A Study on Continuous Integration and Continuous Deployment (CI/CD) on Cloud

Katta Lokesh, Kunala Muni Krishna

[1]Student, School of Science and Computer Studies, CMR University, Karnataka, India [2] Student, School of Science and Computer Studies, CMR University, Karnataka, India

ABSTRACT

A methodology known as Continuous Integration/Continuous Delivery/Deployment (CI/CD) emphasizes the rapid release of small, incremental changes while utilizing automation throughout the development process. The foundation of the DevOps methodology is CI/CD, which is essential to its success. In order to provide constant input, stakeholders expect to be involved with the DevOps team on an ongoing basis. Combining the words "development" and "operations," DevOps aims to bring together the teams responsible for software development and operations, promoting teamwork throughout the software development lifecycle. Industry-standard techniques known as "continuous practices," which include "continuous integration, delivery, and deployment," enable businesses to reliably and consistently roll out new features and products. The passage does not go into detail, but it makes a hint about a comparison of deployment tactics.

Keyword: *Continuous Integration (CI), Continuous Deployment (CD), Cloud Computing, Cloud Infrastructure DevOps*

1. INTRODUCTION

Software engineering practices such as continuous deployment place a high priority on providing end users with software upgrades quickly. This procedure involves regularly deploying small software updates to production systems after they have undergone automated testing. Several well-known software businesses are using continuous deployment to release their products, including Rally Software, GitHub, Facebook, and Netflix. Companies that use continuous deployment have reported a number of benefits, such as lower development effort, better software quality, and increased customer happiness. Software engineering methodologies such as continuous delivery and deployment are intended to quickly provide software updates to end users. Humble and Farley [2] define continuous deployment as the process of automatically deploying software updates to end users after they have passed the necessary automated tests. Conversely, as mentioned software health attributes, reduced overhead costs, accelerated time to market, and adaptability to diverse application development stacks. In summary, CI/CD practices are instrumental in streamlining software development, enhancing quality, and expediting delivery, ultimately benefiting organizations across various industries.

2. LITERATURE SURVEY

[1]A thorough analysis of the problem presented by continuous deployment in the context of changing database schemas was carried out by M. De Jong and A. Van Duersen in 2015. Their investigation focused on the shortcomings of existing tools and solutions, especially with regard to foreign key handling and preventable downtime. They responded to these concerns by introducing a novel strategy and providing an open-source implementation. Initial evaluations show that their strategy is both effective and satisfactorily efficient

[2] The introduction of Rondo, a set of tools intended to support continuous deployment for dynamic, service-oriented applications, is the main topic of O. Günalp's (2015) work. These technologies are based on a deployment mechanism

that is idempotent and deterministic. Günalp and colleagues provide a deployment manager that can be used with Rondo. It can coordinate deployments and dynamically modify applications to conform to platform changes. This tool set also includes a domain-specific language designed specifically for defining deployment requirements. This strategy's efficacy is demonstrated by a number of projects that include ongoing reconfigurations, application installations, and platform provisioning

[3] In 2019, K. Gallaba and associates started a thesis project whose main goal was to improve the efficacy and robustness of CI/CD procedures. Their study consisted of two main parts: First, through carrying out empirical research with big datasets of CI/CD tool users, they contributed conceptually. Finding excellent practices and identifying common flaws in these techniques was the goal.Second, they contributed technologically by creating instruments meant to help stakeholders lessen shared constraints. These restrictions included difficulties like incorrect data interpretation and CI setting errors.The goal of the research project was to strengthen the reliability and effectiveness of CI/CD processes by offering insightful analysis and workable solutions

[4] In 2020, Q. Liao and associates did research with the goal of developing a model that could both abstract the core of the pipeline and provide a testing environment for evaluating the several elements impacting CI/CD performance. This research aimed to provide two important insights.First, using agent-based simulation tools, they set out to create a pipeline model based on queueing systems concepts.Their second goal was to create an experimental setup that would make it easier to assess various setups and operational situations.The main objective of this study was to offer a thorough model and a flexible testing environment in order to offer insightful information about CI/CD performance

[5] In the work of J. Mahboob and J. Coffman (2021), they outline a comprehensive CI/CD pipeline that operates seamlessly on Kubernetes, addressing four notable gaps found in existing implementations. These improvements encompass: Strong Separation of Duties: The pipeline effectively segregates the responsibilities of development, security, and operations (DevSecOps) roles, ensuring a robust separation of duties.

.

[6] Automation Emphasis: Significantly reducing the necessity for human intervention, the pipeline places a strong emphasis on automation. Resource Scope: Resources within the pipeline are meticulously scoped to a Kubernetes cluster, enhancing portability across diverse environments, including public cloud providers Artifact Security To bolster the security of deployment artifacts, the pipeline employs Asylo, a development framework designed for trusted execution environments (TEEs

3. PROPOSED METHOD

Adaptive Schema Evolution for Continuous Deployment (ASE-CD):

In order to reduce downtime and mitigate difficulties resulting from database schema changes in continuous deployment environments, the Adaptive Schema Evolution for Continuous Deployment (ASE-CD) method has been developed. To guarantee that schema updates are handled properly and don't interfere with service, ASE-CD uses a multi-phase schema migration approach and incorporates cutting-edge technologies.

.

3.1. Multi-phase Schema Migration:

In keeping with the idempotent and deterministic deployment procedures described in Günalp's Rondo framework [19], ASE-CD presents a multi-phase schema migration strategy. This strategy entails:

- **Preparation Phase:** Before any schema changes are made, a preparatory script is executed to identify potential conflicts, such as foreign key dependencies or data integrity concerns. The system automatically generates shadow tables and temporary columns to store data safely during the migration process.
- **Transition Phase:** Schema changes are incrementally applied to avoid downtime. During this phase, both old and new versions of the database schema coexist. The application is dynamically updated to read from and write to both schema versions, ensuring compatibility and data consistency.

- **Completion Phase:** Once the system confirms that all instances are using the new schema, legacy components are safely deprecated. The final step includes removing shadow tables and deprecated columns, ensuring a clean and optimized database structure.

3.2. Deployment Manager Integration:

In line with Günalp's methodology, ASE-CD integrates a Rondo-compatible deployment manager to automate the synchronisation of schema modifications among various services. The criteria and sequencing for schema changes are defined by this manager using a language unique to the domain. It ensures that the growth of the schema is carried out in a regulated way, avoiding contradictions and inconsistencies.

3.3. Real-Time Monitoring and Rollback Mechanism:

A real-time monitoring mechanism included in ASE-CD keeps track of database schema changes and their effects on application performance. An automated rollback process is initiated upon detection of any abnormalities or deterioration in service. With the rollback capabilities, there will be as little disturbance and downtime as possible as the database and application code will be returned to their original stable states.

3.4. Continuous Testing Environment:

Drawing inspiration from the research conducted by Liao [22] and Mahboob & Coffman [24], ASE-CD incorporates an agent-based simulation tool-based continuous testing environment. Before deploying schema changes to production, this environment enables developers to test their performance and dependability under a variety of deployment situations. Additionally, queueing system ideas are included in the testing environment to precisely simulate workload and operating situations in the actual world.

3.5. Security and Compliance:

Similar to the techniques outlined by Mahboob and Coffman [24], ASE-CD integrates trusted execution environments (TEEs) like Asylo to improve the security of deployment artefacts and guarantee compliance. This lowers the possibility of unauthorised access or data breaches by ensuring that sensitive processes, such schema modifications, are carried out in a safe and isolated environment.

Experimental Setup for CI/CD in Cloud:

1. Objectives:

- Assess CI/CD pipelines' performance and efficiency in a cloud environment.
- Track performance indicators including scalability, resource use, deployment speed, and system stability.
- Examine how various cloud setups and tools affect the performance of the CI/CD pipeline.

2. Hypotheses:

- Compared to conventional on-premises configurations, CI/CD pipelines implemented in cloud settings will exhibit improved scalability and resource efficiency..
- Human intervention and deployment times will be greatly reduced by automation and cloud-native tools.
- Performance and scalability of CI/CD pipelines won't be considerably impacted by security features.

3. Environment Setup:

a. Cloud Infrastructure:

- **Cloud Providers:** Utilise well-known cloud service providers like as AWS, Azure, and Google Cloud to gain insight into how the performance of CI/CD pipelines varies between cloud settings.
- **Regions:** Establish CI/CD pipelines in several geographical locations to evaluate deployment speed and latency..

b. CI/CD Tools:

- **Tools Selection:** Make use of cloud-native solutions like AWS CodePipeline, Azure Pipelines, and Google Cloud Build, as well as commonly used CI/CD technologies like Jenkins, GitLab CI, CircleCI.
- **Configuration:** Build several pipeline configurations that include integration tests, unit tests, static code analysis, deployment phases, and security scans.

c. Application Under Test:

• Create a sample application using microservices to mimic real-world situations.
• Use a variety of services to evaluate the CI/CD pipeline's orchestration and deployment capabilities.

• Package and deploy the application using containers (Docker), and use Kubernetes to manage these containers.

d. Database and Storage:

• To test database schema changes during deployment, use cloud databases such as Google Cloud SQL, Amazon RDS, and Azure SQL Database.
• Store logs and deployment artefacts in the cloud.

4. Experimentation Plan:

a. Baseline Measurement:

- • For on-premises CI/CD pipelines, measure baseline parameters like deployment time, resource utilisation, and system dependability.
- Create a control group with local servers or virtual machines to compare with cloud configurations..

b. Cloud Deployment Scenarios:

- **Scenario 1: Basic CI/CD Pipeline:** Establish a basic CI/CD pipeline with phases for code commit, build, and deploy.
- **Scenario 2: Full Pipeline with Testing:** Unit testing, integration testing, and security scanning should all be included in the pipeline.
- **Scenario 3: Multi-cloud Deployment:** To evaluate cross-cloud deployment and latency, deploy the CI/CD pipeline across various cloud providers and geographies.
- **Scenario 4: Auto-scaling Pipelines:** Test the auto-scaling capabilities of CI/CD pipelines to handle varying loads and the impact on deployment time and reliability.

c. Security Testing:

• Measure the effect of these security features on the pipeline's performance.
• Put in place security measures including encryption, role-based access control, and vulnerability screening.

5. Metrics and Data Collection:

a. Performance Metrics:

- **Deployment Time:** The duration between a successful deployment and the code commit.
- **Build Time:** how long it takes for a deployment to be successful before a code commit.
- **Test Execution Time:** Time required to execute tests within the pipeline.
- **Scalability:** Ability of the CI/CD pipeline to handle increased load (number of commits or builds).

b. Resource Utilization:

- **CPU and Memory Usage:** Monitor and record CPU and memory utilization of CI/CD tools during the deployment process.
- **Network Bandwidth:** Measure the network bandwidth usage for data transfer and communication.

c. Reliability Metrics:

- **Failure Rate:** Number of failed deployments or builds.
- **Downtime:** Amount of time the system is unavailable during deployment.

d. Security Metrics:

- **Vulnerability Detection:** Number and types of vulnerabilities detected during security scanning.
- **Compliance Checks:** Results from compliance checks integrated into the CI/CD pipeline.

6. Data Analysis:

• To see performance trends and analyse the gathered data, use data analytics tools.
• Examine the performance indicators for various cloud service providers and geographical areas.
• Assess how various CI/CD configurations affect the security, dependability, and speed of deployment.

7. Evaluation and Results:

• Examine the outcomes to determine whether CI/CD solutions and cloud configurations offer the greatest security, scalability, and performance.
• Locate possible bottlenecks and areas where CI/CD pipeline configurations need to be improved.
• Make suggestions on how businesses should improve their CI/CD pipelines in cloud settings.

8. Conclusion:

- • Provide an overview of the experimental setup's results.
- • Talk about how the findings can be applied to enhance cloud-based CI/CD procedures.
- •Summarise the directions for further research based on the results of the experiment..

Tools and Technologies:

- **CI/CD Tools:** Jenkins, GitLab CI, CircleCI, AWS CodePipeline, Azure Pipelines, Google Cloud Build.
- **Containerization:** Docker, Kubernetes.
- **Cloud Providers:** AWS, Azure, Google Cloud.
- **Monitoring and Logging:** Prometheus, Grafana, ELK Stack.
- **Security Tools:** OWASP ZAP, Snyk, Aqua Security.

Ethical Considerations:

- Make sure that any sensitive information used in the research is obscured or anonymised.
- Adhere to best practices and ethical norms while managing and processing data in cloud environments.

7. CONCLUSIONS

CI/CD and DevOps are closely related concepts that share the goal of accelerating software changes' delivery to end users through automated build, testing, and deployment procedures. DevOps, on the other hand, is a more comprehensive approach that covers every stage of the product lifecycle, including development, operations, marketing, planning, sales, and human resources. Organisations are essential in allocating resources to develop and deliver trustworthy, high-quality products that fulfil consumer expectations in the fiercely competitive software market. Customers expect more and more constant interaction with DevOps teams, which enables them to offer continual feedback.In software engineering, continuous techniques have become increasingly important for both research and implementation. While many concerns are addressed by current methods, instruments, and procedures, there are still a number of gaps and problems that call for additional study.

8. REFERENCES

[1] Rahman, A. A. U., Helms, E., Williams, L., & Parnin, C. (2015). Synthesizing Continuous Deployment Practices Used in Software Development. 2015 Agile Conference. doi:10.1109/agile.2015.12

[2] J. Humble, and D. Farley, “Continuous Delivery,” 1st Ed. Addison Wesley, 2011

[3] M. Fowler (2013, May 30), Continuous Delivery [Online].

Available: http://martinfowler.com/bliki/ContinuousDelivery.html

[4] P. Swartout, “Continuous Delivery and DevOps: A QuickStart Guide,” 1st Ed. Packet Publishing, 2012

[5] Shahin, M., Babar, M. A., Zahedi, M., & Zhu, L. (2017). Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges. 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). doi:10.1109/esem.2017.18

[6] Asfin Achdian and M Akbar Marwan, “Analysis of CI/CD Application Based on Cloud Computing Services on Fintech Company,” International Research Journal of Advanced Engineering and Science, Volume 4, Issue 3, pp. 112-114, 2019.

[7] S.A.I.B.S. Arachchi, Indika Perera,” Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management”, 978-1- 5386-4417- 1/18/$31.00 ©2018 IEEE

[8] NIYA N J, JASMINE JOSE,” A Literature Survey on Development and Operation”, 2020 IJCRT | Volume 8, Issue 4 April 2020 | ISSN: 2320-2882