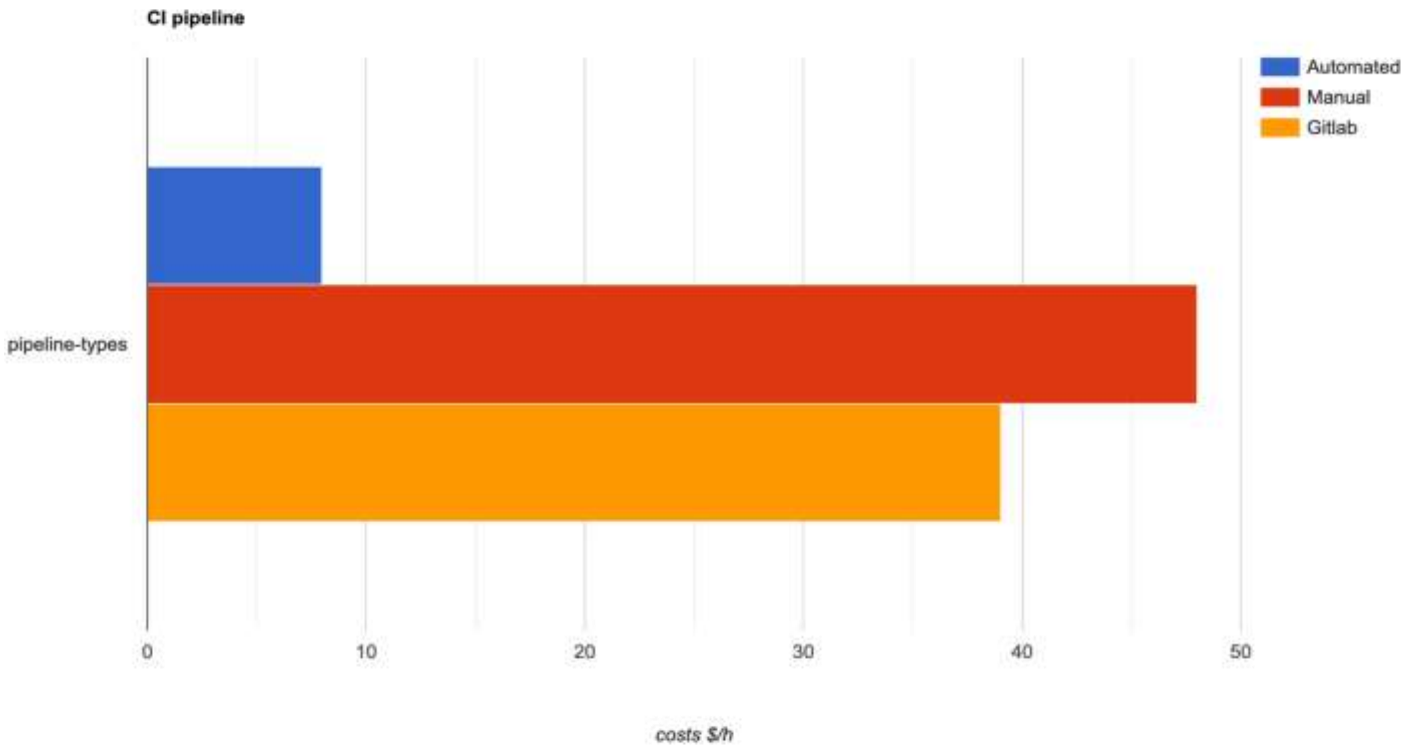
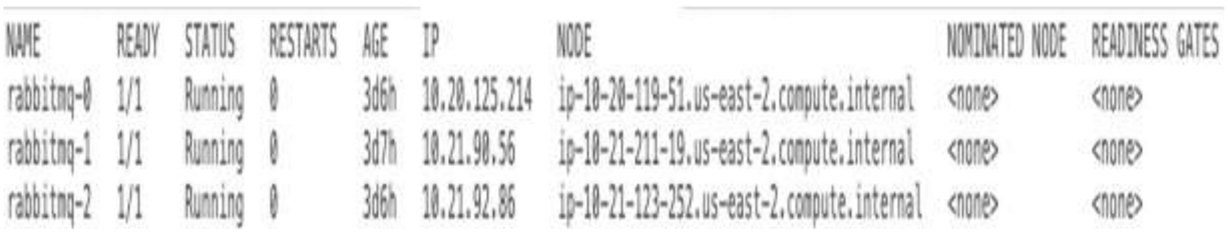


Figure 14 shows price tag on constructing a pipeline.

The benefit of this approach is realized in full with the deployment phase, which is realized through Helm. By acting as a wrapper, this reduces the overhead of maintaining configuration and manifest files for microservices, hence shortening the time of the automated workflow. The deployment process is risk-free, secure, and quick because of these factors.The build phase of the continuous integration pipeline is described in the preceding section, and the outcomes of the whole flow of this suggested solution are shown in the section that follows.The benefits of this technique are shown in these tests because of the hardware implementation it demands. Cloud instances (EC2) hosted on Amazon Web Services' Elastic Kubernetes Services platform hosted the complete system. Once everything was integrated, we kicked off the whole pipeline of our designed solution to roll up a RabbitMQ Helm Chart with one master pod and two slave pods.

A cluster of RabbitMQ that is both reliable and extensible. Each pod is launched on its own EC2 instance in Figure 15 to reduce the likelihood of service disruptions.



```
NAME        READY  STATUS   RESTARTS  AGE   IP             NODE                                          NOMINATED NODE   READINESS GATES
rabbitmq-0  1/1    Running  0         3d6h  10.20.125.214  ip-10-20-119-51.us-east-2.compute.internal    <none>           <none>
rabbitmq-1  1/1    Running  0         3d7h  10.21.90.56    ip-10-21-211-19.us-east-2.compute.internal    <none>           <none>
rabbitmq-2  1/1    Running  0         3d6h  10.21.92.86    ip-10-21-123-252.us-east-2.compute.internal   <none>           <none>
```

Figure 15 shows the suggested solution's deployment of RabbitMQ

4.0 CONCLUSION

The time it takes to create new software may be cut in half, and its quality can be increased, with the aid of continuous integration solutions. The machine takes care of the double integration, freeing up the developers to concentrate on the software's architecture. Testers may be put through their paces with the help of continuous, fast feedback. The continuous integration delivery system is a major step forward in the development of fully automated systems. It aids in the creation of higher-quality software systems and the implementation of lean processes for continuous improvement, as well as in the maturation of software projects and the enhancement of software service levels. The system is built on Agile principles, which are in charge of the features of apps being automatically integrated, tested, and delivered.

4.1 Limitations, Discussion, besides Future Research

There was a dramatic shift in how organizations and other entities released and improved their goods over the internet after adopting the practices of continuous integration and continuous deployment [31]. These integrations can be constructed and distributed quickly, without downtime or bottlenecks, thanks to the capabilities of cloud platforms [39], making this a tool with minimal drawbacks.The first possible enhancement would be to rewrite the Bash scripts that comprise this solution into Golang [41], making it cross-platform [42]. Golang would also reduce development and compilation times [43]. The ability to develop and release software on many platforms (Windows, Arm, etc.) is another goal for the future. The suggested approach identified potential issues that might arise as a result of CI/CD/CDD practices, which presents some opportunities for further study.

References

1. Fogelstrom, N.D.; Gorschek, T.; Svahnberg, M.; Olsson, P. The impact of agile principles on market-driven software product development. J. Softw. Maint. Evol. Res. Pract. 2010, 22, 53–80. [CrossRef]
2. Weaveworks. Building Continuous Delivery Pipelines. Available online: https://www.weave.works/assets/images/blta80840 30436bce24/CICD_eBook_Web.pdf (accessed on 10 January 2022).
3. Moreira, M. The Agile Enterprise: Building and Running Agile Organizations, 1st ed.; Apress: Berkeley, CA, USA, 2017.
    a. Singh, S.; Sharma, R.M. Handbook of Research on the IoT, Cloud Computing, and Wireless Network Optimization (Advances in Wireless Technologies and Telecommunication), 1st ed.; IGI Global Hershey: Pennsylvania, PA, USA, 2019.
4. Losana, P.; Castro, J.W.; Ferre, X.; Villalba-Mora, E.; Acuña, S.T. A Systematic Mapping Study on Integration Proposals of the Personas Technique in Agile Methodologies. Sensors 2021, 21, 6298. [CrossRef] [PubMed]
5. Fitzgerald, B.; Stol, K.-J. Continuous Software Engineering: A Roadmap and Agenda. J. Syst. Softw. 2017, 123, 176–189. [CrossRef]
6. Awscloud. A Roadmap to Continuous Delivery Pipeline Maturity. Available online: https://pages.awscloud.com/rs/112-TZM-
7. Liu, D.; Zhao, L. The Research and Implementation of Cloud Computing Platform Based on Docker. In Proceedings of the 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing (ICCWAMTIP), Chengdu, China, 19–21 December 2014.
    a. Xia, C.; Zhang, Y.; Wang, L.; Coleman, S.; Liu, Y. Microservice-based cloud robotics system for intelligent space. Robot. Auton. Syst. 2018, 110, 139–150. [CrossRef]

8. Dr.Naveen Prasadula A Review of Literature on Pipeline-Generated Continuous Integration And Deployment Method For Agile Software Development
9. Górski, T. Towards Continuous Deployment for Blockchain. Appl. Sci. 2021, 11, 11745. [CrossRef]
10. Chacon, S.; Straub, B. Pro Git, 2nd ed.; Apress: Berkeley, CA, USA, 2014.
11. Zolkifli, N.N.; Ngah, A.; Deraman, A. Version Control System: A Review. Procedia Comput. Sci. 2018, 135, 408–415. [CrossRef]
12. Burns, B.; Grant, B.; Oppenheimer, D.; Brewer, E.; Wilkes, J. Borg, omega and kubernetes. Commun. ACM 2016, 14, 70–93.
13. Gitlab Inc. Build with Kaniko. Available online: https://docs.gitlab.com/ee/ci/docker/using_kaniko.html (accessed on 12 March 2022).
14. Jamal, M.; Joel, C. A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework. In Proceedings of the 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 27–30 January 2021.
15. Reis, D.; Piedade, B.; Correia, F.F.; Dias, J.P.; Aguiar, A. Developing Docker and Docker-Compose Specifications: A Developers' Survey. IEEE Access 2022, 10, 2318–2329. [CrossRef]
16. Bhimani, J.; Yang, Z.; Mi, N.; Yang, J.; Xu, Q.; Awasthi, M.; Pandurangan, R.; Balakrishnan, V. Docker Container Scheduler for I/O Intensive Applications Running on NVMe SSDs. IEEE Trans. Multi-Scale Comput. Syst. 2018, 4, 313–326. [CrossRef]
17. Packard, M.; Stubbs, J.; Drake, J.; Garcia, C. Real-World, Self-Hosted Kubernetes Experience. In Proceedings of the Practice and Experience in Advanced Research Computing (PEARC 2021), Boston, MA, USA, 18–22 July 2021.
18. Dr.Naveen Prasadula A Review of Literature on Pipeline-Generated Continuous Integration And Deployment Method For Agile Software Development
19. Karamitsos, I.; Albarhami, S.; Apostolopoulos, C. Applying DevOps Practices of Continuous Automation for Machine Learning.
20. Information 2020, 11, 363. [CrossRef]
21. Zhou, Y.; Ou, Z.; Li, J. Automated Deployment of Continuous Integration Based on Jenkins. Comput. Digit. Eng. 2016, 44, 267–270.
    a. Buchanan, S.; Rangama, J.; Bellavance, N. Helm Charts for Azure Kubernetes Service. In Introducing Azure Kubernetes Service; Apress: Berkeley, CA, USA, 2019.
    b. Fedak, V. What is Helm and Why You Should Love It? 2018. Available online: https://hackernoon.com/what-is-helm-and-why- you-should-love-it-74bf3d0aafc (accessed on 14 March 2022).