Enhancing Operational Efficiency through the Integration of CI/CD and DevOps in Software Deployment

Partha Sarathi Chatterjee
Scholar, Master of Computer Science
Liverpool John Moores University (LJMU, UK)
Email: parthastudent@yahoo.co.in

Harish Kumar Mittal
Thesis Supervisor, Upgrad Education, India,
Principal, BM Institute of Engineering and Technology,
Sonepat, India
Email: harish_mittalin@yahoo.co.in

Abstract **- In the rapidly evolving digital era, the complexity and dynamic nature of software applications have significantly increased, driven by ever-changing consumer and business requirements. This shift poses a challenge to traditional software development and deployment methodologies, which often struggle to keep pace with these rapid changes. This paper explores the transition from conventional methods to agile methodologies, emphasizing their critical role in maintaining application stability and facilitating seamless updates with minimal impact on end-users. Central to this study is the examination of automated deployment models, particularly Continuous Integration/Continuous Deployment (CI/CD), and their transformative impact on the software deployment process. The research delves into the intricacies of the DevOps lifecycle, highlighting the importance of various environments such as Development (Dev), Testing (Test), and Production (Prod). These environments are crucial in ensuring that any updated version of an application is rigorously tested and free of bugs before its deployment in a production setting. Through a comprehensive case study conducted in an AWS lab environment, this paper demonstrates the effectiveness of automated deployment models in overcoming the limitations inherent in manual deployment processes. The findings reveal significant improvements in operational efficiency, product quality, and customer satisfaction. The study also discusses the broader implications of these findings, including the necessity for businesses to adopt modern, agile deployment strategies to stay competitive and responsive in the digital landscape. This research contributes to the understanding of how automated deployment strategies, underpinned by CI/CD and DevOps practices, can revolutionize software development processes. It provides valuable insights for organizations looking to enhance their software deployment methodologies, ultimately leading to improved business outcomes and customer experiences in the digital age.**

Keywords: Automated Deployment, CI/CD, DevOps, Agile Methodologies, Software Application Complexity, Digital Transformation.

I. INTRODUCTION

The transition to a digital lifestyle has fundamentally reshaped market needs. Activities ranging from grocery shopping to booking medical appointments have been simplified to a mere click, thanks to modern software applications. This digital convenience, however, brings forth significant challenges for the IT industry, tasked with meeting the escalating demands of an increasingly digital populace.

In the early days of computing, software applications were limited in scope and complexity, catering to specific, unchanging requirements. Today, the landscape is vastly different. The IT industry confronts the dual challenge of not only scaling these applications to serve a growing user base but also ensuring their stability. This necessitates a dynamic and robust model for rapid enhancement and feature addition, all while maintaining application integrity.

CI/CD and DevOps: Revolutionizing Deployment: The advent of Continuous Integration/Continuous Deployment (CI/CD) within the DevOps framework has been transformative, enabling automated deployment and testing with minimal human intervention. This innovation ensures swift, error-free application delivery, addressing the industry's need for speed and reliability in software deployment [1]

The Shift in Software Development Paradigm:
The last decade has witnessed a paradigm shift in software development, moving from Software as a Product (Saab) to Software as a Service (SaaS). This transition represents a move from single-instance software on customer machines to shared instances running on cloud platforms, reflecting the evolving nature of software consumption and delivery [2]

Containerization in Agile Environments: In agile environments, the need for applications to be scalable, performant, and highly available is paramount. To meet these requirements, containerization has emerged as a widely adopted strategy in the IT industry. Particularly in cloud-based Software as a Service (SaaS) environments, where release cycles are frequent and demanding, containers offer an efficient solution. Their growing popularity is attributed to their ability to provide isolated, consistent, and efficient environments for application deployment, making them ideal for both cloud computing and high-performance computing scenarios [3]

The Essence of CI/CD in Software Development: Continuous Integration and Delivery (CI/CD) have become fundamental to the software development process. Continuous Integration (CI) involves the creation and testing of deployable artifacts from source code, encompassing activities like compilation, code quality checks, and unit tests. Continuous Delivery (CD) extends this process to include the deployment of these artifacts to a production system, following successful integration and testing. The overarching goal of CI/CD is to expedite the process from code changes to deployment, significantly reducing the time to deliver updates and new features [4].

CI/CD: A Paradigm Shift in Development Practices: CI/CD represents a paradigm shift in software development practices. It requires developers to frequently merge their code changes into a shared version control repository, with each check-in being verified by an automated build. This approach enables early detection of problems and ensures that the process of releasing small sets of feature changes is visible, traceable, and largely automated. The CI/CD pipeline orchestrates the build, test, and deployment of software across various environments, culminating in production deployment. This method contrasts sharply with traditional *practices where developers-built application features in* isolation and submitted them separately, often leading to integration challenges [5]

A CI/CD pipeline refers to the largely automated process of integrating committed changes to the code, testing, and then moving the code from the commit stage to the production stage Frequent but small (merge) commits are encouraged Indeed, a large-scale analysis conducted by Zhao et al. determined around 21 (median) commits are made per day for 575 open-source projects [6]

In the past decade, a great shift occurred in software development from creating Software as a Product (Saab), which is executed as a single instance on customers' machines, towards providing Software as a Service (SaaS) where many users share instances that run on cloud. This shift underscores the need for more dynamic, scalable, and cloud-compatible development practices [2]

A. Problem Statement

The core of this research is to delve deeply into the DevOps methodology, utilizing a business case study to illustrate its effectiveness in overcoming the challenges associated with traditional manual deployment processes. The study will dissect the essential components of the DevOps methodology, which include Continuous Planning, Contentious Integration, Continuous Delivery and Continuous Monitoring.

1) Continuous Planning:

The dynamic nature of modern software applications, such as a food delivery service, necessitates continuous planning to incorporate new features and updates. This approach allows for agility in adapting to changing market demands and customer preferences, ensuring the application remains relevant and competitive[5]

2) Contentious Integration (CI)

CI is a critical process in DevOps, where developers merge their changes into a central repository, triggering automatic builds and tests. This automated approach contrasts with manual deployment and testing, which are time-consuming and prone to errors. CI streamlines the development process, enhancing efficiency and reducing the likelihood of miscommunications and errors. Some popular CI tools are available in the present market and their major differences are Jenkins, Bamboo and TeamCity.

3) Continuous Delivery

Following CI, the focus shifts to deploying the application into various pre-production environments, such as DEV, QA, and STAGE.

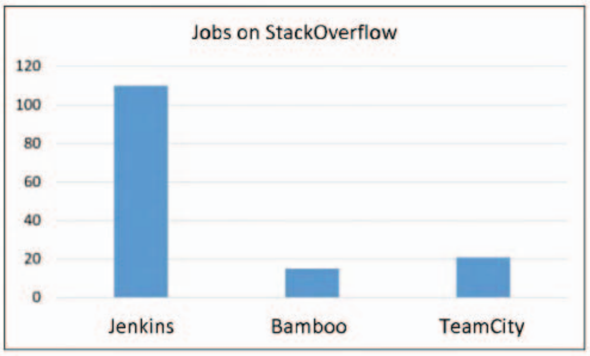

Fig .1. Comparing the Jobs on Stack overflow for various integration tools [7]

This step ensures that the new release is thoroughly tested and free from bugs before final deployment. Continuous Delivery automates this process, significantly reducing the time and effort required compared to manual deployment methods. Some Configuration Management tools are Chef , Puppet, Ansible and Saltstack.

Tools used in CI/CD can be specialized for a particular stage of the pipeline (standalone), such as integration or deployment, or can combine both phases into a single tool that can be used for the entire pipeline (integrated). Many of the popular tools used today are primarily CI solutions that are now being retrofitted to also focus on cloud-based deployments. [6]

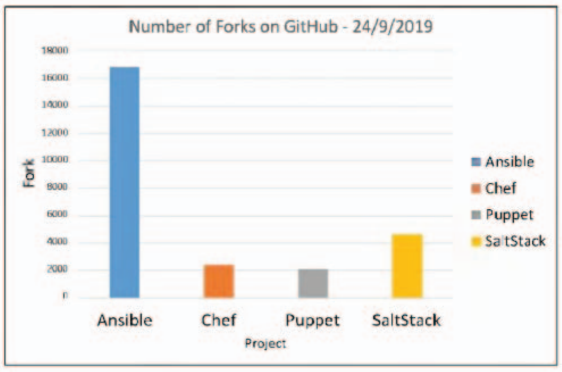

Fig .2. Comparing the number of forks for various deployment tools [7]

4) Continuous Monitoring:

Once deployed, continuous monitoring is essential to maintain the application's performance and stability. This involves proactive monitoring to quickly address any issues, ensuring uninterrupted service to end-users. Continuous Monitoring is crucial in environments where applications are frequently updated and expanded with new features [8]

B. Aims and Objectives:

The study aims to highlight the limitations and challenges of manual deployment in the current business landscape and explore how automated strategies can effectively address these issues.

Objectives -

- **To Critically Evaluate Manual Deployment Strategies:** A detailed examination of existing manual deployment strategies, identifying and analyzing the key challenges and inefficiencies they present in a modern business context.
- **To Explore Automated Deployment Strategies:** Understanding automated deployment methods, especially CI/CD pipelines, and assessing their potential in overcoming the limitations identified in manual deployment processes.
- **Conducting AWS-Based Experimental Analysis:** Implement within an AWS cloud environment to showcase the application and effectiveness of automated deployment strategies.
- **Evidence-Based Evaluation:** Systematically gather and analyze empirical data from the experiment to demonstrate how automation can address and mitigate the inherent challenges of manual deployment processes.

C. Significance of the Study

As software applications grow in complexity, the industry faces the dual challenge of timely feature releases and maintaining application stability. Miscommunication between development and operations teams often exacerbates these challenges, leading to production issues and potential business disruptions. This research is significant for its focus on analysing the limitations of manual deployment processes in this high-stakes environment. Key areas of investigation include:

- **Efficiency in Feature Release:** Assessing how manual deployment may hinder timely updates, impacting business responsiveness and competitiveness.
- **Reduction of Miscommunication Risks:** Exploring the role of manual processes in operational errors and inefficiencies due to team miscommunications.
- **Advantages of Automated Deployment:** Demonstrating the potential of automation, particularly through CI/CD pipelines, to streamline deployment, reduce errors, and enhance overall operational effectiveness.

This study aims to provide valuable insights into optimizing deployment strategies in cloud computing, offering practical solutions to current industry challenges.

D. Scope of the study

This research focuses on analyzing the limitations of manual application deployment and the advantages of transitioning to automated deployment methods, especially CI/CD pipelines. The study aims to:

1. **Identify Key Challenges in Manual Deployment:**
   - Assessing impacts on customer retention, meeting business deadlines, and potential revenue losses.
   - Evaluating the competitive disadvantages due to slower delivery and inefficiencies in service delivery.
   - Analyzing risks of manual errors, miscommunication, and difficulties in version rollback.
2. **Explore Benefits of Automated Deployment:**
   - Conducting a comparative analysis to understand how automation can address these challenges.
   - Highlighting improvements in operational efficiency, error reduction, and customer service responsiveness.

The scope is to empirically demonstrate how automated deployment can enhance business processes in software development and cloud computing, offering a clear contrast to manual methods.

E. Structure of the Paper

This Section includes A brief background details about the rapid changing in the IT industry and its demands. We also discussed the problem statement briefly which we are going to see in detail in the upcoming sections. We covered the significance of our study which is to analyse everything from a business point of view and about the scope of the study. This research paper is organized into the following sections to provide a thorough exploration of the transition from manual to automated deployment strategies in cloud computing:

Literature Review - This section reviews existing literature, examining the evolution of deployment strategies, the development of CI/CD practices, and previous research on manual versus automated deployment processes.

Methodology - Here, the research methodology is outlined, including the design of the AWS-based experimental setup, data collection methods, and the approach for the comparative analysis between manual and automated deployment.

Experimental Setup and Analysis - This section presents experimental study and a detailed case study that illustrates the practical application of CI/CD pipelines in a business environment, along with an analysis of the results obtained from the experimental setup.

Research Outcome and Analysis - In this section, the findings from the case study and experimental analysis are discussed, comparing these results with existing literature and highlighting their implications for the IT industry.

Conclusion and Future Work - The final section summarizes the key findings of the research, discusses its implications, and suggests areas for future research. It also reflects on the broader impact of automated deployment strategies in the field of cloud computing.

II. LITERATURE REVIEW

In this section, we explore the concept of DevOps and its advantages over the traditional waterfall model. We examine various research works to understand and analyze the benefits of the DevOps methodology. As the demand for software applications grows, the need for a dynamic deployment strategy becomes crucial. A strategy that ensures timely delivery and contributes significantly to maintaining application quality is essential. We will delve into why DevOps is instrumental in fulfilling these requirements.

A. DevOps Conception

DevOps represents a paradigm shift in the IT industry, bridging the gap between developers focused on automated development and operations teams concentrated on automated deployment and monitoring systems. DevOps fosters an organizational shift where both teams work closely as a single unit, delivering faster and reducing communication gaps [9]. The primary goal of DevOps is to minimize the divide between developers and operations teams, promoting continuous high-quality service delivery [10]

Buttar et al. (2023) discuss how DevOps enables enterprises to deliver high-quality software capabilities efficiently. This study focuses on the use of DevOps for cloud application deployment, emphasizing cost, memory, and CPU capacity optimizations. [11]

Here is a graphical representation of how DevOps methodology works and we will discuss each step-in detail

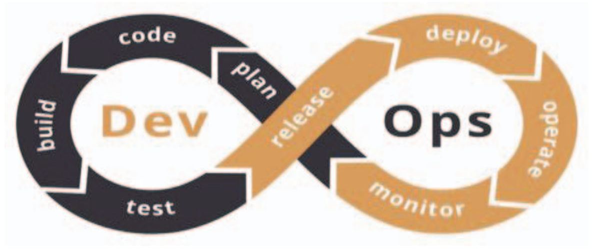

Fig .3. DevOps life cycle

The DevOps lifecycle, as illustrated in the diagram, encompasses stages like plan, code, build, and test, falling under Continuous Integration (CI). We categorize tasks for the CI section into major categories:

Continuous Planning

Continuous planning is vital for designing and planning customer requirements before building the software. In agile methodology, accommodating all requirements concurrently may not be feasible, but a set of requirements can be addressed in each sprint. This approach allows for feedback and adjustments based on customer satisfaction with delivered features.[12]

Continuous coding and building

After analyzing and identifying requirements, the next stage is continuous coding and building, where developers add new code to meet the evolving requirements.

Continuous Testing

As new features and code modifications are added, continuous testing becomes crucial. Testers perform unit and integration testing to ensure each new feature functions correctly and the entire application operates as intended.

Continuous Release

Following successful code pushes and testing, the next step is to release new artifacts with required changes. These artifacts are typically stored in an artifact repository before deployment.

Continuous Delivery/ Continuous Development

This stage involves deciding on strategies to deploy the application in various environments. Continuous Delivery and Continuous Deployment differ in their approach to deployment [10]. Ali (2023) explores the integration of DevOps and CI/CD in software development, highlighting how this combination streamlines processes and enhances release efficiency. They highlighted collaborative benefits of DevOps and CI/CD, as well as the challenges associated with their implementation. [13].

Continuous Deployment

In these delivery strategies, there is no manual intervention required and the build artifact is directly deployed into the infrastructure. Once the developer modifies the code or adds any new feature. the new artifact is directly deployed. Now, in most cases, the artifact is deployed in lower-level environments such as sandbox or pre-production. Then after rigorous testing, once all teams are satisfied, they migrate the code into a production environment [10]. Salameh's provides insights into the elements of DevOps relevant to automation, visibility, and control practices. [14]

Continuous Delivery

Continuous Delivery automates the build and artifact formation, but manual approval is required before production deployment. This strategy is commonly used, where development and testing teams review the outcome before approving deployment [15].

Continuous Operation and Monitoring

Post-deployment, maintaining the stability and uptime of the application is crucial. The IT operations team handles reactive and proactive faults, ensuring minimal downtime and impact on end-users. Continuous monitoring and logging are employed to proactively identify and resolve issues, preventing production outages.

The study by Bhattacharya and Mittal [17] delves into the intricate dynamics of container runtimes within Kubernetes clusters, offering a nuanced exploration of their performance metrics. They highlighted the critical role these runtimes play in optimizing containerized applications, thereby underpinning the operational efficiencies pivotal to modern software deployment strategies. This investigation not only broadens the understanding of container runtime capabilities but also provides a foundational basis for future research aimed at enhancing the agility and responsiveness of cloud-native technologies

B. Popular Tools in DevOps

DevOps methodology relies on a stack of technical tools fitting into various stages of the process. Kupale and Powar (2023) discuss how Azure DevOps supports efficient software development, deployment, and maintenance. This study illustrates the practical application of DevOps in creating efficiency and improving communication in software development. [16]. We will explore the roles of these tools in DevOps.

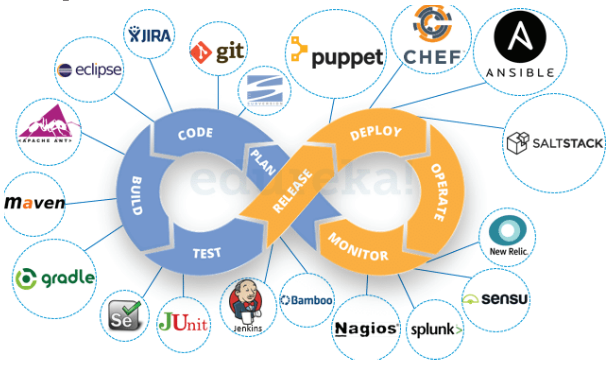

Figure 4. Tools used in DevOps

The diagram showcases popular tools in DevOps. We will examine some of these tools:

- **Git -** A Version Control System (VCS) allowing multiple users to share code within a team.
- **Eclipse -** An IDE widely used for coding, supporting various languages through plugins.
- **Jira -** a versatile tool used for bug tracking, agile project management, and change management ticketing.
- **Maven/Gradle -** used for building artifacts after coding, facilitating the creation of deployable software components.
- **Junit /Se -** used for running test cases, including unit and integration testing.
- **Jenkins /Bamboo** - These tools are crucial for building CI/CD pipelines, automating stages like build, test, release, and deploy without manual intervention [4]
- **Puppet/Chef/Ansible/SaltStack -** These are configuration management tools used for tasks like server patching or application installation across multiple servers.
- **Nagios/Splunk/Sensu -** These tools are used for proactive monitoring, detecting faults early to prevent server damage or outages.

This section discussed various DevOps concepts and their role in agile deployment strategies. We analyzed the major components of DevOps, including Continuous Integration and Continuous Delivery, and the differences between them. We also examined the steps involved in Continuous Integration and Continuous Deployment, along with the tools and applications used in each step of DevOps across business units.

III. RESEARCH METHODOLOGY

This research undertakes a comparative analysis between manual deployment and automated deployment using CI/CD methodologies. The analysis will be anchored around a hypothetical business case study, illustrating the impact of persisting with manual deployment processes. We will identify and scrutinize the challenges faced in manual deployment and demonstrate how these can be effectively addressed through automated CI/CD strategies.

Evaluating Business Impact Through Defined Metrics: The analysis will focus on specific metrics that are commonly impacted by deployment strategies:

1. **Loss of Existing Customer Base:** Assessing how manual deployment affects customer retention.
2. **Missing Business Deadlines:** Evaluating the impact on project timelines.
3. **Losing Business Revenues:** Analyzing the financial implications of inefficient deployment.
4. **Slower Delivery Compared to Competitors:** Measuring the pace of deployment against industry standards.
5. **Inefficiency in Delivery Model:** Examining the operational delays and customer service implications.
6. **Manual Errors and Miscommunication:** Investigating the time and efficiency lost due to manual processes.
7. **Challenges in Version Rollback:** Understanding the complexities involved in reverting to previous versions in manual setups.

A. AWS Lab Setup for Deployment

To comprehensively understand and compare the automated and manual deployment processes, we will construct a laboratory environment within AWS. This lab will serve as a controlled setting for deploying a sample application, enabling us to directly observe and measure the efficiencies and challenges of each deployment method.

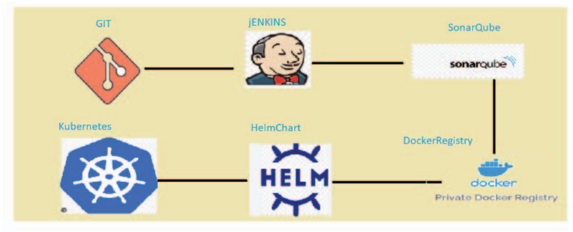

Fig .5. Lab diagram

The diagram provides a visual representation of the lab setup, detailing the infrastructure components and their interconnections within the AWS environment.

B. Result or Outcome

The results from the lab experiment will be analyzed against the predefined setback metrics to evaluate the effectiveness of automated deployment. Key outcomes include:

1. **Loss of Existing Customer Base:** Demonstrating how automation enhances project delivery timelines, thereby improving customer satisfaction and retention.

2. **Missing Business Deadlines:** Showcasing the efficiency of automated pipelines in meeting project deadlines with minimal manual intervention.

3. **Losing Business Revenues:** Illustrating the reduction in service penalties due to adherence to SLAs through automated processes.

4. **Competitive Delivery Speed:** Highlighting how structured and automated deployment accelerates delivery, outpacing manual methods.

C. Tools Used

The experiment will utilize a suite of tools within the AWS cloud platform, each playing a critical role in the deployment pipeline:

1. **GitHub:** Serving as the code repository.

2. **Maven:** Used for building and testing the application.

3. **Docker:** Creating the final application artifact as a Docker image.

4. **Docker Registry:** Acting as the repository for Docker images.

5. **Jenkins:** Automating the CI/CD pipeline, facilitating build, test, and deployment phases.

6. **Kubernetes (EKS):** Hosting the application within a managed Kubernetes cluster.

This study will delve into the drawbacks of manual deployment processes, using the AWS lab to demonstrate how these challenges are mitigated through an automated CI/CD pipeline. The discussion will focus on the tangible improvements observed in the lab experiment, correlating them with the broader research objectives.

This section summarizes the research methodology, including the lab setup and the tools employed. It outlines the expected outcomes of the experiment; particularly how automated deployment strategies can mitigate common drawbacks associated with manual processes. The summary ties back to the initial research objectives, setting the stage for a comprehensive analysis of the results.

IV. Experimental Analysis and Implementation

We will establish a laboratory environment within AWS to conduct our experiment. This lab will simulate real-world deployment scenarios, enabling us to compare the efficacy of manual versus automated deployment processes. The diagram in Fig. 5 provides a detailed visual representation of our AWS lab setup. It illustrates the infrastructure components, including computing resources, networking setup, and their interconnections, crucial for understanding the experimental environment.

A. Integration and Automation in the CI/CD Pipeline

Our experiment begins with storing sample Java code in GitHub, our chosen Version Control System (VCS). The integration of GitHub with Jenkins is key to triggering automated builds using Maven whenever code modifications occur.

During the build process, SonarQube will automatically assess code quality, ensuring it meets predefined standards. Subsequently, a Docker Image is created and stored in the Docker Registry, marking the completion of the Continuous Integration (CI) phase.

In the CD phase, the Docker image is deployed into Kubernetes, specifically Elastic Kubernetes Service (EKS), using Helm Charts for deployment orchestration. Jenkins automates these tasks, including helm chart creation and Docker image deployment into the Kubernetes Pod, ensuring a seamless and manual-intervention-free process.

Upon a new code push to the repository, the entire process from Docker image creation, code quality checking, to deployment in a Kubernetes pod, is executed automatically.

This automation highlights the efficiency and speed of the CI/CD pipeline in contrast to manual deployment methods.

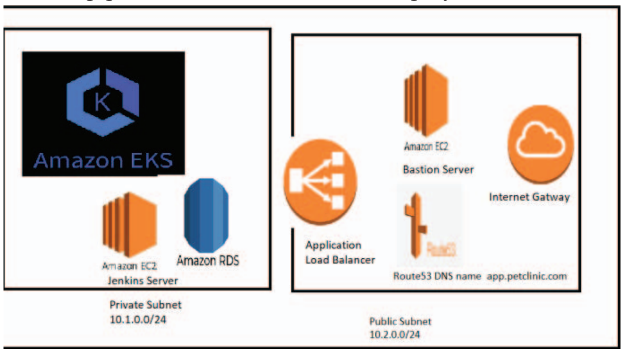

Fig .6. AWS Topology

The AWS topology diagram (Fig. 6) illustrates how the Pet Clinic application will be deployed within the Amazon EKS Cluster. Jenkins, hosted on an EC2 instance, will pull code from GitHub, build the artifact, and deploy it within the Kubernetes cluster. This setup ensures that each code update triggers an automated CI/CD pipeline, deploying artifacts from Docker Hub into our EKS cluster.

B. Load Balancing and DNS Configuration:

The application will be accessible via an Application Load Balancer with a public subnet. Route53 will be used for DNS management, creating an easy-to-remember domain (apps.petclinic.com) mapped against the load balancer's DNS. This setup ensures that end-users can access the application seamlessly over the internet.

For backend data storage, Amazon RDS will be used, deployed within a private subnet for enhanced security.

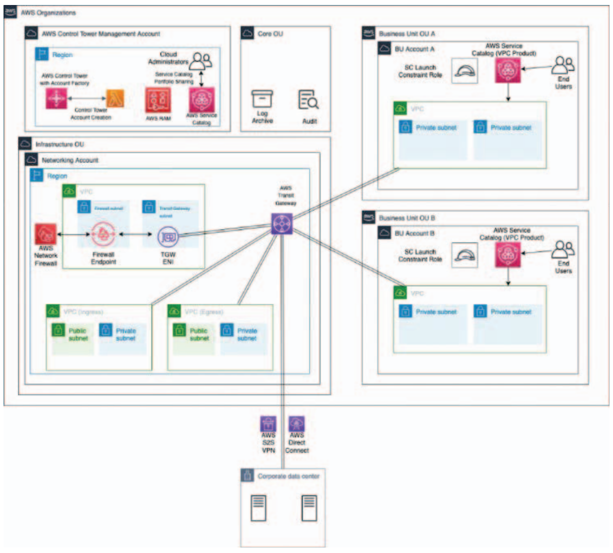

Fig .7. AWS Topology with Multiple Environment

This architecture is designed to facilitate secure communication between different virtual private clouds (VPCs) across multiple AWS accounts. This arrangement allows for isolated and thorough testing in each environment, ensuring the final application deployed in production is robust and error-free.

C. Case Study Analysis:

This segment of the research involves a case study with a hypothetical organization to evaluate the benefits of automated CI/CD deployment processes. We will conduct a comparative analysis under two distinct scenarios: one employing traditional manual deployment methods and the other utilizing automated deployment strategies.

Case Study: Pet Clinic

Pet Clinic, a UK-based veterinary service provider, faces operational challenges due to an increase in customer demand. To streamline their services, they decide to transition online, offering appointment bookings and product purchases through their website. This move necessitates the formation of an IT department tasked with developing a robust online platform within 60 days.

Scenario 1: Manual Deployment Process

Challenges of Manual Deployment: Adopting a manual deployment approach with a waterfall strategy, the IT team encounters several setbacks:

- **Development and Testing Delays:** Coding and manual testing processes consume significant time, leading to delays and inefficiencies.
- **Deployment Challenges:** The application, deployed just before the deadline, is riddled with issues, impacting user experience and functionality.
- **Impact Analysis:** We will assess how this manual approach affected various business metrics, including customer retention, deadline adherence, and revenue generation.

Scenario 2: Automated Deployment process using CI/CD pipeline

In this scenario, Pet Clinic's IT team opts for an agile methodology, integrating an automated Continuous Integration and Continuous Deployment (CI/CD) pipeline. This approach is a strategic shift from the traditional waterfall model, aiming to enhance efficiency, reduce time-to-market, and improve the overall quality of the software deployment process.

Sprint-Based Development Approach:

- **Minimum Viable Product (MVP):** The team plans to release the application in multiple sprints, with the first sprint focusing on deploying a basic yet functional MVP. This approach allows for early feedback and iterative improvements.

- **Multiple Environment Setup:** To ensure thorough testing and quality assurance, the team sets up three distinct environments: Development (Dev), Pre-Production (Pre-Pod), and Production (Prod). Each environment serves a specific purpose in the testing and deployment lifecycle.

- **Sprint Duration and Objectives:** Each sprint spans 20 days, within which a set of features is developed, tested, and prepared for release. The objective is to incrementally build upon the MVP, adding features and enhancements in each sprint.

Automated CI/CD Pipeline Implementation:

- **Integration with Version Control:** The team uses GitHub for version control, ensuring all code changes are tracked and managed efficiently.

- **Automated Builds with Jenkins:** Jenkins, integrated with GitHub, automates the build process. Every code commit triggers a new build, ensuring continuous integration of new features and bug fixes.

- **Quality Checks and Dockerization:** SonarQube is employed to maintain code quality standards. Post quality checks, the code is packaged into Docker images, ready for deployment.

- **Deployment with Kubernetes and Helm:** The Docker images are deployed into an Elastic Kubernetes Service (EKS) cluster using Helm charts. This process is fully automated and managed through Jenkins, minimizing manual intervention.

Benefits of the Automated Approach:

- **Enhanced Speed and Efficiency:** The automated pipeline significantly reduces the time taken from code commit to deployment, ensuring rapid delivery of features.

- **Improved Quality and Reliability:** Continuous testing in multiple environments ensures that each release is stable and meets quality standards.

- **Increased Customer Satisfaction:** The agile approach, coupled with faster and reliable releases, leads to enhanced user experience and customer satisfaction.

- **Scalability and Flexibility:** The CI/CD pipeline provides the scalability needed to accommodate future enhancements and the flexibility to adapt to changing requirements.

Outcome of the Automated Deployment:

- **Timely Project Completion:** The project is completed within the 60-day deadline, with each sprint successfully delivering its objectives.

- **Positive Customer Feedback:** The deployed application receives positive feedback from users, leading to increased customer retention and acquisition.

- **Business Growth:** The successful deployment and operation of the online platform contribute to the growth and expansion of Pet Clinic's business.

In summary, this scenario demonstrates the effectiveness of adopting an agile methodology and an automated CI/CD pipeline in overcoming the challenges faced in manual deployment processes. The approach not only meets the immediate needs of Pet Clinic but also sets a foundation for continuous improvement and growth.

The study by Nandi et al. [18] offers insights into the broader implications of automation in the realm of CRM. Their research underscores the dual role of automation systems as both facilitators and potential obstacles in achieving success, suggesting that the strategic integration of technology is paramount in enhancing business processes and outcomes. This enriches our discussion on the technological advancements in Kubernetes, highlighting the importance of thoughtful technology adoption beyond technical efficiencies.

The section concludes with a detailed analysis of the lab setup and the experimental steps. It contrasts the outcomes of manual versus automated deployment strategies, focusing on how the latter mitigates drawbacks associated with manual processes.

V. RESEARCH OUTCOME AND ANALYSIS

The experiment conducted in previous section provided a clear comparative analysis between manual and automated deployment strategies. We evaluated the impact of these strategies on key business metrics, which are summarized in the table below:

Table 1. Drawback Metrics

Drawback Metrics	Scenario 1: : Manual Deployment	Scenario 2: Automated Deployment
1) Loss of existing customer base	TRUE	FALSE
2) Missing business deadlines	TRUE	FALSE
3)Losing business revenues	TRUE	FALSE
4) Slow delivery process than competitors	TRUE	FALSE
5) Inefficient delivery model:	TRUE	FALSE

A. *Business Impact and Key Performance Indicators (KPIs):*

We now turn our attention to the business or revenue impact of these deployment strategies, analysing common KPIs and their fulfilment in both scenarios.

- **Customer Satisfaction and Market Reputation:**
  - Customer satisfaction is crucial for revenue growth and market reputation. Successful organizations prioritize customer needs and satisfaction
- **Accuracy and Timeliness:**
  - Delivering accurate products within the agreed timeline is essential for business success. Maintaining on-time delivery is a key indicator of operational efficiency.
- **Quality and Efficiency:**
  - Delivering high-quality products and services on the first attempt minimizes waste, effort, and cost, contributing to overall business efficiency.

Now Let us understand what the impact for both of our scenarios was.

1) *Impact on KPIs in Scenario 1, Manual Deployment Strategy*

- **Customer First:** The application delivered was of inferior quality, leading to customer dissatisfaction.
- **On-timeline Delivery (OTD):** Despite meeting the deadline, the product was plagued with issues, failing to meet quality standards.
- **Right First Time:** The application was delivered with numerous bugs, indicating a failure to meet this KPI.

2) *KPI Fulfilment in Scenario 2, Automated Deployment Strategy*

- **Customer First:** The automated strategy ensured high product quality and met customer requirements, leading to increased satisfaction.
- **On-timeline Delivery (OTD):** The application was delivered within the deadline, with superior quality.
- **Right First Time:** The application fulfilled all user requirements without issues, indicating success in this KPI.

This section summarizes the outcomes of the experiment and their implications from a business perspective. The analysis of KPIs for both manual and automated deployment scenarios reveals significant differences in their impact on business performance. The automated deployment strategy not only mitigated the drawbacks identified in the manual approach but also positively influenced key business indicators.

Table 2. KPI Metrics

KPI Metrics	Baseline Scenario	Scenario 1: Manual Deployment	Scenario 2: Automated Deployment
Customer First	Moderate Satisfaction	Slight Increase	Significant Increase
On-timeline Delivery (OTD)	Frequent Delays	Reduced Delays	Consistent Timeliness
Right First Time	Low Success Rate	Improved Success	High Success Rate

The table above encapsulates the overall performance of each deployment strategy against the defined KPIs, highlighting the superior outcomes achieved through automated deployment processes.

VI. Conclusion and Future Work

This research has comprehensively demonstrated the significant impact of deployment strategies on business efficiency, product quality, and customer satisfaction. Through a detailed comparative analysis between manual and automated deployment processes, it is evident that the traditional manual deployment, particularly following a waterfall strategy, leads to numerous challenges. These include prolonged delivery times, subpar product quality, and ultimately, customer dissatisfaction. The experiment highlighted the lack of strategic planning and efficient testing in manual processes, resulting in unnoticed bugs and issues in the production environment.

In stark contrast, the automated deployment process, structured into multiple sprints and supported by rigorous testing across various environments, proved to be markedly more efficient. This approach not only enhanced the quality of the final product but also significantly improved customer satisfaction. The automated process, with its strategic division of development into sprints and comprehensive testing, ensures the delivery of a high-quality, user-friendly product.

In the current digital era, where the demand for rapid and efficient software deployment is continuously escalating, adopting an automated deployment process is increasingly beneficial. However, the decision to upgrade to such a system should be informed by a thorough analysis of both current capabilities and future business needs. For growing businesses, the initial investment in automated deployment systems can lead to substantial long-term operational cost savings and efficiency improvements. It is crucial for organizations to consider their present and future growth trajectories and the potential impact on customers when deciding on their deployment strategy.

Future Work

Looking ahead, there are several avenues for further research and development in this area:

1. **Scalability and Adaptability:** Future studies could focus on the scalability of automated deployment processes in larger, more complex organizational structures and diverse operational environments.

2. **Integration with Emerging Technologies:** Exploring the integration of automated deployment processes with emerging technologies like AI and machine learning could provide insights into further enhancing efficiency and predictive capabilities in deployment strategies.

3. **Long-Term Impact Study:** Conducting a longitudinal study on organizations that have transitioned from manual to automated processes would offer valuable insights into the long-term impacts, including operational cost analysis and return on investment.

4. **Customization for SMEs:** Tailoring automated deployment strategies for small and medium-sized enterprises (SMEs), considering their unique challenges and resource constraints, would be a valuable area of research.

In conclusion, while the shift to automated deployment processes presents clear advantages in terms of efficiency and customer satisfaction, it is imperative for each organization to conduct comprehensive research to determine the most effective strategy that aligns with their specific business goals and market position. The insights gained from this research provide a strong foundation for organizations looking to navigate the complexities of modern software deployment in an increasingly digital world.

REFERENCES

[1] J. Mahboob and J. Coffman, ‘A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework’, in 2021 IEEE 11th Annual Computing and Communication Workshop and Conference, CCWC 2021, 2021, pp. 529–535.

[2] T. Rangnau, R. v. Buijtenen, F. Fransen, and F. Turkmen, ‘Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines’, in Proceedings - 2020 IEEE 24th International Enterprise Distributed Object Computing Conference, EDOC 2020, 2020, pp. 145–154.

[3] M. K. Abhishek, D. R. Rao, and K. Subrahmanyam, ‘Framework to Deploy Containers using Kubernetes and CI/CD Pipeline’, International Journal of Advanced Computer Science and Applications, vol. 13, no. 4, pp. 522–526, 2022.

[4] T. F. Dullmann, O. Kabierschke, and A. van Hoorn, ‘StalkCD: A Model-Driven Framework for Interoperability and Analysis of CI/CD Pipelines’, in Proceedings - 2021 47th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2021, 2021, pp. 214–223.

[5] R. Parashar, ‘Path to success with CICD pipeline delivery’, International Journal of Research in Engineering, Science and Management, vol. 4, no. 6, pp. 271–273, 2021.

[6] S. Haque, Z. Eberhart, A. Bansal, and C. McMillan, ‘Semantic Similarity Metrics for Evaluating Source Code Summarization’, in IEEE International Conference on Program Comprehension, 2022, pp. 36–47.

[7] S. Mysari and V. Bejgam, ‘Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible’, in International Conference on Emerging Trends in Information Technology and Engineering, ic-ETITE 2020, Institute of Electrical and Electronics Engineers Inc., Feb. 2020. doi: 10.1109/ic-ETITE47903.2020.239.

[8] K. Purohit, ‘Executing DevOps & CI/CD, Reduce in Manual Dependency’, IJSDR2009086 International Journal of Scientific Development and Research, 2020, [Online]. Available: www.ijsdr.org

[9] A. Hemon, B. Lyonnet, F. Rowe, and B. Fitzgerald, ‘From Agile to DevOps: Smart Skills and Collaborations’, Information Systems Frontiers, vol. 22, no. 4, pp. 927–945, Aug. 2020, doi: 10.1007/s10796-019-09905-1.

[10] S. M. Mohammad, ‘DevOps automation and Agile methodology’, Article in SSRN Electronic Journal, 2017, doi: 10.1729/Journal.24060.

[11] A. M. Buttar et al., ‘Optimization of DevOps Transformation for Cloud-Based Applications’, Electronics 2023, Vol. 12, Page 357, vol. 12, no. 2, p. 357, Jan. 2023, doi: 10.3390/ELECTRONICS12020357.

[12] A. Wahaballa, O. Wahballa, M. Abdellatief, H. Xiong, and Z. Qin, ‘Toward unified DevOps model’, in Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS, IEEE Computer Society, Nov. 2015, pp. 211–214. doi: 10.1109/ICSESS.2015.7339039.

[13] J. M. Ali and J. M. Ali, ‘DevOps and continuous integration/continuous deployment (CI/CD) automation’, Advances in Engineering Innovation, vol. AEI Vol.4, no. 1, pp. 38–42, Nov. 2023, doi: 10.54254/2977-3903/4/2023031.

[14] H Salameh, ‘The impact of devops automation, controls, and visibility practices on software continuous deployment and delivery’, in Proceedings of the 2nd International Conference on Research in, Dec. 2019, pp. 22–46. Accessed: Jan. 02, 2024. [Online]. Available: https://www.dpublication.com/wp-content/uploads/2019/09/IME-F793.pdf

[15] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, ‘DevOps’, IEEE Softw, vol. 33, no. 3, pp. 94–100, May 2016, doi: 10.1109/MS.2016.68.

[16] A. A. Kupale and P. S. Powar, ‘International Journal of Computer Science and Mobile Computing Azure DevOps The Next Era of Application Lifecycle Management’, International Journal of Computer Science and Mobile Computing, vol. 12, no. 7, pp. 1–6, 2023, doi: 10.47760/ijcsmc.2023.v12i07.001.

[17] M. H. Bhattacharya and H. K. Mittal, “Exploring the Performance of Container Runtimes within Kubernetes Clusters”, IJC, vol. 22, no. 4, pp. 509-514, Dec. 2023.

[18] V. Tewari Nandi, H. K. Mittal, and V.V.R. Raman, "Automation Systems: Driver or Inhibitor for Successful CRM," in 2013 Third International Conference on Advanced Computing & Communication Technologies, April 2013, doi: 10.1109/ACCT.2013.79.