An Approach to basic GUI-enabled CI/CD pipeline with Static Analysis tool

Kiran Kumar HK[1], K Subrahmanya[2], Kavana R[3] and Shwetha Baliga[4]

1,2,3 *Student, RV College Of Engineering,Bangalore, India*

4 *Assistant Professor, RV College Of Engineering.Bangalore, India*

1 kkirankumar.ec17@rvce.edu.in , 2 ksubrahmanya.ec17@rvce.edu.in ,

3 kavanar.ec17@rvce.edu.in *and* 4 shwethaprabhun@rvce.edu.in

Abstract: *Automation in software delivery process is considered best practice in secure Software development life cycle(sSDLC) and DevOps Deployment of software occurs multiple times in a week, day or within a span of few minutes. Manual deployment of the code and database which comprises the desired software is not only tedious but also prone to errors. The Continuous Integration and Continuous Deployment (CI/CD) pipeline ensures that the software delivery is done in an efficient and reliable way so that the software is available for use at any instant of time. In this paper, we discuss a basic approach towards development of customised CI/CD pipeline with static analysis tool (SAT) integration providing greater reliability to our architecture. SAT is one of the major component of sSDLC which checks the codebase for static errors that helps in identifying potential bugs and vulnerabilities. This approach is vital for smaller teams in industries having less bandwidth or in long term academic projects. We discuss the development of a customised CI/CD pipeline in detail. Finally, a ReactJS based GUI is designed to obtain the pipeline status and SAT results.*

Keywords: CI/CD Pipeline, Python, Tmux, Static Analysis Tool, Code analysis, vulnerability analysis, golang, ReactJS

1. Introduction

In a software development life cycle, secure and efficient software delivery to the end-device or end-user makes the development meaningful. The software delivery in industries or in institutions is achieved manually, but is increasingly being replaced by DevOps essential, Continuous Integration and Continuous Deployment (CI/CD) pipeline since the manual deployment is prone to errors and time consuming to catch up with the pace of software development and enhancement. Today, industries are using numerous open source as well as commercial deployment pipelines, but teams in some industries rely on manual deployment due to the small size of the project or lack of bandwidth and budget. Development of basic CI/CD pipeline from scratch provides a way of customisation for specific projects in industries, saving team bandwidth and fetching immense knowledge in academic activities. Integration of the Static Analysis tool in this pipeline adds a way to debug the software without actually running it. Coding and compliance requirements must be met and making mistakes isn’t an option in industries. Static analysis checks for various issues in code and assures the code to be in compliance with the industry standards by detecting early bugs, vulnerabilities and security flaws. The Static analysis is done by analysing a subset of code against a vast set of coding rules. The integration improves the efficiency of continuous integration pipeline in specific. The sections in this paper discuss the development of deployment-centric

CI/CD pipeline for website delivery. The pipeline is equipped with the graphical user interface (GUI) depicting the status of the pipeline blocks and static analysis results.

2. Literature Survey

Automated unit-tests are very difficult to implement for dynamically typed programming languages. Statically typed programming languages like Java have mature test production tools. However, due to lack of knowledge on the type of language and the dynamic character of language, it is considerably harder to automatically produce supporting tests for dynamically typed programmes like Python. This paper [1] explains the testing issue with the dynamically typed languages. The competence of the organisation to provide high speed services and applications includes competitiveness on the market effectively. These management processes require quick and effective management practises and resources. When developing applications in the cloud changes must begin at software engineering level, so our DevOps processes must be automatic by means of cloud- and non-cloud DevOps automation tools. The objective of this paper is to transfer DevOps to Cloud and become flexible when developing and running applications. At the same time, it is the research's main approach to consider how to apply this DevOps framework and automation to public and/or private clouds.This article examines the rise of DevOps as a dramatic shift in the IT environment to enhance the processes. The objective is to understand how DevOps and Cloud work together to help companies achieve their transformation objectives.[3] The productivity of projects has improved in agile practises with the approach of continuous integration and continuous delivery or continuous deployment (CI/CD).Development benchmarking is an important activity since the living system is interrupted. A load test to calculate planned device status should be carried out in the new version. Due to simulated traffic conditions, the conventional method of load testing cannot determine the efficiency of the output. This approach [4] has expanded the CI/CD Pipeline for three phases of automation called bench-marking, load testing and scaling to solve these problems. It minimises the system interference using a test bench method when bench-marking the system and uses load monitoring with production traffic, which results in more exact results. The device scaling can be assessed once the bench-marking and load test phases are finished. The pipeline was initially built with the Ansible automation server Jenkins CI and Git repository. As the microservices architecture is being incorporated widely everywhere. Monolithic architecture(MA) performs poorly in the scalability of a system. There is a lot of overhead generated which in turn reduces throughput drastically. Hence, there is a need for the migration of old Monolithic architecture to microservices architecture for large scalable systems. This paper is the definition of a strategy that would be capable of supporting migration from Monolithic Architecture to a Microservices Architecture. The strategy solely aims to be applied on monolithic systems, assisting their evolution into a microservices architecture system. This migration strategy helps the newly created systems to leverage the benefits offered by microservices architecture, for example, scalability and maintainability of the systems. The existing software based on monolithic architecture can use this strategy to migrate into new and more flexible microservices-based systems, evolving into a more powerful system. Microservices architecture(MSA) has improved scalability, maintainability and evolvability.[13] Describes an efficient method of transforming from MA to MSA. It provides a five-step process to break a MA to assign the split to relevant business units starting with function analysis and then the business unit identification, its analysis and assigning every business units to create a final microservices architecture from the inputs from each module. The strategy provided requires a lot of refinements to be into a final production stage. [13]

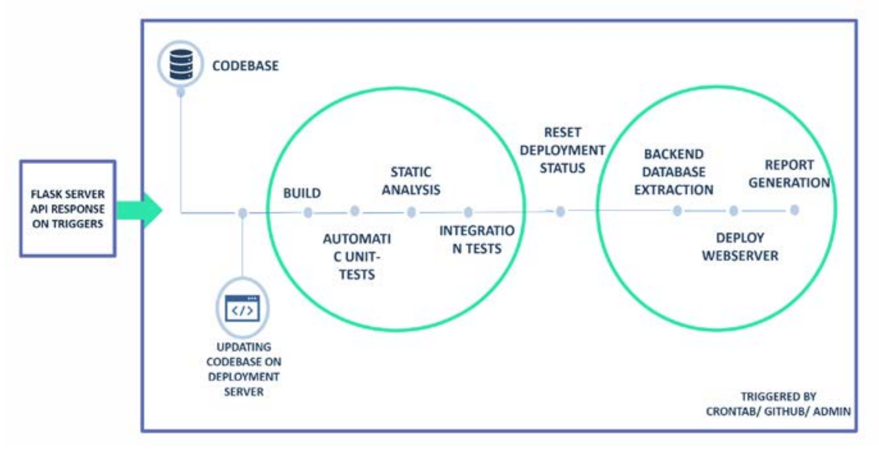
3. Development of CI/CD pipeline

The development of the CI/CD pipeline begins with the acquisition of triggers or events which automatically execute the pipeline blocks.. The basic foundation that hosts the pipeline stages and End-points or otherwise called Application Programming Interfaces (APIs), which automatically triggers the pipeline script, is built using the Python Flask application. The block diagram for the basic CI/CD pipeline is shown in Figure 1.1.The setting up of the flask application for the pipeline execution, and development of various stages for the deployment are discussed in following sections.

3.1. Setting up Flask Application

The flask application sets up the foundation for the pipeline server which is deployed in debug mode to keep track of the application correctness. The events which are designed to trigger the pipeline execution are:

1. WebHooks: Event providing information or notification of the code commit into Version control tool like GitHub,
2. Time Triggered database generator script: Event following the automatic update of the existing database. The time trigger is achieved through CronTab where the execution of the script is scheduled in a virtual environment. The database is copied automatically before the trigger through Secure Copy and Secure File Transfer Protocol. The cURL POST notifying the availability of a new database is named with the database version label for easier backup.
3. Administrator CLI: Command Line tool providing basic external control on the pipeline. The development of this Admin Command line is discussed in section 4.5.
4. Static analysis tool : The Static analysis is invoked after unit-tests in the pipeline. The tool after completion responds with the code analysis report. The response is


CODEBASE
FLASK SERVER API RESPONSE ON TRIGGERS
BUILD
STATIC ANALYSIS
AUTOMATI C UNIT-TESTS
INTEGRATIO N TESTS
RESET DEPLOYMENT STATUS
BACKEND DATABASE EXTRACTION
REPORT GENERATION
DEPLOY WEBSERVER
UPDATING CODEBASE ON DEPLOYMENT SERVER
TRIGGERED BY CRONTAB/ GITHUB/ ADMIN

analysed for errors or flaws and the pipeline is moved to the next block.

Figure 2: CI/CD Pipeline Block Diagram

The flask application analyses the triggers received and invokes the pipeline blocks according to the trigger type. For a code commit event and new database push, the pipeline is completely executed covering all the blocks. The Static analysis event runs irrespective of the other trigger type as an integral part of the pipeline. Specific blocks of pipeline can be executed using the Admin Command Line. The pipeline blocks are implemented as functions in the flask application.

3.2. Updating Code-base

The first step in the software deployment is obtaining the latest version of the software in the deployment machine. The source code for any software is stored in the version control tools like GitHub, which is used for implementation of this paper. The deployment script on receiving the trigger navigates to the local repository where the remote repository is cloned. The cloned repository includes the necessary source code required for the deployment of the software to the end-device/user. It is very important to note that code changes are not done in the deployment machine in general and in the deployment repository in specific. This leads to the abrupt change in 'upstream server' and local changes might perform differently from what was designed. The Steps involved in Codebase updation include:

1. Accessing Local Remote Repository : The deployment script navigates to the local repository located on the deployment machine to check for the repository state as a first step.
2. Saving Local Repository State : There are events which can modify the local changes to deploy the software reliably to the correct destination. These changes are saved before updating the codebase to avoid source code mismatch, which might lead to unintended results once the software is deployed.
3. Updating and Merging Local Repository : The codebase is updated based on the trigger received after the code commit. The challenge in automatically updating the local repository is avoiding the security prompt when extracting the updated git repository. The security prompt is overcome with the use of GitHub Personal access Token (PAT) and having a remote url for the repository set for the deployment machine.

3.3. Continuous Integration Pipeline

Continuous integration (CI) is a software development process in which every time a team member changes, small modifications are tested to the underlying code in the application. CI seeks to speed up the release process by allowing teams to detect and resolve defects early during the development cycle and to promote stronger communication between the developers.

1. Build : It is basically designed to create a functional software instance that can potentially be sent to the end user. If the build stage is not passed, the configuration of the project would be crucial. The build phase in this paper is used to install the unavailable dependencies for the software that is being deployed. The requirements are provided as a .txt file by the developer in the code-base which is then to install the packages specified in the file. After the dependencies are installed, numerous modules of the software are packaged into a single executable instance using bash script. This software package is invoked into the pipeline.
2. Unit-Testing : Once the code-base is merged in the local machine, the modules that make the software together need to be tested for error and flaws before they are deployed to the end-user. The unit-testing in this paper is done using python-

inbuilt unit-testing library.The Unit-testing begins with the import of the modules that need to be tested. Uni-testing is done separately for the software modules with the functions that implement the functionality for software are tested with independent Test Cases. The modules are automatically tested in the pipeline using *Discover*. A complete coverage report is obtained from the unit-testing. The coverage threshold is set to 85%. If the threshold is not met, the pipeline declines the updated software and the previous version is deployed.

3. Static Analysis : This stage basically adds the static checks to the code. Doing a static analysis helps in finding out the early bugs and vulnerabilities and also keeps the code in compliance with the coding standards and uniform style formatting. This stage is a must in making CI/CD secure. More about static analysis tool is explained in further sections.
4. Integration Tests : The integration tests are performed on the complete source code modules to check the functionality is as intended when the modules are run together. The integration testing in this paper is essentially communicating the unit-testing test cases among different modules. The test suite is extended to perform integration testing using python context managers.

The Continuous integration pipeline developed in this paper adds an extra element of static analysis over the conventional pipeline. The unit-tests are automated providing both test and coverage results. Since the pipeline design is deployment centric, the continuous integration is limited to deployable modules in the source directory. Once the continuous integration phase is passed successfully, the pipeline slides to the continuous deployment block.

3.4. Continuous Deployment Pipeline

The pipeline proposed in this paper is implemented in a webpage. Therefore, its deployment after integration and testing is prepared for deployment in essentially three phases:

1. Database acquisition and sanitisation :The database, generated on different machines, is copied to the deployment machine using secure copy and secure file transfer protocol with database name as “data_Label.db”. The flask server receives the database along with its label from the database generator through cURL. The database label received from API is searched in the databases directory where the database was copied from the database generator. The database label which matches with the flask request is then verified for its correctness among database tables. If database sanity is verified, it is deployed to the end device through the back-end server.
2. Backend Server Deployment:The back-end server holds all the data that is fetched by the webpage. The back-end server obtains the data from the database based on the requests from the user.Since the requirements required for the back-end needs to be tested again along with various other command line functions like spawning a virtual environment, navigating through directories, a bash script is developed for the backed server deployment, which is then invoked into the pipeline using python *subprocess.call()* command.
3. Web Server Deployment : Once the back-end server is running the APIs are open to be accessed by the web-server. So, the web-server is deployed after the back-end to prevent the delays between the UI loading and data fetching.The web-server is deployed using *yarn build* which ensures again that there are no dependencies which are left to be installed that would essentially lead to failure of the deployment.Once the web-server is successfully built ,*yarn serve* serves the

web page or dashboard to the end-user which can be accessed anywhere from the internet. The upstream server is set to the IP that is specified during development.

4. Report Generation : A very vital step in any CI/CD pipeline is report generation and notification facility. Since the complete process is automated, one is not in light about the process that is being carried out by the automated pipeline on requests received. The report contains vital processes including the Code-base updation status, Unit-test results along with coverage, Static analysis results and software deployment status. The report also includes the snapshot of the deployed dashboard to ensure that the dashboard is displayed as intended. The path to the log file and other navigation information is also included.The report is mailed on selecting the option in the CI/CD pipeline dashboard. The Python *smtplib* package is used to send an email notification to the dashboard administrator after every deployment is completed by the pipeline. The UI equipped with this CI/CD pipeline also provides an option to directly download the deployment report.

In the proposed CI/CD pipeline, every process is logged into a customized log file which displays log information in a color coded manner for better understanding of the process. The log file is arranged in banners for a particular block of the CI/CD pipeline, so that the developer or administrator can easily navigate to the block, in case of failure or for process details. Every step in the CI/CD pipeline development discussed in this section is designed with exception handling to ensure that at no point of time, the server is frozen or terminated due to any technical or logical errors. In case of error or any haphazard behaviour of the pipeline the administrator or developer is instantly notified

3.5. Command Line for pipeline control

The continuous integration and continuous deployment pipeline is automated and is running at every instant of time listening for triggers. In such cases, there is a high probability that the functionality deviation of the pipeline or the tools invoked in the pipeline are deployed and are unnoticed. An unified Command-line interface is developed that provides multiple options to the administrator to deploy, freeze, restart or gracefully shutdown the server in case of critical failures, which was notified from the report generation step in section 4.2. On initializing the admin Command line Interface, the following options are provided for the user to control the CI/CD pipeline externally. The flask APIs process the CLI options to perform the intended action including running a specific block of pipeline.

1. Complete Re-Deployment : The complete CI/CD pipeline is executed starting from updation of codebase. This option is mainly used in this paper to re-deploy the backup databases when the deployed database is not correct or there is a redundancy in the deployed database.
2. Re-deploying Continuous Integration Block :This is an important option to be present to externally test the code-base to ensure that there are no changes made to the deployed code-base after deployment. The developer can also use this option to test the code while debugging the issue. With this option the CI/CD pipeline can be explicitly used as a Build and Test tool to evaluate the software before deploying either through the pipeline proposed in this paper or through external CI/CD tools.
3. Re-deploying Continuous Deployment Block :This option deploys the back-end server and the web-server without executing the continuous Integration block. So, using this option the admin can easily revert the local changes and externally

re-deploy the back-end server and web-server without having to wait for the pipeline to receive the trigger.

4. Complete pipeline shutdown :This option is only used under critical failures in deploying the software. On complete shutdown, the pipeline is 'gracefully terminated' by releasing the ports that were used by the servers, the virtual environment is deactivated and cleared for residue code cache.

One important and handy feature of this admin CLI is that the script can be run like an application on any device that is connected to the intranet. This helps the administrator and the developers to control and manage the pipeline at their convenience remotely without having to access the deployment server whenever there is a need for re-deployment or complete shutdown or just running the Build and Test. It is important to note that the report is generated and the process is logged for every action taken and is mailed to the person invoking the pipeline using Command line options given.

3.6. Comparison with available CI/CD tools

The CI/CD pipeline discussed in this paper provides a basic software delivery mechanism which is designed specifically as a deployment centric CI/CD pipeline. The CI/CD tools available today provide better features than the pipeline being proposed. The tools are to be enabled through payment for obtaining complete access. This is less adaptable for small project size and smaller teams in industries. The proposed pipeline, though raw, provides comparable features and improved functionality when compared to available tools with well-equipped GUI.

In general, CI/CD tools available today comprise basic pipeline blocks like updating codebase, Build, tests and deployment. The proposed pipeline adds a static analysis functionality to the pipeline which enables the code to be verified even before testing for bugs and vulnerabilities. This added feature not only detects code flaws but also checks for code quality.

Another Important highlight proposed is that the database sanitization and deployment can also be included in the pipeline by just specifying database path in the configuration, similar to specifying path for deployment repository. The deployment report is smartly mailed to the admin in case of normal deployment with triggers and is mailed to the operating person when triggered from the admin Command Line Interface. The Admin Command Line is another added feature which enables developer or operator to Start, stop, freeze or even run specific blocks of the pipeline remotely from anywhere on the internet.

The Important aspect of the proposed pipeline is that it is completely integrated with Terminal Multiplexer, also called TMux. This helps the pipeline to work in a systematic manner where each block is executed as a separate window in the TMux. This provides greater stability and debugging capability to the pipeline. The process information can be easily viewed from the TMux console for each block. Using TMux provides an edge over software disaster management where the software can be terminated with just one command that closes the TMux session.

The limitations of the proposed tool with available CI/CD tools is that the source code supported is limited to Python, Bash, JS (ReactJS) and GoLang, whereas the tools available today are compatible with numerous coding languages. The proposed tool comes with overhead of setting up the configuration and TMux session for the first run which also enables it to be a completely customizable pipeline.

4. Static Analysis Tool in CI/CD

Static Analysis can be described as one of the methods of debugging where a SAT automatically examines source code without actually running the program. It checks for various issues in code and assures the code to be in compliance with the industry standards. It helps the development team to produce error free code. On-time delivery of high-quality releases are must and should for the product development team. Coding and compliance requirements must be met and making mistakes isn't an option. Static Analysis speeds up the process of identifying early bugs and security flaws. Static Code Analysis and Static Analysis are used interchangeably; they are also known as source code analysis. Static analysis is done by analysing a subset of code against a vast set of coding rules.This type of analysis looks for flaws in source code that could lead to vulnerabilities. This can, of course, be accomplished through manual code reviews. However, using automated tools is far more efficient. Static analysis is frequently used to ensure that coding guidelines, such as MISRA, are followed. It's also frequently used to meet industry standards, such as ISO 26262.

The static analysis tools can be implemented in CI/CD pipeline in broadly three ways. One is by installing it while running the CI/CD pipeline, this method can be followed if the static analysis tool is of small size. Golangci-lint is one of the small footprint static analysis tool which can be installed during the CI/CD run and made to lint the code and show the errors in a file. The Second method of incorporating static analysis tool is by using a seperate software outside the CI/CD pipeline like SonarQube which is open source software that can lint code written in more than 20 programming languages. These external softwares can be made to provide a verdict on the current code base and then make the Static Analysis stage pass or fail accordingly. The third way is by aggregating many of the linters into a single program which can be built in the build stage of the CI/CD and then run in the static Analysis stage to provide a boolean output which decides to pass or fail the stage. If there are issues found in the code then the Static Analysis stage must fail and output a file containing the description of the errors found in the code, location of the error and optionally the name of the linter.

The CI/CD pipeline designed in this paper takes a boolean Input from the Static Analysis tool, if there are errors then a boolean False is returned and vice versa. The errors found are logged into a single file which is then redirected to a GUI for displaying the errors found and the description of the errors in a web tool.

5. Design of GUI

The GUI[graphical user interface] is designed and developed on React JS[JavaScript].React is an open-source front-end JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications.The GUI is developed here for the CI/CD tool where the home page consists of the pipeline regarding the tasks completed.There is also a progress bar showing the percentage of work completed. The Run button triggers the CI/CD pipeline and Schedule button is used to setup a Cron Job to run CI/CD at scheduled time. The Download and Send Email Button perform the respective operations on the report generated.The Show Static Errors button is used to show the errors,location of the errors ,description of the errors and the linter name.

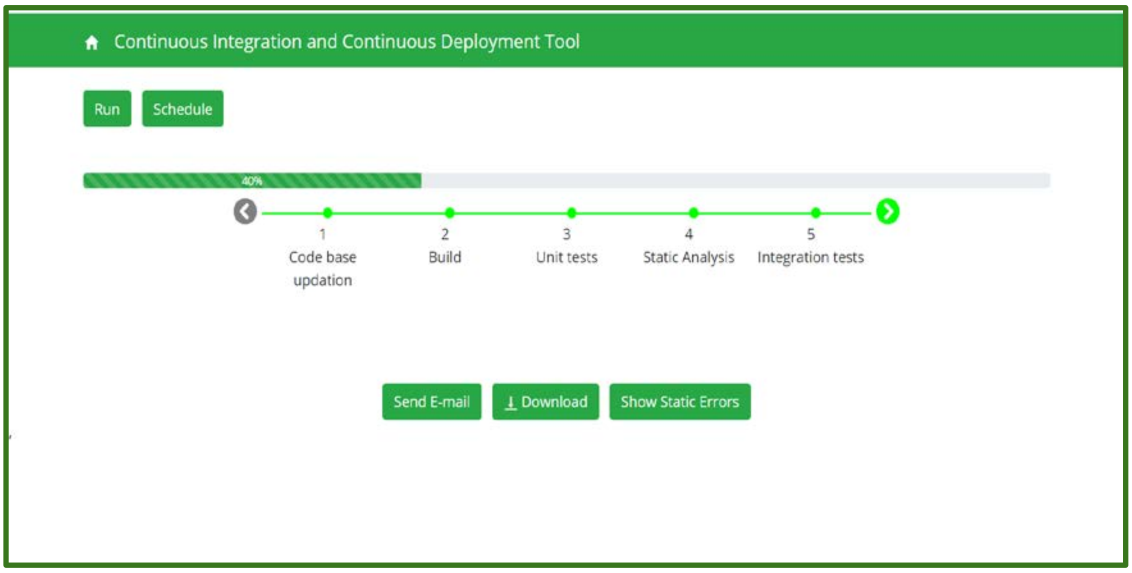


Figure 1 - CI/CD Tool GUI

6. Conclusion

The pipeline is built over a flask server that executes on receiving triggers from various sources like GitHub, static analysis tool, database collector script and from the administrator CLI.The pipeline built is completely customised to be deployment centric. It is built over Tmux which provides greater visibility and adaptability. The GUI shows the progress of the CI/CD pipeline and also has an option to display all the issues identified by the Static Analysis Tool.

7. Future Scope

The CI/CD automation can be extended to further level by reducing the memory footprint of the static analysis toolThe CI/CD pipeline which is now running on TMux Session can be executed as a Linux Service with complete Linux Daemon Life-cycle in place.The configuration of the pipeline is currently semi-automated which can be automated to completely prevent human intervention and provide seamless software delivery.As a

Next-Gen solution the database table created in the backend can be directly pushed to Apache Kafka Server, which can then be imported in Grafana Application.

REFERENCES

[1] I. Ristemi, M. A. Trpkovska, and B. Cico, “Mygitissues web application as a solu-tion in dealing with issues on github,” in2019 8th Mediterranean Conference onEmbedded Computing (MECO), 2019, pp. 1–4.doi:10.1109/MECO.2019.8760175.

[2] S. Lukasczyk, F. Kroiß, and G. Fraser, “Automated unit test generation for python,”inSearch-Based Software Engineering, A. Aleti and A. Panichella, Eds., Cham:Springer International Publishing, 2020, pp. 9–24,isbn: 978-3-030-59762-7.

.[3] P. Agrawal and N. Rawat, “Devops, a new approach to cloud development testing,”in2019 International Conference on Issues and Challenges in Intelligent ComputingTechniques (ICICT), vol. 1, 2019, pp. 1–4.doi:10.1109/ICICT46931.2019.8977662.

[4] S. Arachchi and I. Perera, “Continuous integration and continuous delivery pipelineautomation for agile software project management,” in2018 Moratuwa EngineeringResearch Conference (MERCon), 2018, pp. 156–161.doi:10.1109/MERCon.2018.8421965.

[5] M. Wurster, U. Breitenb¨ucher, M. Falkenthal, C. Krieger, F. Leymann, K. Saatkamp,and J. Soldani, “The essential deployment metamodel: A systematic review of de-ployment automation technologies,”SICS Software-Intensive Cyber-Physical Sys-tems, vol. 35, no. 1, pp. 63–75, Aug. 2020,issn: 2524-8529.doi:10.1007/s00450-019-00412-x. [Online]. Available:https://doi.org/10.1007/s00450-019-00412-x.

[6] P. Vogel, T. Klooster, V. Andrikopoulos, and M. Lungu, “A low-effort analyticsplatform for visualizing evolving flask-based python web services,” in2017 IEEEWorking Conference on Software Visualization (VISSOFT), 2017, pp. 109–113.doi:10.1109/VISSOFT.2017.13.

[7] S. Mysari and V. Bejgam, “Continuous integration and continuous deploymentpipeline automation using jenkins ansible,” in2020 International Conference onEmerging Trends in Information Technology and Engineering (ic-ETITE), 2020,pp. 1–4.doi:10.1109/ic-ETITE47903.2020.239.109.

[8] R. Potluri and K. Young, “Analytics automation for service orchestration,” in2020International Symposium on Networks, Computers and Communications (ISNCC),2020, pp. 1–4.doi:10.1109/ISNCC49221.2020.9297249.

[9] B. Zhao, Z. Li, and T. Zhang, “The heterogeneous deployment tool for hardware andsoftware co-design,” in2020 International Conference on Computer, Informationand Telecommunication Systems (CITS), 2020, pp. 1–5.doi:10.1109/CITS49457.2020.9232649.

[10] O. Javed, J. H. Dawes, M. Han, G. Franzoni, A. Pfeiffer, G. Reger, and W. Binder,“Perfci: A toolchain for automated performance testing during continuous integra-tion of python projects,” in2020 35th IEEE/ACM International Conference onAutomated Software Engineering (ASE), 2020, pp. 1344–1348..

[11] “Ieee approved draft standard for devops: Building reliable and secure systemsincluding application build, package and deployment,”IEEE P2675/D2, October2020, pp. 1–93, 2021.

[12] O. Al-Debagy and P. Martinek, “A comparative review of microservices and mono-lithic architectures,” in2018 IEEE 18th International Symposium on Computa-tional Intelligence and Informatics (CINTI), 2018, pp. 000 149–000 154.doi:10.1109/CINTI.2018.8928192.

[13] L. De Lauretis, “From monolithic architecture to microservices architecture,” in2019 IEEE International Symposium on Software Reliability Engineering Work-shops (ISSREW), 2019, pp. 93–96.doi:10.1109/ISSREW.2019.00050.

[14] K. Bakshi, “Microservices-based software architecture and approaches,” in2017IEEE Aerospace Conference, 2017, pp. 1–8.doi:10.1109/AERO.2017.7943959.

[15] A. R. Manu, J. K. Patel, S. Akhtar, V. K. Agrawal, and K. N. B. S. Murthy, “Dockercontainer security via heuristics-based multilateral security-conceptual and prag-matic study,” in2016 International Conference on Circuit, Power and ComputingTechnologies (ICCPCT), 2016, pp. 1–14.doi:10.1109/ICCPCT.2016.7530217.[16] A. U. Rehman, A. Nawaz, M. T. Ali, and

[16] M. Abbas, “A comparative study of agilemethods, testing challenges, solutions tool support,” in2020 14th InternationalConference on Open Source Systems and Technologies (ICOSST), 2020, pp. 1–5. doi:10.1109/ICOSST51357.2020.9332965.Department of Electronics and Communication Engineering, 2020-2021110.