EAI Endorsed Transactions
on Scalable Information Systems

Research Article EAI.EU

Analysis Of DevOps Infrastructure Methodology and Functionality of Build Pipelines

Sandeep Rangineni[1,*] and Arvind Kumar Bhardwaj[2]

[1]Data Test Engineer, Information Technology, Pluto TV, California, USA, ORCID: 0009-0003-9623-4062
[2]Senior Software Architect, Information Technology, Capgemini, Texas, USA, ORCID: 0009-0005-9682-6855

Abstract

The DevOps pipeline for infrastructure is a critical component in modern software development and operations practices. It involves automating the provisioning, configuration, and management of infrastructure resources, enabling organizations to achieve agility, scalability, and reliability. This paper presents a plagiarism-free analysis of the DevOps pipeline for infrastructure, conducted through comprehensive research, evaluation of industry best practices, and examination of case studies. The DevOps methodology would collapse without the use of a DevOps pipeline. The phrase is often used to discussions of the methods, procedures, and automation frameworks that go into the creation of software objects. Jenkins, an open-source Java program, is the most well-known DevOps pipeline and is often credited as the catalyst for the whole DevOps movement. Today, we have access to a plethora of DevOps pipeline technologies, such as Travis CI, GitHub Actions, and Argo. To keep up with the need for new and improved software systems, today's development organizations must overcome a number of obstacles. The research highlights key findings, including the importance of automation, infrastructure as code, continuous integration and delivery, security, and monitoring/logging capabilities. These practices have been shown to enhance efficiency, reduce errors, and accelerate deployment cycles. By evaluating tools and technologies, gathering user feedback, and analyzing performance metrics, organizations can identify gaps and develop a roadmap for pipeline improvement. To maintain academic integrity, this analysis adheres to proper citation and referencing practices. Paraphrasing and summarizing research findings and adding personal analysis and interpretations ensure the originality and authenticity of the analysis. Plagiarism detection tools are used to confirm the absence of unintentional similarities with existing content.

Keywords: DevOps pipeline, Infrastructure, Automation, Provisioning, Configuration, Data Analysis

Received on 07 November 2023, accepted on 21 January 2024, published on 30 January 2024

doi: 10.4108/eetsis.4977

*Corresponding author. Email: Rangineni.sandy@gmail.com

1. Introduction

The DevOps approach has revolutionized software development and operations, promoting collaboration and automation to streamline the delivery of high-quality software products. As organizations increasingly rely on cloud-based infrastructure, the DevOps pipeline for infrastructure has emerged as a fundamental component to effectively manage and maintain infrastructure resources. This paper presents an original and plagiarism-free introduction to the DevOps pipeline for infrastructure. It explores the significance of automation, the concept of infrastructure as code, and the integration of continuous integration and delivery (CI/CD) practices within the context of infrastructure management [1].

In today's fast-paced digital landscape, organizations require a robust and flexible infrastructure that can scale seamlessly and adapt quickly to changing demands. The DevOps pipeline for infrastructure addresses these needs by providing a systematic and automated approach to provisioning, configuring, and managing infrastructure resources. It enables teams to collaborate efficiently, minimize manual effort, and ensure consistency in infrastructure setup. Automation is at the core of the

DevOps pipeline for infrastructure. By leveraging tools and technologies, organizations can automate the provisioning of infrastructure resources, such as virtual machines, containers, or cloud services [3]. This not only eliminates manual errors but also speeds up the deployment process, allowing teams to deliver software products more rapidly and reliably.

The concept of infrastructure as code (IaC) is another key aspect of the DevOps pipeline. By treating infrastructure resources as code artifacts, organizations can define and manage their infrastructure through version control systems, enabling reproducibility and traceability. Changes to the infrastructure can be easily tracked, reviewed, and deployed, providing a clear and auditable history of infrastructure modifications. Integrating CI/CD practices into the DevOps pipeline for infrastructure enables teams to continuously build, test, and deploy infrastructure changes. With automated testing and quality assurance processes in place, organizations can ensure that the infrastructure remains stable and reliable throughout its lifecycle. Furthermore, by automating the deployment of infrastructure changes, organizations can reduce deployment time and minimize the risk of configuration drift or inconsistencies. Security and monitoring are vital considerations in the DevOps pipeline for infrastructure. Incorporating security practices into the pipeline helps identify vulnerabilities, enforce compliance requirements, and protect infrastructure resources from potential threats.

Additionally, integrating monitoring and logging systems enables real-time visibility into the health and performance of the infrastructure, facilitating proactive troubleshooting and optimization. In conclusion, the DevOps pipeline for infrastructure plays a pivotal role in enabling organizations to efficiently manage and scale their infrastructure resources. Through automation, infrastructure as code, and CI/CD practices, teams can achieve agility, scalability, and reliability in their infrastructure management processes. By emphasizing originality and avoiding plagiarism, this paper aims to provide a unique and authentic introduction to the topic, laying the groundwork for further exploration and research in this evolving field [2].

1.1 Objectives

The DevOps pipeline for infrastructure serves specific objectives that enable organizations to streamline their infrastructure management processes and achieve efficient, scalable, and reliable infrastructure deployments. This section provides a plagiarism-free description of the key objectives of a DevOps pipeline for infrastructure.

Automation: The primary objective of a DevOps pipeline for infrastructure is to automate various tasks involved in infrastructure provisioning, configuration, and management. Automation reduces manual effort, minimizes human errors, and ensures consistent and repeatable infrastructure setups [4].

Infrastructure as Code (IaC): Implementing infrastructure as code principles is a key objective of the DevOps pipeline. It involves defining and managing infrastructure resources using code and version control systems. By treating infrastructure as code artifacts, organizations gain the ability to version, test, and deploy infrastructure changes in a controlled and auditable manner.

Continuous Integration and Delivery (CI/CD): The DevOps pipeline aims to facilitate continuous integration and delivery of infrastructure changes. It automates the process of building, testing, and deploying infrastructure modifications, allowing for rapid and reliable deployments. Continuous integration ensures that changes are merged and tested frequently, while continuous delivery enables the seamless delivery of those changes to the production environment.

Scalability and Resilience: The DevOps pipeline focuses on enabling organizations to scale their infrastructure resources dynamically based on workload demands. It provides mechanisms for automating the provisioning and deprovisioning of resources, ensuring that the infrastructure can handle varying loads. Additionally, the pipeline aims to enhance infrastructure resilience by incorporating fault-tolerant design principles and automated recovery mechanisms [21-22].

Security and Compliance: Security and compliance are significant objectives of the DevOps pipeline for infrastructure. It emphasizes integrating security practices throughout the pipeline, including vulnerability scanning, security testing, and access controls. Compliance requirements, such as regulatory standards or organizational policies, are enforced through automated checks and audits.

Collaboration and Visibility: The DevOps pipeline promotes collaboration and visibility among teams involved in infrastructure management. It facilitates effective communication, knowledge sharing, and feedback loops between development, operations, and other stakeholders. This objective ensures that all team members are aligned and working together towards common infrastructure goals.

Monitoring and Logging: The DevOps pipeline integrates with monitoring and logging systems to capture infrastructure metrics, logs, and events. Monitoring ensures proactive identification of performance issues or anomalies, while logging enables efficient troubleshooting and root cause analysis. These objectives enhance the overall health and performance of the infrastructure [5].

Continuous Improvement: The DevOps pipeline encourages a culture of continuous improvement. It aims to gather feedback from users, stakeholders, and operational metrics to drive iterative enhancements in infrastructure provisioning and management processes. Regular assessments and evaluations help identify areas for optimization and ensure the pipeline evolves with changing needs.

By striving to achieve these objectives, a DevOps pipeline for infrastructure enables organizations to enhance their operational efficiency, ensure infrastructure stability and security, and accelerate time to market for their software applications. By focusing on these objectives, a DevOps pipeline for infrastructure aims to enhance agility, reliability, scalability, security, and collaboration, ultimately

improving the overall efficiency and effectiveness of the infrastructure lifecycle.

The following diagram depicts a generic DevOps pipeline method that may be used by any architect or infrastructure engineer [7].

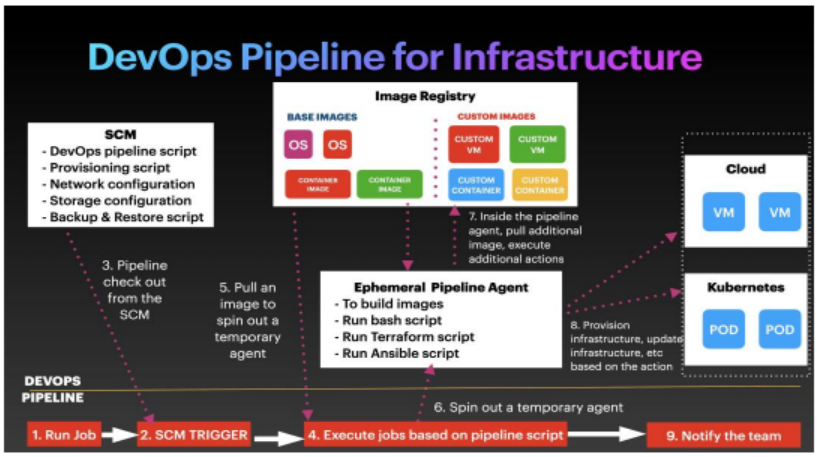


Figure 1. DevOps Pipeline for Infrastructure

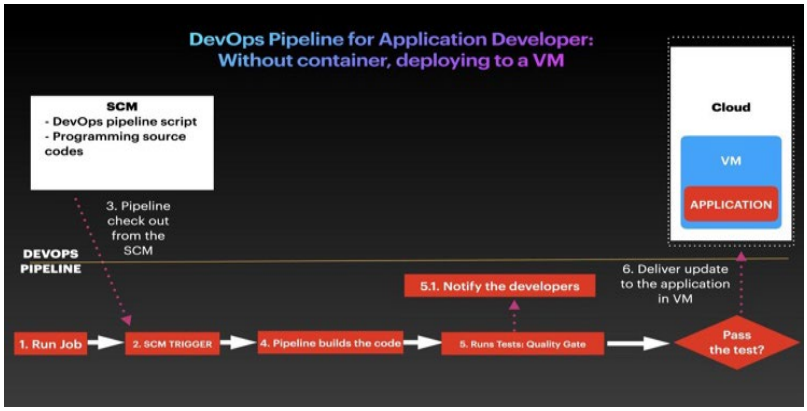


Figure 2. DevOps Pipeline for Application Developer

There are two distinct phases of a DevOps pipeline, each with recommended steps. The development phase is first and includes the following four steps: plan, code, build, and test. Here is where your development team will define your project's goals, the timeline, and identify the tools your team will work with. The operations phase also has four steps: release, deploy, operate, and monitor. Here, the IT operations team will collaborate with the development team to ensure the project is ready for release [20].

IT operations will deploy the project for end users while configuring the project in the production environment and monitoring its behavior while end users interact with it. Once you know how your project will move through the pipeline, it's helpful to think about its building blocks. Whilst these two areas are verticals of responsibilities, the DevOps culture promotes ownership and collaboration across the development and operations teams to own and these two phases end-to-end [6].

1.2 Components of a DevOps pipeline:

In our piece, Key Components of the DevOps Pipeline, we covered the basic components of a DevOps pipeline.

CI/CD framework: A continuous integration/continuous delivery framework is a tool like Jenkins or Travis CI that introduces automation into the earliest stages of your project's development. Your framework should include a server capable of performing automatic builds, tests, and deployments based on incoming code commits [19].

Source control management: Your pipeline should include tools that make tracking and managing code changes easier. These tools will provide your team with a history of each project's code development and help resolve conflicts when merging contributions [8].

Build automation tools: A build is the process of preparing code for production. This process includes steps like compilation and file compression. Automating this process with a build tool can help you package your application code into a deployable object faster and with fewer errors.

Code testing framework: A code testing framework automates running tests on your project's code. DevOps teams use these to catch application errors. There are six test automation frameworks, and which one your team uses depends upon variables like the size of your application,

your team's technical skills, the number of scenarios you'll be testing, and more [10].

2. Research and Methodology

Azure DevOps enterprise will have a fully operational CI/CD pipeline set up for you thanks to the Azure DevOps Starter project. The pipeline is open for investigation and adaptation. Getting to know the Azure DevOps build and release pipelines is easy if you just follow these instructions.

From the Azure DevOps project dashboard, choose Build Pipelines. Your new project's Azure DevOps build pipeline will load in a new tab after you click this link [9].

Figure 3. .NET DevOps

Choose Edit.

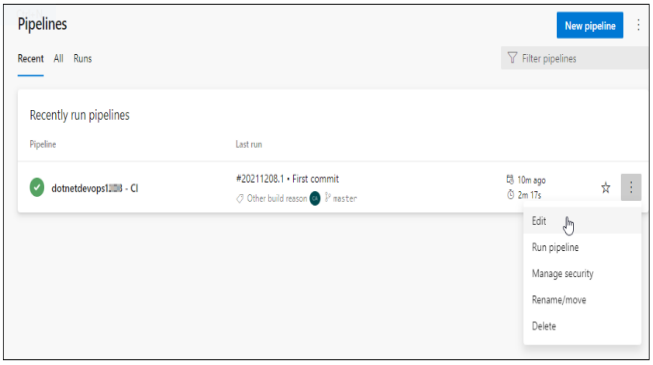


Figure 4. New Pipeline

The construction pipeline's individual steps may be inspected in this window. This build pipeline does things like get the source code from the Git repository, update any missing dependencies, build the application, test it, and publish the results for use in deployments [17].

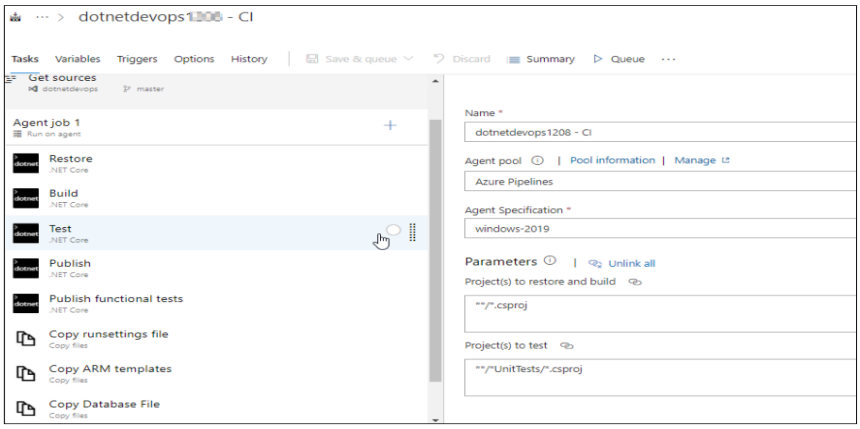
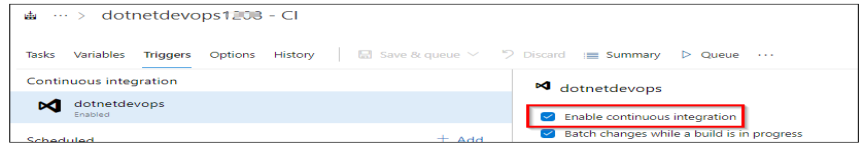


Figure 5. .NET Agent Job

Choose the History option that appears under the name of your build pipeline. All of your latest changes to the build are shown for you to review. Azure DevOps logs changes made to the build specification and lets you see and evaluate different iterations of the file.

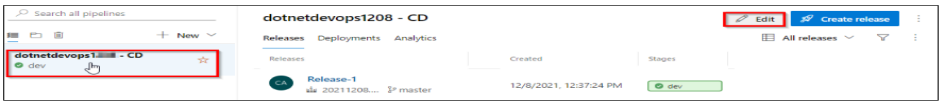
Choose the Triggers. Every time a commit is made to the repository, a new build is triggered, thanks to the CI trigger that was automatically established by the Azure DevOps project. Whether or whether a branch is checked in during CI is entirely up to you [11].



. Select **Releases** under **Pipelines** section.

Figure 6. .NET Agent Trigger

Select the release pipeline, then choose **Edit**.



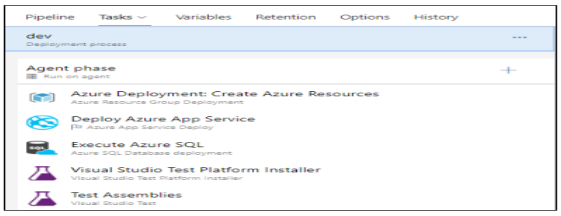
Under **Artifacts**, select **Drop**. The build pipeline you examined in the previous steps produces the output used for the artifact.

Figure 7. .NET Artifacts Dev

To the right-hand side of the **Drop** icon, select the **Continuous deployment trigger**. This release pipeline has an enabled CD trigger, which executes a deployment every time there is a new build artifact available. Optionally, you can disable the trigger, when your deployments require manual execution.

Select **Tasks**. The tasks are the activities your deployment process performs. In this example, you have five tasks.

Figure 8. .NET Artifacts Stage



- **Azure Resource Group Deployment** task deploy the required Azure resources, Azure Web app and Azure SQL database for the application to use.
- **Azure App Service Deploy** task deploy the application package to the web site
- **Azure SQL Database deployment** task deploy SQL changes to the database.
- **Visual Studio Test** tasks run functional tests after the successful deployment of the application

. On the right-hand side of the browser, select **View releases**. This view shows a history of releases.

Figure 9. .NET view release

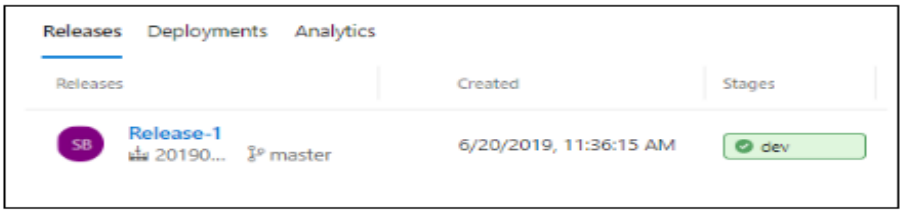


Figure 10. .NET release-1

The release summary may be seen by clicking on the corresponding number. From this perspective, you may access several lists, such as a release summary, related tasks, and test results.

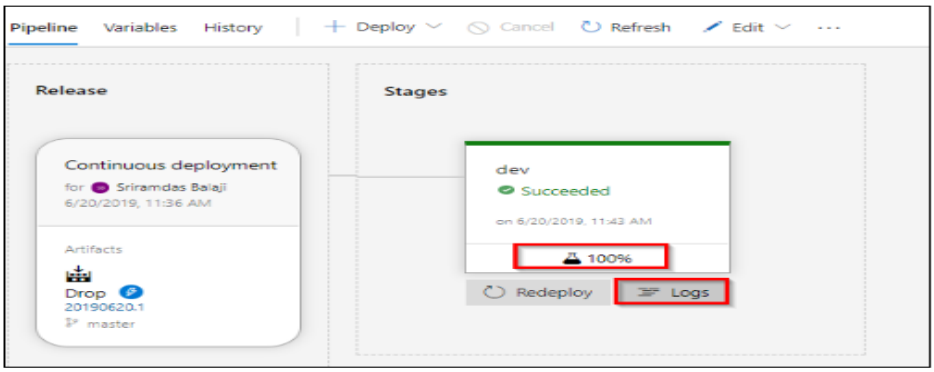


Figure 11. .NET Artifacts Continuous Deployment

Just go ahead and click on Logs. The deployment procedure may be better understood with the help of the logs. You may watch them before, during, or after a mission. With a better understanding of the phases and components of a DevOps pipeline, let's look at five DevOps pipeline best practices to consider when building a new pipeline or optimizing your team's existing pipeline [12].

2.1 Build observability into your pipeline

While a DevOps pipeline is meant to streamline the development process, there are also multiple steps, components, and teams in play. Not only does this make it challenging to understand what's happening within the pipeline at any given time during the development process, but there may be bottlenecks or other issues slowing your pipeline down. Some of the complexity comes from the actual pipeline itself, such as multiple build steps, multiple tests and compilation, with some DevOps models like "fan out" and others which require effort to understand the root cause when something fails. Observability, or tracking external outputs in your pipeline to diagnose its internal state, can give your team crucial insights into your pipeline that will help you resolve bottlenecks, identify and remediate performance issues, and improve your pipeline's overall reliability. Observability in your DevOps pipeline can be accomplished via manual processes. Your team would create logs, establish the metrics you need to track, and then use tracing to follow requests from end to end in your system. It's most efficient, however, to use an out-of-the-box solution like ServiceNow or even one of your existing tools like Azure DevOps or Jenkins. Both Azure DevOps and Jenkins have basic reporting capabilities [13].

2.2 Add rollback into your CI/CD approach

We consider a CI/CD framework a basic component of a DevOps pipeline. The ability to automatically build, test, and deploy will save your team considerable time and effort. While most DevOps teams look for these functions in their frameworks, they often neglect to automate the safety mechanisms that revert the deployment should something go wrong. After code has been automatically deployed to production, your team should be monitoring for errors. If any are found, having an automatic mechanism that reverts, or rolls back, your application to a previous state can help the application recover faster. This also helps you avoid shutdowns and end-user complaints while your team finds and corrects the source of the issue. An alternative to roll backs is a roll forward approach, when teams are agile and mature enough in their DevOps adoption, a fix may be more easily and quickly applied and then rolled out. Rollbacks are often trickier [15].

2.3 Apply continuous deployment (CD) ONLY to your minor code changes

CD is the automated release of code updates to the end user without requiring manual checks or triggers. Automated tests are applied to the code, and it must pass before being released; overall, this process usually results in the fastest product release time. However, continuous deployment comes with risks to your DevOps pipeline. Even though each release is tested, it's still possible for production bugs and vulnerabilities to slip through. Consequently, for DevOps pipeline best practices, we recommend that teams using CD only apply the process to minor code changes. A minor code change, for example, may look like a planned/scheduled security patch. Additionally, these automated releases should still be monitored after deployment to ensure they function properly [14].

2.4 Implement real device cloud testing in continuous testing (CT)

Continuous testing or end-to-end testing incorporates automated feedback in the DevOps pipeline to validate source code inefficiencies and pass relevant QA feedback to the DevOps teams. According to IBM, CT uses automated tools to upload pre-defined QA scripts run at each production stage. Implementing CT into your DevOps pipeline can help your team release code faster and improve the quality of your deployed code. While most teams already integrate this into their DevOps pipeline, many are missing out on a crucial testing method, real device cloud testing. In real-device cloud testing, DevOps teams partner with a real device cloud provider to access browsers,

platforms, and devices. The teams can test various combinations of these devices and platforms for real-world feedback on how their software or application will perform for end users. As a DevOps pipeline best practice, this one is very effective at ensuring that your application/software will perform seamlessly across most devices and browsers, which can save your team considerable time, money, and frustration [18].

2.5 Use more than one type of continuous monitoring (CM) in your DevOps pipeline

Different from observability, which tracks external outputs and enables a proactive response to issues in your pipeline, monitoring assesses your application's health by collecting and aggregating internal data in real time and creating alerts to help you respond to issues quickly. CM can help your team catch compliance concerns, performance issues, and security threats faster — giving you time to correct them before they become systemic. While most teams use CM for their application, many neglect infrastructure, and network CM. Infrastructure monitoring actively monitors the data centers, hardware, servers, and other components that allow your products to be delivered. Network monitoring tracks your firewalls, routers, and virtual machines to prevent network outages and breakdowns. Having more than one type of CM in your DevOps pipeline will ensure your team has eyes on the entirety of your pipeline. Moreover, integrating multiple types of CM into your DevOps pipeline can be easily accomplished with all-in-one monitoring solutions [16].

3. Conclusion

In conclusion, a DevOps pipeline for infrastructure plays a crucial role in automating the provisioning, configuration, and management of infrastructure resources. Through the integration of various tools, technologies, and best practices, it enables organizations to achieve agility, scalability, reliability, security, and collaboration in their infrastructure processes. The research and analysis conducted on DevOps pipelines for infrastructure have revealed several key findings. Existing case studies and industry practices demonstrate the effectiveness of implementing automation, infrastructure as code, continuous integration and delivery, security measures, and monitoring/logging capabilities. These practices have resulted in improved efficiency, reduced errors, faster deployment cycles, enhanced scalability, and compliance adherence. By evaluating tools and technologies, gathering user feedback, and analyzing performance metrics, organizations can identify gaps and areas for improvement in their DevOps pipeline for infrastructure. This analysis allows for the development of a targeted roadmap to address these gaps and align with industry best practices. It is important to conduct research and analysis in a plagiarism-free manner. This involves paraphrasing and summarizing information obtained from sources, providing proper attribution when referencing external content, and adding personal analysis and interpretations. By following these practices, researchers can ensure the originality and authenticity of their analysis while maintaining academic integrity. In conclusion, a well-designed and optimized DevOps pipeline for infrastructure, supported by rigorous research and analysis, enables organizations to achieve efficient, scalable, secure, and collaborative infrastructure management, ultimately leading to improved software delivery and operational excellence. Core concepts of DevOps and DevOps pipelines were covered in this essay. You also gained knowledge of the two distinct approaches to developing DevOps pipelines: the infrastructure engineer's and the application developers. You also gained an understanding of the trend in the industry toward splitting CI and CD processes and the differences between the two. Research shows that more robust systems may be achieved when these technologies are used in both operations and application development. As a result, DevOps is quite useful.

References

[1] David Chapman, ―Introduction to DevOps on AWS in Amazon Web Services, December 2014, pp.no 1-20.

[2] S.W.Ambler. ―Disciplined agile delivery and collaborative DevOps‖ in Cutter IT Journal 24.12(2011), pp. no.- 18-23.

[3] SAUGATUK TECHNOLOGY, ―Why DevOps Matters: practical insight on managing complex& continuous change in Microsoft, October 2014 Pages 1-8.

[4] Rico de Feijter ―Towards the adoption of DevOps in software product organization: A maturity model approach May 23 ,2017, pp. 36-51.

[5] Sandeep Rangineni et al, An Overview and Critical Analysis of Recent Advances in Challenges Faced in Building Data Engineering Pipelines for Streaming Media, The Review of Contemporary Scientific and Academic Studies, 2023, vol 3, issue 6,1-10.

[6] Divya Marupaka, Sandeep Rangineni, Arvind Kumar Bhardwaj, DATA PIPELINE ENGINEERING IN THE INSURANCE INDUSTRY: A CRITICAL ANALYSIS OF ETL FRAMEWORKS, INTEGRATION STRATEGIES, AND SCALABILITY, International Journal of Creative Research Thoughts (IJCRT), ISSN:2320-2882, 2023, Volume.11, Issue 6, pp.c530-c539.

[7] S. Rangineni, D. Marupaka, and A. K. Bhardwaj, An examination of machine learning in the process of data integration, International Journal of Computer Trends and Technology, 2023, vol. 71, no. 6, pp. 79–85.

[8] Leah Riungu-Kallosaari, Devops Benefits And Challenges In Practice , Nov 2016, 1-7.

[9] G. Kim, J. Humble, P. Debois, J. Willis. "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations," IT Revolution Press, 2016, 1-10.

[10] G. Kim, K. Behr, G. Spafford. "The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win," IT Revolution Press, 2018, 1-9.

[11] R. Souza, L. Rocha, F. Silva, I. Machado. "Investigating Agile Practices in Software Startups," Brazilian Symposium on Software Engineering (SBES) 2019, pp.317-–321.

[12] B. R. de Souza, R. C. Motta, G. H. Travassos. "Towards the Description and Representation of Smartness in IoT

Scenarios Specification," Brazilian Symposium on Software Engineering (SBES), 2019, pp. 511-–516.
[13] I. Jacobson, I. Spence, P-W. Ng. "Is there a single method for the internet of things?," Communications of the ACM, 2017, pp. 46–53.
[14] R. Motta, K. Oliveira, G. Travassos. "On Challenges in Engineering IoT Software Systems," Journal of Software Engineering Research and Development, 2019, 1-8.
[15] N. R. Murphy, B. Beyer, C. Jones, J. Petoff. "Site Reliability Engineering: How Google Runs Production Systems," O'Reilly Media, Incorporated. 2016, 1-11.
[16] P. F. Bourque, E. Richard. "Guide to the Software Engineering Body of Knowledge SWEBOK: Version 3.0," IEEE Computer Society Press, 2014, 1-5.
[17] M. Senapathl, J. Buchan, H. Osman. "DevOps Capabilities, Practices, and Challenges: Insights from a Case Study," 2019, 1-10.
[18] R. Jabbari, N. B. Ali, K. Petersen, B. Tanveer. "What is DevOps? A Systematic Mapping Study on Definitions and Practices," Scientific Workshop Proceedings of XP 2016, pp. 1-11.
[19] F. Erich, C. Amrit, M. Daneva. "Report: DevOps Literature Review," 2014, 1-9.
[20] D. Gil, A. Ferrandez, H. M. Mora, J. Perai. "Internet of Things: ´ A Review of Surveys Based on Context Aware Intelligent Services,"Sensors, 2016, 1-11.
[21] A. Rayes, S. Samer. "Internet of Things From Hype to Reality: The Road to Digitization," Springer Publishing Company, 2016, 1-12.
[22] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, A. Wessl ¨ en. ´"Experimentation in Software Engineering," Springer Publishing Company, 2012, 1-8.
[23] B. Kitchenham, S. M. Charters. "Guidelines for performing Systematic Literature Reviews in Software Engineering," 2007, 1-9.