DEVOPS FOR OPTIMIZING DATABASE MANAGEMENT: PRACTICE IMPLEMENTATION, CHALLENGES AND COMPARISON WITH THE CONTEXT-BASED APPROACH

Hassane Tahir

Department of Computer Science, IUT de Paris - Rives de Seine, Université Paris Cité
143 Av. de Versailles, 75016 Paris, France

ABSTRACT

Integrating DevOps practices into database management presents unique challenges that must be addressed to fully harness the benefits of agile development and continuous delivery. The rapid and iterative nature of DevOps often clashes with traditional database management methodologies, leading to inefficiencies and bottlenecks. This paper explores the challenges organizations encounter when implementing DevOps to optimize database management, including managing the complexities of data state, ensuring synchronization between application and database changes, and automating database deployments. It also highlights the cultural shift needed to bridge the divide between development and operations teams, underscoring the importance of collaboration and communication. In meeting these challenges, database management practices can be effectively incorporated into DevOps workflows to create greater operational efficiencies, faster delivery cycles and improved quality. We finish this research paper by comparing DevOps approach for database management with the context approach that we have used to improve database administration procedures.

KEYWORDS

Agile Development, Automating Deployments, Continuous Delivery, Context Approach, Database Administration, Database Changes, Database Management, DevOps Approach, Practices, Procedures.

1. INTRODUCTION

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) to deliver features, fixes, and updates more rapidly while aligning with business objectives. Database DevOps extends these principles to include database code within the same workflows as application code, addressing a common bottleneck: database code changes. It integrates the principles, tools, and processes of DevOps into database management workflows, ensuring that data store updates are synchronized with the broader application development pipeline. This approach bridges the gap between database management and the agile, collaborative mindset of DevOps, enabling smoother, faster, and more synchronized software delivery cycles.

A key component of Database DevOps is treating database code like application code, employing similar tools and capabilities to create a systematic and automated approach to change management. This process begins with developers, who are often tasked with making changes and writing SQL code. Experienced database administrators (DBAs) often review database changes to detect poor code (i.e. queries) that could compromise performance, security, or data integrity.

While this system works well in theory, practical implementation often reveals inefficiencies. Database deployments remain a significant bottleneck in many organizations. Manual processes dominate database code changes, with DBA reviews often delaying production releases. Developers face frustration as their weeks-old code changes await review, stalling progress. The traditional database modification process, in its current form, is an obstacle. To overcome this, teams need to move DBA reviews earlier in the process and consolidate all code changes into a unified workflow.

Incorporating database management into the DevOps pipeline will help meet the requirements to solve common bottlenecks related to database changes. Treating database code like application code and employing automation tools allows teams to synchronize updates, streamline workflows, and accelerate software delivery. The adoption of more and more database management practices in DevOps will improve the duration and speed of development cycles [11] [17].

This paper explores the integration of DevOps practices into database management. It begins by outlining the key steps involved in the process of implementing DevOps practices within database operations. Next, it examines the challenges organizations face during this process, such as data persistence, the complexity of managing large data sizes, and ensuring data integrity across different environments. Finally, the paper concludes with a comparative analysis of the DevOps approach to database management and the Context approach to database administration.

2. PROCESS OF IMPLEMENTING DEVOPS FOR DATABASE MANAGEMENT

Implementing DevOps for database management involves systematically integrating database operations into the DevOps pipeline. Figure 1 shows the main steps involved in the Process of implementing DevOps for database management. The following subsections present a detailed breakdown of the process.

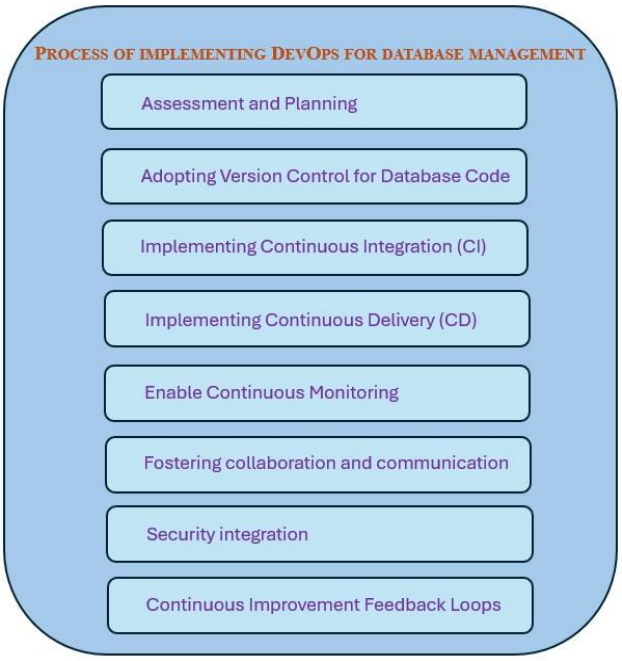

Figure 1. Main steps in the Process of implementing DevOps for database

2.1. Assessment and Planning

Implementing Database DevOps begins with a thorough assessment and planning phase. First, evaluate the current state of your database management processes, including workflows, tools, and practices used in database development, deployment, and maintenance. This evaluation provides a baseline understanding of existing procedures. Next, clearly define the objectives for adopting Database DevOps, which typically include reducing deployment times, minimizing errors, fostering collaboration, and enhancing security. With these goals in mind, develop a comprehensive transition plan. This plan should outline the steps necessary to integrate DevOps practices into your database management workflows and include detailed timelines, resource allocation, and key milestones. By carefully assessing the current state, setting clear objectives, and planning the transition, organizations can effectively prepare for and implement Database DevOps, ultimately achieving more efficient and streamlined database operations.

2.2. Adopting Version Control for Database Code

Adopting version control for database code is essential for effective database management within a DevOps framework. By using tools like Git, organizations can manage database scripts and schemas efficiently, ensuring that all changes are meticulously tracked. This practice allows for previous versions to be restored when necessary, providing a safety net against errors. Additionally, employing branching and merging strategies facilitates parallel work on database code, enabling multiple developers to collaborate without conflicts. This approach allows the simultaneous development of different features or fixes without interfering with the work of others. Version control not only enables collaboration, but also reliability of the database management process and overall quality. It ensures that all team members are on the same page, reduces the risk of errors, and streamlines the workflow. The adoption of practices allows organizations to significantly improve their database management capabilities, by aligning them with the DevOps principles of continuous integration and delivery (CI/CD), ultimately leading to more robust and efficient software development cycles.

2.3. Implementing Continuous Integration (CI)

Implementing Continuous Integration (CI) for database management is a crucial step in ensuring efficient and reliable software development cycles. The process begins by setting up automated builds for database changes using CI tools like Jenkins [8], GitLab CI [5], or Azure DevOps [12]. These tools help manage and streamline the integration of changes into the database, ensuring that every modification triggers a build process that validates the changes. This automated build process is very important to maintain the integrity and stability of the database because it makes it possible to detect very early conflicts and errors which could arise in new code.

Automating builds involves configuring the CI tools to monitor changes to the database code repository. Whenever a developer commits a change, the CI tool automatically initiates the build process. This build process typically includes steps such as compiling the code, running database scripts, and preparing the database environment. Automating these tasks allows teams to significantly reduce the time and effort required to integrate changes into the database, leading to more frequent releases and faster development cycles.

In addition to automating builds, it is equally important to integrate automated testing into the CI pipeline. Automated testing plays a vital role in ensuring that database changes do not cause issues. This includes running unit tests, which test individual components or functions of the database code to ensure they work as expected. Integration tests are also essential as they verify

that different parts of the application work together seamlessly with the database. Performance tests are carried out to ensure that changes do not have a negative effect on database's performance, such as system resource usage and query execution times.

Integrating automated testing into the CI pipeline involves setting up testing frameworks and tools that can execute the tests automatically as part of the build process. Popular tools for this purpose include NUnit [14], Junit [9], and Selenium[16] for unit and integration testing, and tools like JMeter [2] for performance testing. The CI tool is configured to run these tests whenever a new build is initiated. Then the test results are analyzed, and any problems or failures are immediately notified to the development team.

This continuous feedback loop is an essential feature of CI that allows developers to quickly identify and resolve problems. Identifying and addressing issues early in the development process helps teams prevent the cumulative effects of bugs and defects, resulting in more stable and reliable software. Furthermore, automated testing reduces the reliance on manual testing, which can be time-consuming and prone to human error.

Another key benefit of implementing CI for database management is the ability to ensure consistent and repeatable builds and tests. Since the build and test processes are automated, they are performed the same way every time, leading to more predictable and reliable outcomes. This consistency is crucial for maintaining the quality and stability of the database across different environments, such as development, testing, staging, and production. Moreover, CI practices promote better collaboration and communication within development teams.

2.4. Implementing Continuous Delivery (CD)

Implementing Continuous Delivery (CD) [7] for database management is a critical step in ensuring efficient and reliable database deployments. The process involves automating the deployment of database changes using Continuous Delivery tools. This automation includes setting up scripts and pipelines to deploy changes across different environments, such as development, testing, staging, and production. Deployment automation enables organizations to streamline the process, guarantee consistency in all environments and reduce manual errors.

The first step in automating deployments is to configure CD tools like Jenkins, GitLab CI, or Azure DevOps to handle database changes. This involves creating scripts that can execute database updates and setting up pipelines that define the workflow for deploying these changes. Each pipeline typically includes the stages of database creation, testing, and the deployment of changes, so that they are thoroughly validated before reaching production.

One of the key practices in Continuous Delivery is the use of rolling deployments. Rolling deployments apply changes incrementally across database instances rather than all at once. This approach minimizes downtime and reduces the risk of errors during deployment. In a rolling deployment, a subset of database instances is updated first while the others continue to operate. Once the changes are verified and deemed stable, the deployment process continues with the next subset of instances.

Automating deployments also involve implementing robust testing mechanisms within the CD pipeline. Automated tests, including unit tests, integration tests, and performance tests, are crucial to ensure that database changes do not introduce any issues. These tests validate the changes at each stage of the pipeline, providing immediate feedback to developers and database administrators. By catching and addressing problems early, teams can maintain high-quality database operations and avoid last-minute firefighting.

Another important aspect of Continuous Delivery is environment consistency. Automating deployments ensure that the same process is followed across all environments, from development to production. This consistency helps prevent environment-specific issues and makes it easier to reproduce and debug problems. Additionally, using infrastructure-as-code (IaC) tools like Terraform [21] or Ansible [1] can further enhance environment consistency by automating the provisioning and configuration of the infrastructure on which the database runs.

2.5. Enable Continuous Monitoring

Continuous monitoring is a critical aspect of DevOps for database management, ensuring the ongoing health and performance of the database environment. This process includes the implementation of various monitoring tools and techniques that make it possible to generate real-time information on database operations. By enabling continuous monitoring, organizations can track key metrics such as performance, availability, and security, ensuring that any issues are detected and addressed promptly. The implementation begins with selecting appropriate monitoring tools. Among the Tools that are widely used for their robust capabilities in collecting, analyzing, and visualizing data, we can cite Prometheus [15] , Grafana [6] , and the ELK Stack (Elasticsearch, Logstash, Kibana). These tools enable teams to set up dashboards and alerts that provide a comprehensive view of the database's status.

The ability to track performance metrics is one of the main benefits of continuous monitoring. This includes monitoring query execution times, system resource usage, and transaction rates. By closely monitoring these metrics allows teams to identify performance bottlenecks and optimize database operations. In addition to performance, continuous monitoring also covers availability. Monitoring tools help detect outages or downtimes, allowing teams to respond swiftly and restore services. Automated alerts enable administrators to be informed of any disruption, thereby reducing resolution time and minimizing disruption impact on users. Security is another critical component of continuous monitoring. By continuously tracking security credentials and access patterns, organizations can detect potential vulnerabilities and unauthorized access attempts. This proactive approach helps safeguard the database against both external and internal threats.

It is necessary to integrate these tools into the CI/CD pipeline to effectively implement continuous monitoring. This integration can ensure that monitoring is indeed an ongoing process rather than a periodic inspection. Automated monitoring scripts can be included in the deployment pipeline, providing real-time feedback on the status of the database. Moreover, continuous monitoring facilitates better collaboration and communication within the team.

2.6. Fostering Collaboration and Communication

Fostering collaboration and communication is a fundamental aspect of implementing DevOps for database management. It starts with creating cross-functional teams that include developers, database administrators (DBAs), and operations staff. These teams work together throughout the development and deployment processes, ensuring that all perspectives are considered and that there is a shared understanding of goals and challenges.

Regular meetings, such as daily stand-ups and retrospectives, play a crucial role in maintaining open lines of communication. These meetings provide a platform for team members to discuss progress, address any issues, and plan next steps. Effective collaboration tools can effectively simplify and facilitate communication. Tools like Slack [4], Microsoft Teams [13], and Jira [3] enable real-time communication, task management, and issue tracking.

Documentation is another key element. To help team members have a clear understanding of processes, procedures and configurations, documentation should be maintained and updated. This documentation serves as a reference, helping new team members get up to speed quickly, reducing the reliance on verbal communication. Encouraging a culture of continuous feedback is vital for continuous improvement. This can be achieved through regular reviews, peer feedback sessions, and performance metrics analysis.

Team members should have received training and development opportunities to ensure they acquire necessary skills and knowledge. This involves technical training on tools and technologies, and soft skills training required for improving communication and collaboration. Another important aspect is aligning incentives and objectives across teams. Making sure that all teams have common goals and performance measures promotes a sense of unity and collectiveness responsibility. This alignment helps in driving coordinated efforts towards achieving the desired outcomes.

2.7. Security Integration

Incorporating security into DevOps for database management is very important to protect sensitive data and maintain database integrity throughout the development and deployment processes. This approach, known as DevSecOps, integrates security practices into the CI/CD pipeline and helps ensure continuous protection.

Integrating security into DevOps for database management includes integrating security measures throughout the CI/CD pipeline, implementing strict access controls, data encryption, continuous monitoring, conducting regular audits, promoting a DevSecOps culture, leveraging automation, maintaining and updating comprehensive documentation. These practices ensure that databases are protected against threats and maintain a high level of security and integrity, supporting the overall goals of DevOps.

2.8. Continuous Improvement Feedback Loops

Continuous improvement is a cornerstone of successful Database DevOps, focusing on enhancing processes through regular feedback and ongoing training. Establishing feedback loops is essential to gather input from both team members and stakeholders. These loops provide a structured way to capture insights, identify challenges, and propose solutions. Organizations can adjust their database DevOps processes by regularly collecting and analysing this feedback regularly. The goal is to ensure that they evolve to meet changing needs and improve over time. These strategies ensure that Database DevOps processes are constantly refined and enhanced, leading to more efficient and effective database management

3. CHALLENGES OF DEVOPS FOR DATABASE MANAGEMENT

Implementing DevOps for database management introduces significant challenges including data persistence, data size complexity, maintaining data Integrity across environments, handling schema changes, and many other challenges as shown in Figure 2. The following sections present the different challenges [10].

3.1. Data Persistence Challenge

Data persistence is a significant challenge in implementing DevOps for database management. Unlike application code that can be easily version-controlled and rolled back, databases must

retain their data through changes. This need for persistent data retention complicates migrations and updates. Larger databases (e.g., 1 TB or more) amplify challenges due to their size and complexity. Ensuring that data is preserved across different environments (development, testing, staging, production) is also vital. Effective database DevOps practices must address these complexities to maintain consistent and reliable database operations.

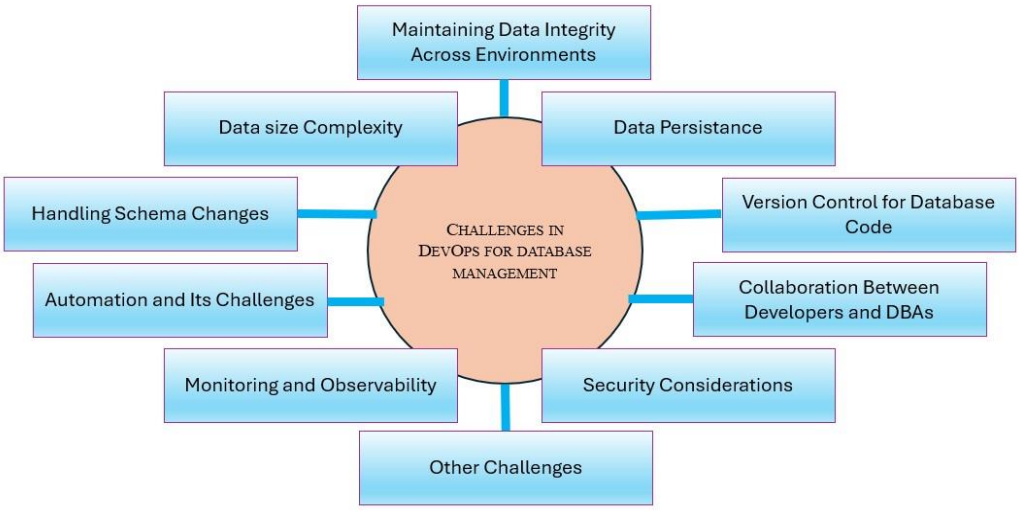

Figure 2. Challenges of DevOps for database management

3.2. Data Size Complexity

Data size complexity is a significant challenge in implementing DevOps for database management. Database size has a direct impact on the speed and complexity of deployments. Small databases pose minimal challenges and can be easily managed. However, as database sizes grow to terabytes (TB) or petabytes (PB), the complexity increases substantially. Large databases require careful planning and consideration for several reasons. First, the time needed to perform migrations, backups, and restores increases with data size. This can lead to longer deployment windows and potential downtime. Second, the risk of data loss or corruption during migrations is higher, necessitating meticulous execution of migration scripts.

Performance can also be affected by the size of the database. Large datasets may result in more query execution time and increased resource consumption, impacting the entire system performance. Ensuring data consistency across different environments becomes more challenging as the size and complexity of the data grows.

3.3. Maintaining Data Integrity Across Environments

Maintaining data integrity across environments is a critical challenge in implementing DevOps for database management. This ensures that data remains accurate, consistent and reliable as it progresses through the various stages, from development to testing, staging and production. Each environment may have different data states, making it challenging to ensure that database changes applied in one environment will work correctly in another. For example, development environments often use mocked or subsetted data, whereas production environments use fullscale data, leading to potential discrepancies.

3.4. Handling Schema Changes

Handling schema changes in Database DevOps is quite complex. These changes should be handled carefully to avoid data loss or corruption. This includes adding, updating tables, columns or constraints. Synchronizing schema changes with application updates is essential to ensure that both the application and the database evolve together seamlessly. Often, schema changes require special handling because they can impact live operations. To mitigate these risks, planned downtime or maintenance windows are sometimes necessary. However, minimizing downtime is a priority, so strategies to achieve this must be carefully implemented.
Effectively managing schema changes in Database DevOps necessitates careful planning, robust tooling, automated testing, and strong collaboration between teams. By addressing these challenges, organizations can ensure the integrity and performance of their database systems.

3.5. Version Control for Database Code

Version control for database code is a crucial aspect of Database DevOps. While tools like Git are highly effective for managing application code, database scripts and changes require additional considerations to ensure data integrity and consistency. One of the main challenges is versioning database migrations. These migrations need to be carefully tracked and applied in the correct sequence. Implementing version control for database code involves creating migration scripts that can be executed incrementally. These scripts update the database schema and data state while preserving existing data, minimizing the risk of data loss or corruption. Developers and database administrators (DBAs) need to work together to plan and review database changes, ensuring that they align with overall application updates.

3.6. Automation and its Challenges

Automation is crucial for managing the complexities of database deployments in DevOps. However, it comes with its own set of challenges. Automating database deployments requires robust tooling capable of handling schema migrations, data transformations, and rollback procedures effectively. This ensures that changes are applied consistently and accurately across different environments. One major challenge is integrating automated tests into the pipeline. These tests validate database changes before they are deployed, checking for potential errors, compatibility issues, and performance impacts. However, creating and maintaining these tests requires significant effort and expertise. Automating schema migrations is another challenge which involves writing scripts that can update the database schema incrementally without causing data loss or corruption.

3.7. Collaboration Between Developers and DBAs

Collaboration between developers and database administrators (DBAs) is crucial. DevOps promotes a culture of shared responsibility, but database management has traditionally been siloed. Bridging this gap requires fostering communication and collaboration, ensuring that both developers and DBAs are aligned with the DevOps goals. This often involves training team members in DevOps tools and practices, as well as establishing clear processes to manage database changes.

3.8. Monitoring and Observability

Monitoring and observability are crucial for effectively managing the data state complexities. Monitoring is often lacking in production systems, especially regarding changes to underlying

structures. This lack of monitoring forces teams to be reactive, discovering problems only after changes have been made. Implementing robust monitoring practices is essential for proactive database management. Continuous monitoring tools such as Prometheus and Grafana provide real-time insights into database performance, helping teams detect issues and resolve them quickly. Logging frameworks like ELK Stack (Elasticsearch, Logstash, Kibana) can be used to detect changes and analyze the impact of database updates, to ensure that any anomalies are quickly identified and resolved.

3.9. Security Considerations

Security is a key aspect of DevOps in database management, given that databases often store sensitive information, making them prime targets for attacks. Embedding security practices into the DevOps pipeline, known as DevSecOps, helps ensure data integrity and privacy. This integration involves several key practices such as automated security scans, access controls and encryption mechanisms.

Integrating these security measures into the DevOps pipeline helps organizations to protect their databases against vulnerabilities and unauthorized access, thereby ensuring integrity and confidentiality of their data throughout the development and deployment lifecycle. This proactive approach is essential for maintaining a secure database environment in the face of evolving security threats.

3.10. Other challenges

Implementing DevOps for database management faces several other additional challenges. Scaling DevOps for large, distributed databases introduces complexity, requiring coordinated updates to maintain consistency across multiple nodes or regions. High transaction volumes can negatively impact deployment processes, demanding careful planning to accommodate these changes. Additionally, managing the lifecycle of database environments—development, testing, staging, and production—requires consistent configurations and data states. Ensuring that each environment mirrors production closely is essential for smooth transitions and reliable operations.

4. DEVOPS VS.CONTEXT APPROACHES:ENHANCING DATABASE MANAGEMENT

4.1. DevOps Approach

A core feature of the DevOps approachis simplifying the process of implementing and deploying database changes, thereby reducing the time and effort involved. It also prioritizes automating database deployments, covering tasks such as schema modifications, data migrations, and rollbacks. Furthermore, it ensures that updates to applications and databases remain synchronized, eliminating discrepancies that could result in errors.

However, this approach comes with its challenges. Handling the complexities of data states across various environments, including development, testing, and production, is particularly demanding. Synchronizing application updates with database changes can also be intricate and prone to errors. Additionally, implementing and maintaining automated processes for database management can prove challenging, especially in the context of large or highly complex systems.

4.2. Context Approach

The Context approach highlights the significance of adapting DBA tasks to their specific contexts and fostering incremental knowledge growth. This methodology provides practical strategies that enable smarter and more adaptive database management practices [18]. The main feature of this approach is the use of Contextual Graphs, which capture and contextualize wellestablished procedures, empowering DBAs to tailor their actions to suit unique situations [19][20]. Additionally, it promotes Incremental Knowledge Acquisition, encouraging continuous learning and knowledge sharing among DBAs, allowing them to progressively enhance their expertise. Another main aspect is the focus on collaboration, which enhances teamwork among various stakeholders in database management, fostering a more unified and informed decision-making process.

The context approach brings three notable advantages. First, through Contextual Graphs, DBAs can boost their efficiency and effectiveness by adapting proven procedures to the nuances of their specific contexts. Second, it facilitates Proactive Management, enabling DBAs to anticipate and resolve potential issues before they escalate into critical problems. Lastly, it fosters stronger Collaboration and Communication among DBAs, developers, and other stakeholders, resulting in improved outcomes and a more unified approach to database management.

4.3. Comparing the Use of Devops and Context Approaches to Enhance Database Management

The DevOps approach and the Context approach offer two distinct paradigms for database management, each addressing specific organizational needs (see Table 1). The DevOps approach is centered on integrating DevOps principles into database management processes, emphasizing automation and synchronization to simplify deployments and streamline workflows. By relying on automation tools, DevOps facilitates faster delivery, improved collaboration, and efficient workflows, making it particularly suitable for environments that demand high-speed operations. However, it also encounters challenges, including managing data state complexities, ensuring synchronization across various environments, and addressing the complexities of maintaining automated processes. While this approach is highly processdriven and reactive to issues, it focuses on standardization and creating uniform workflows.

In contrast, the Context approach highlights the importance of adapting database administration tasks to specific organizational needs and contexts. It promotes incremental knowledge acquisition through tools such as Contextual Graphs, which enable DBAs to tailor wellestablished procedures to unique situations. This approach emphasizes learning and collaboration, fostering an environment where knowledge is continuously shared and refined. By emphasizing context-specific adaptations, the Context approach enables DBAs to adopt a proactive stance, anticipating and resolving potential issues before they arise. It also fosters greater collaboration among all database stakeholders, leading to a more unified and informed approach to database management.

The DevOps approach benefits from its strong emphasis on speed and process efficiency, whereas the Context approach focuses on flexibility, adaptability, and improving DBA effectiveness through contextual learning. While DevOps relies heavily on automation, the Context approach is more adaptive, allowing for adjustments to specific scenarios and promoting a greater understanding of the organization’s unique requirements. Each approach has its strengths and challenges. DevOps is particularly effective for environments prioritizing speed and uniformity, while the Context approach is better suited for scenarios requiring adaptability and tailored solutions. Both methodologies offer valuable perspectives, and organizations can

benefit from selecting or combining elements from each approach to optimize their database management strategies.

Table 1. Contrasting the use of DevOps and Context approaches for Database Management.

Aspect	DevOps Approach	Context Approach
Focus	Integration of DevOps principles	Contextualizing DBA tasks and promoting knowledge acquisition
Key Tools	Automation, synchronization tools	Contextual Graphs
Primary Benefits	Simplified deployments, faster delivery, improved workflows	Improved DBA efficiency, proactive management, enhanced collaboration
Main Challenges	Managing data state complexities, ensuring sync, automation	Tailoring procedures to contexts, fostering continuous learning
Approach to Learning	Emphasizes automation and process standardization	Emphasizes incremental knowledge acquisition and context-specific learning
Collaboration	Focus on integrating teams through DevOps	Focus on enhancing collaboration among all database stakeholders
Flexibility	Automation-driven processes	Adaptive, context-sensitive processes
Management Style	Reactive to issues, process-driven	Proactive, context-aware

5. CONCLUSIONS

In conclusion, integrating DevOps practices into database management requires addressing unique challenges to fully realize the benefits of agile development and continuous delivery. The rapid, iterative nature of DevOps often clashes with traditional database methodologies, causing inefficiencies and bottlenecks. This research highlights key challenges, such as managing data state complexities, synchronizing application and database changes, and automating deployments. Furthermore, it highlights the need for a cultural shift to close the gap between development and operations teams, stressing the importance of collaboration and communication. By overcoming these obstacles, organizations can seamlessly incorporate database management into DevOps workflows, achieving faster delivery cycles, enhanced quality, and greater operational efficiency. The paper concludes by comparing the DevOps approach with the Context approach for improving database administration procedures, showcasing how each method contributes to more effective and adaptive database management practices. Through these insights, organizations can better navigate the integration of DevOps principles with their database management strategies, ultimately driving innovation and success

REFERENCES

[1] Ansible Collaborative, https://www.redhat.com/en/ansible-collaborative, accessed on November 2024.
[2] Apache JMeter, https://jmeter.apache.org/, accessed on October 2024.
[3] Atlassian, https://jira.atlassian.com/, accessed on October 2024.
[4] E. Derby, D. Larsen, K. Schwaber, Agile Retrospectives: Making Good Teams Great, Pragmatic Bookshelf, 26 juil. 2006 - 200 pages.
[5] GitLab, Get started with GitLab CI/CD, https://docs.gitlab.com/ee/ci/, accessed on November 2024.
[6] Grafana, https://grafana.com/, accessed on October 2024.
[7] J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation, Addison-Wesley, 2010.
[8] Jenkins, https://www.jenkins.io/, accessed on October 2024.
[9] JUnit 5, https://junit.org/junit5/, accessed on October 2024.
[10] G. Kim, P. Debois, J. Willis, J. Humble, Foreword by J. Allspaw, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations, IT Revolution, 6 oct. 2016 - 480 pages.

[11] Liquibase, Guide to Database DevOps, https://www.liquibase.com/resources/guides/database-devops,, accessed on September 2024.
[12] Microsoft, Azure DevOps, https://azure.microsoft.com/en-us/products/devops/, accessed on October 2024.
[13] Microsoft, Get ready for the future of work with Microsoft Teams, https://www.microsoft.com/en-us/microsoft-teams/group-chat-software, accessed on October 2024.
[14] NUnit, https://nunit.org/, accessed on October 2024.
[15] Prometheus, https://prometheus.io/, accessed on October 2024.
[16] Selenium, https://www.selenium.dev/, accessed on October 2024.
[17] Stackoverflow, Developer Survey Results 2019, https://survey.stackoverflow.co/2019#technology,accessed on October 2024.
[18] H. Tahir, Supporting the contextualization of database administration, PhD Thesis, Sorbonne University, Paris, November, 2013.
[19] H. Tahir, P. Brézillon, Improvement of database administration by procedure contextualization, HCP-2011. Workshop on Human Centered Processes. Proceedings of HCP 2011 - Fourth Workshop on Human Centered Processes. Genoa (Italy) February 10-11, 2011.
[20] H. Tahir, P. Brézillon, Procedure contextualization for collaborative database administration. Proceedings of the 15th International Conference on Computer Supported Cooperative Work in Design, Lausanne, Switzerland, June 8-10, 2011.
[21] Terraform, https://www.terraform.io/, accessed on November 2024.