DevOps Practices and Configuration Management:
A Case Study on the Royal Bank of Canada

Ummen Nasser, Saad Umar, Paris Wang, Tommy Wang
University of Toronto, Faculty of Information
February 6, 2024

Executive Summary

The Royal Bank of Canada (RBC) has embraced DevOps in collaboration with IBM, aiming for infrastructure modernization and streamlined development processes. This case study provides an overview of RBC's DevOps adoption, emphasizing key practices in source code management, release management, and environment configuration. RBC utilizes a diverse toolchain, including Confluence, Jira, GitHub, and Jenkins, to enhance collaboration and efficiency throughout the software delivery lifecycle. The organization's primary objective is agility, aligning with evolving customer demands in the post-pandemic era. Despite benefits, challenges like a potential shortage of mainframe developers and Git adoption issues among developers are recognized. Release Management practices ensure stability, while the paper explores alternatives for environment configuration, such as external configuration files and environment variables, to enhance security and flexibility. In conclusion, RBC's DevOps journey showcases a commitment to innovation and customer satisfaction, navigating challenges with strategic solutions.

Table of Contents

Introduction

With a long history spanning 156 years, the Royal Bank of Canada (RBC) stands as a testament to resilience and adaptability in the face of evolving technological landscapes. While embracing modernization, RBC's reliance on mainframe technology for critical operations presents a need to find a balance between tradition and innovation (Chan, 2020). In response to the challenges posed by the digital era, the company has embarked on a comprehensive transformation journey, aiming to update infrastructure and streamline development processes. To achieve this, RBC has been collaborating with IBM since 2018 and has adopted DevOps practices aimed at automating processes and enhancing their responsiveness to the evolving financial landscape (Chan, 2020). Utilizing RBC as a subject of analysis, this paper examines the key aspects of their DevOps practices. It explores the landscape of DevOps adoption, source code management practices, release management, and environment configuration. By delving into each aspect, we aim to provide a holistic understanding of how RBC leverages DevOps tools and methodologies to not only meet their business objectives but also navigate the intricate challenges inherent in this transformative process.

DevOps Overview

This section examines RBC's DevOps practice including the tools, business objectives, benefits, and challenges in the process. RBC's DevOps practice employs a comprehensive toolchain across various software delivery phases, including planning, development, building, testing, deployment, and feedback (Figure 1). Each tool is crucial in enhancing efficiency, collaboration, and overall software quality. Due to its complexity, we will only focus on a few tools used in their process. To start with, in the planning phase, Confluence takes the lead as a collaborative platform for ideation and effective documentation. This step is followed by Jira for agile project management, enhancing collaboration and adaptation.

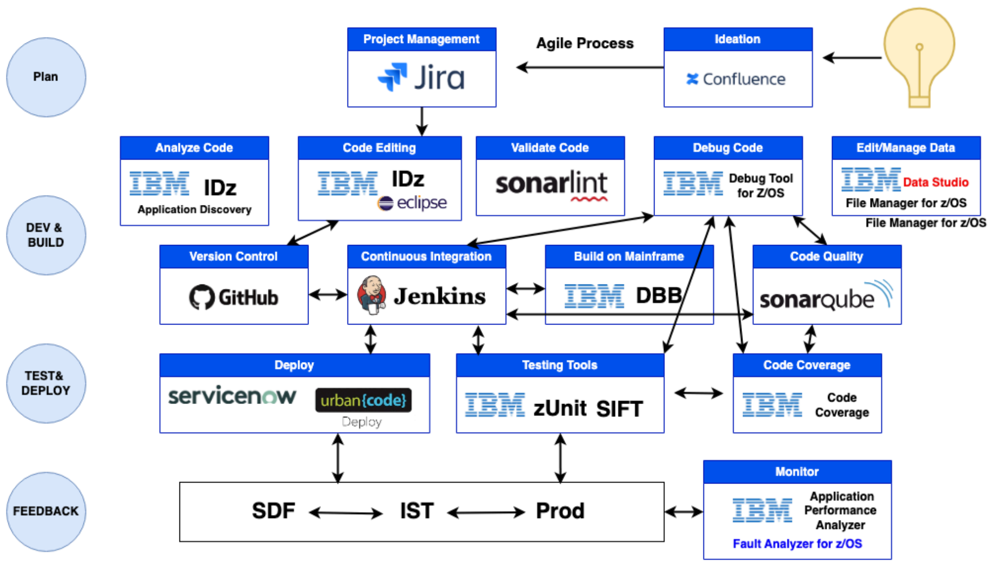


Figure 1. RBC DevOps Structure (Hanna, 2023)

During the development and build phase, IBM Developer for z/OS (IBM IDz) serves as an integrated development environment (IDE) designed for mainframe application development, which plays the role of code analysis and editing (IBM z/OS, n.d.). Additionally, both IBM IDz and Eclipse are pivotal in code editing and provide developers with versatile environments to create and modify code efficiently. SonarLint is integrated into the process to validate code, giving developers real-time feedback and ensuring adherence to code quality standards. The debugging process is streamlined with the use of IBM Debug Tool for z/OS, allowing for efficient identification and resolution of bugs within mainframe applications. Source code management is handled by GitHub, which helps facilitate collaboration, version tracking, and code traceability. Furthermore, to promote efficiency among multiple developers, Continuous Integration (CI) is orchestrated through Jenkins which automates the integration and testing of code changes. For RBC's legacy system, IBM Dependency Based Build (DBB) streamlines z/OS application development in a modern DevOps pipeline by automating compilation, testing, and deployment, enhancing efficiency and Git/Jenkins integration (IBM DBB, 2023). Finally, SonarQube continuously inspects and enhances code quality in the development and Continuous Integration and Continuous Deployment Process (CI/CD).

In the test & deploy phase, RBC uses ServiceNow and IBM UrbanCode Deploy to automate application, middleware, and database deployment, streamlining the DevOps pipeline. (IBM Urban code). For comprehensive testing strategies, RBC utilizes IBM z/OS Automated Unit Testing Framework (zUnit), and SIFT to thoroughly examine application functionalities. To assess the extent of code exercised during testing, IBM Code Coverage provides valuable insights into the test coverage quality. After deployment and testing, the feedback phase includes monitoring using tools like IBM Application Performance Analyzer, which quickly identifies bottlenecks and faults for ongoing performance. Additionally, RBC also created their in-house tool, Helios, to respond to the need to automate their originally manual DevOps pipeline. Helios streamlines GitHub integration, allowing users to choose technologies and environments while also automating the setup of essential tools like Jenkins, SonarQube (Hanna, 2023). It offers a user-friendly interface for easy monitoring, approvals, and issue resolution, thus expediting software delivery (Hanna, 2023). These intricate interactions ensure a synchronized and efficient development lifecycle, which promotes collaboration and quality assurance.

RBC's key business objective, emphasized by Naim Kazmi, CIO of Retail and Payments Technology, centers on agility to meet customer demands (Hanna, 2023). Meeting customer demands is crucial for business success, especially in the post-pandemic era with a growing focus on online transactions. RBC can achieve this objective by utilizing DevOps tools mentioned. First, Helios can automate pipeline onboarding and provide a user-friendly interface for developers. This accelerates the onboarding process and enables faster adaptation to changing business needs. Furthermore, IBM zUnit and SIFT tools enhance online transaction satisfaction by ensuring application reliability through comprehensive testing. Jenkins and UrbanCode Deploy accelerate software delivery by automating workflows, ensuring speedy updates and enhancements to meet agility requirements. Moreover, IBM Application Performance Analyzer provides proactive monitoring to ensure that, after releasing new features or enhancements to the end users, any performance issues can be quickly identified and addressed, mitigating disruptions in online transactions and enhancing overall customer experience.

While these DevOps tools offer benefits in achieving RBC's business objectives, there are several challenges that need to be addressed. The development and build phase, which focuses on mainframe development using IBM IDz and IBM Debug Tool, faces a potential shortage of experienced mainframe developers and future talents within RBC (Hanna, 2023). The effectiveness of these tools also depends on the familiarity of developers with mainframe environments, which can have a steeper learning curve. Additionally, the transition to GitHub as the source code management tool has been challenging for some COBOL and JCL developers at RBC (Hanna, 2023), particularly due to GitHub's interface and features, which can be overwhelming for new users. Furthermore, RBC's legacy systems may not be natively compatible with Git, making the integration process complex and requiring careful planning to ensure a smooth transition, considering potential dependencies on older version control systems. Additionally, with large and complex codebases, Git can face performance challenges, reducing efficiency in source code management (Raymond, 2023). Using GitHub as a cloud-based platform also raises concerns about security and compliance, which are particularly significant in the financial industry dealing with sensitive customer data. While GitHub offers private repositories with security features, there is an additional cost associated with them, and given the size of RBC, this cost could be substantial. While DevOps tools hold the potential to advance RBC's business goals, the organization must proactively address the mentioned challenges to ensure their successful implementation and business objective realization.

Adoption of the source code management practices

Source Code Management (SCM), commonly known as a Version Control System, encompasses practices and tools dedicated to overseeing and monitoring modifications in source code, binaries, configuration files, and runtime dependencies. The principles of integrity, robustness, synchronization, linearity, and revision control are paramount in SCM (Majumdar, 2017). This capability of SCM, tracing modifications to specific individuals, allows for collaboration among developers, facilitating project management and maintaining a comprehensive history of their work. SCM tools are also instrumental in handling tags and versions. They allow developers to create distinct versions of their codebase, allowing tracking of changes over time and facilitating rollback to specific points in the development history. Tags, in this context, serve as valuable markers for labeling and referencing specific versions of the code, aiding in the organization and retrieval of crucial points in the project timeline. Moreover, SCM tools extend their utility by integrating features of Continuous Integration/Continuous Deployment (CI/CD). This integration automates testing, integration, and code delivery processes, streamlining the path from development to production. Along with these CI/CD practices, SCM contributes to an efficient software development and deployment lifecycle.

Steps to resolve issues using Source Code Management

In a scenario where users report an issue in the RBC app, the team leader takes initiative by creating a dedicated bug-fixing branch for a new developer at RBC to address the reported problem. The tool used for source code management at RBC is Git. The developer will follow a structured series of steps to effectively resolve the issue:

1. The developer will first clone the repository which holds the source code to their local machine. This will create a local copy of the entire repository with all the branches.
   git clone <repository-url>
2. As a precautionary step, the developer will first run the *git branch* command to check what branch they are currently on. By default, the developer should be on the main branch, to fix the bug, the developer will switch to the bug-fixing branch which was created by the team leader. This can be done using the following command: *git checkout bug-fixing-branch*
3. On the git-fixing branch, the developer will work on fixing the bug and update the necessary files in the working directory.
4. It is likely that there are other developers working on the source code as well. Assuming changes occur in parallel, to have the most latest version of the main branch, it's essential to fetch any changes made by other developers to avoid conflicts during merge. Switch to the main branch and run the following command to pull the latest changes:
   git pull origin main
5. After the bug has been resolved and the changes have been made to the code, the next step is to stage the changes (from the bug fixing branch), this is done by using the command: *git add <filenames-to-be-staged>*. This basically tells git what files need to be updated in the next commit. However, when the git add command is run, it doesn't actually affect the repository in any way until the git commit command is used. The commit is used to actually record the modifications made to the code and a meaningful message is recorded along with the commit for other people to understand what/why modifications were made to the selected code. The commit command is:
   git commit -m "Bug X has been resolved"
6. Next step is to push the changes to the remote repository in order for other team members to collaborate and review the fix. The bug has been fixed on a branch named: *'bug-fixing-branch'*. From this branch, the updated code will be pushed using the following command: *git push origin bug-fixing-branch*
7. Once the changes have been pushed to the remote repository (e.g. github), a pull request is opened. Pull request is a mechanism used in version control systems to propose a change from one branch to another, allowing for other developers in the repository (or developers who are assigned a particular pull request) to review the modifications and once the code has been deemed satisfactory, it is merged with another branch. In this case, the developer at RBC will open a pull request from the *bug-fixing-branch* to the *main-branch*. Other team members will then review the code, and make any suggestions if necessary and once the code is approved, the changes will then be merged into the *main* branch. Once the code has been merged with the main branch, the issue created by the user can be marked as resolved and closed. Creating the pull request and then merging can be done using the user interface of github itself. Using the interface, pull request can be created using the following steps:
   a. Initiate a pull request and select the base branch as the branch in which you want to merge your changes into (most commonly its the main branch).
   b. Provide the pull request details (title, description, etc).
   c. Create the pull request
   d. Once other team members have reviewed the changes to the code and have approved it, the project lead or someone with authorization will merge the changes using the web interface of Github.
   e. Confirm the merge and delete the feature branch if needed.

This structured approach enables collaboration as well as ensures a systematic approach of resolving bugs with efficient integration of changes into the codebase.

Release Management Practices

Release Management is a process of overseeing the planning, scheduling, and controlling of software builds throughout each stage of the development process (*5 Steps to a Successful Release Management Process*, 2020). It typically includes testing and deployment of software releases. RBC developers would need to initially identify the scope of their project, and decide on deadlines, timelines and requirements for the new release which includes fixing bugs and adding new enhancements (*5 Steps to a Successful Release Management Process*, 2020). The deployment process to include new code fixes and enhancements in the next release is as follows:

1. **Branching and Code Fixes:** Developers identify the bug in the released version and create a hotfix branch from the main branch. They work on this branch to fix the bug following steps mentioned above for source code management. This branch can be shared with other developers, making one or more commits (e.g., commit #8) to resolve the issue. Banks and financial institutions (such as RBC) employ rapid hotfixes to swiftly address security vulnerabilities within their trading platforms and financial software. This proactive approach is crucial for preventing unauthorized access and safeguarding customer funds against cyber theft (Kiruthika, 2023).

2. **Development of New Enhancements:** Parallel to the hotfix work, another branch is created for new features and enhancements. The development team works on this branch to implement the enhancements, resulting in a series of commits.

3. **Testing:** Before merging these changes into the main codebase, rigorous testing is conducted within both branches. Developers employ unit tests, integration tests, and other automated testing methods to confirm that the new code does not introduce further issues and meets the expected criteria. Here developers can choose to deploy Release Testing, which is a critical phase in the software development lifecycle that acts as a bridge between development and production. Its purpose is to detect any overlooked defects by rigorously evaluating the software's stability, performance, and functionality against specified requirements. This phase encompasses various types of testing, including functional, performance, compatibility, security, and usability testing (*Complete Release Testing Guide to Become a 10X Engineering Team | Zeet.co*, n.d.). Each component plays a vital role in confirming that the software operates as expected, integrates well with other systems, performs efficiently under different conditions, provides a user-friendly interface, and maintains a high level of security and stability. The only drawback of release testing is time.

4. **Code Approval and Integration:** The changes in both branches undergo peer review. Upon approval, any changes from the main branch are pulled and merged with the hotfix branch to create a new version, which is then merged back into the main branch for release, resulting in a new commit on the main branch (Figure 2).

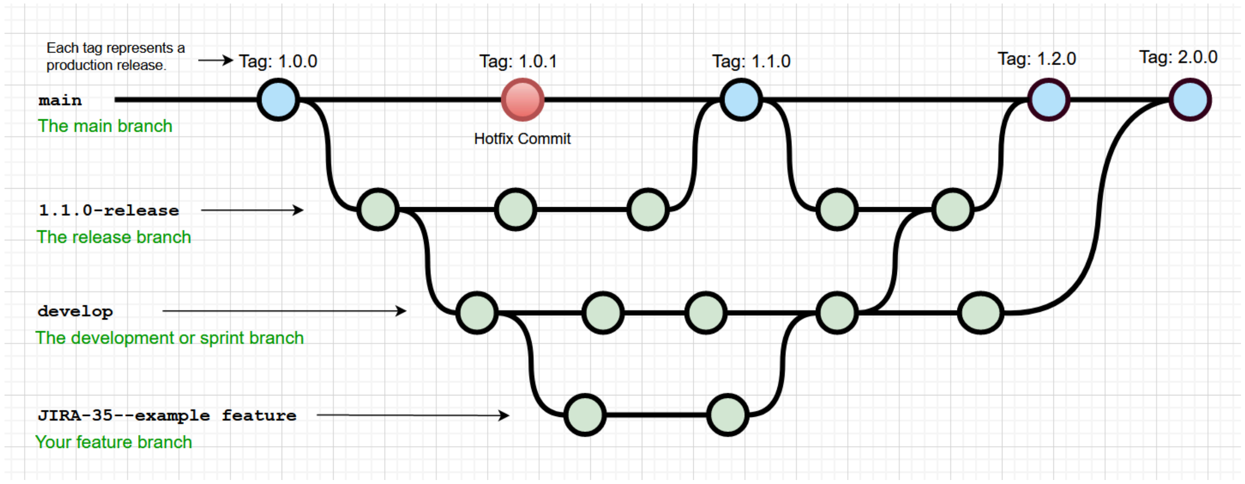


Figure 2. Branch Management Diagram (Braun, n.d.)

5. **Testing Post-Integration:** The integrated code is tested to ensure that the new enhancements work correctly with the hotfixes. Regression testing is performed to ensure that existing functionality is not adversely affected. Release Management teams can also perform Smoke Tests. The test is successful if they can execute at least one transaction to verify that the release was completed and that the system is up and running(*INF1005_1006_W3.pdf: INF1005H S LEC0107 20241:Information Workshop I*, n.d.).
6. **Release Preparation:** As the release nears completion, the integrated code within the main branch is locked down, signalling the end of the development phase for this release. The software version is incremented, and the release is appropriately tagged in the version control system to mark this milestone. The release package is then prepared for deployment, with the operations team conducting a series of production readiness checks to confirm the environment is primed for the new release.
7. **Deployment:** The release package is prepared, and deployment to production is planned. The operations team ensures that all production readiness checks are completed.
8. **Monitoring and Post-Release Support:** After deployment, the application is monitored for any unforeseen issues. A support plan is in place to address any immediate issues that may arise after the release.

Artifacts Needed for Production Deployment in Final Package Release

In the case of RBC, the final package prepared for deployment to the production environment could include the following artifacts:

1. **Application Code:** The latest version of the application code that includes both the hotfixes and new enhancements.
2. **Executable Files:** Compiled binaries or executable files that are ready to run in the production environment.
3. **Configuration Files:** Files containing environment-specific settings that tell the application how to operate in the production environment.

4. **Database Scripts:** Scripts to update the database schema and data transformations required for the new release.
5. **Deployment Scripts:** Scripts or automation tools that facilitate the deployment of the application to production servers.
6. **Documentation:** Release notes detailing the changes, bug fixes, and enhancements included in the release and installation and configuration instructions for the deployment team.
7. **Testing Artifacts:** Test plans, test cases, and test results demonstrating the testing that was performed before release.
8. **Rollback Plan:** A contingency plan with steps to revert to a previous version in case the new release encounters critical issues in production.

By following these steps and ensuring that these artifacts are included, RBC would be able to execute a structured release process that includes both code fixes and new enhancements while maintaining the stability and reliability of their production environment.

Environment Configuration

Given the interoperability of RBC's DevOps tools, proper environment configurations are crucial for efficient integrations between software components during development, testing, and deployment (IBM, 2021). An example of an information system containing environment configurations is their online banking system, which has allowed RBC to consistently deliver innovative digital experiences to clients. The online banking system is a complex network spanning RBC's diverse lines of business such as personal and commercial banking and wealth management, however, this case study will focus primarily on the RBC mobile app, which gives users access to a variety of personal banking services including checking account balances, depositing cheques, and transferring money. The development and production environments for the RBC mobile app are dynamic and interoperable, involving multiple tools as previously mentioned in RBC's DevOps Overview. The application system needs to maintain high levels of functionality and security. To study environment configuration of RBC's online banking app, it is important to identify some configuration items as listed below:

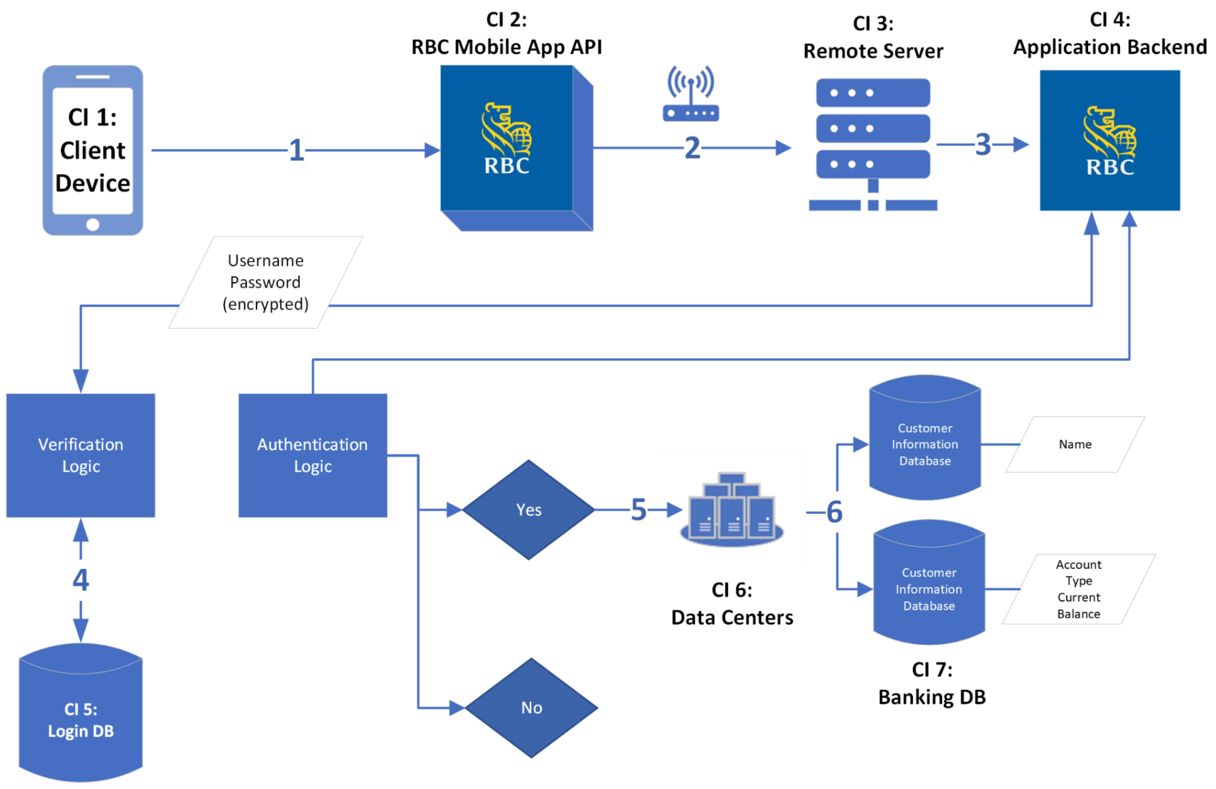
The RBC mobile app first prompts users to log in using a username and password. This process requires: 1. Client device configuration to sync device settings such as notification and language preferences; 2. API configuration to process client requests and send to remote servers; 3. Backend configuration to receive client requests; 4. Database configuration to access a login database or a multi-factor authentication server, which is configured to communicate back to the server; 5. Backend configuration to RBC's datacenters as RBC uses a hybrid infrastructure combining cloud and on-premise data services (Microsoft, 2021); and 6. Banking database configuration to retrieve client identification and account information, such as name, balances, transactions and account updates, which gets sent back to the server and displayed in the app through API (Appendix A). This list, while not exhaustive, provides an example of how critical configuration management is in creating a dynamic and secure application system.

Hardcoding configurations in the RBC mobile application could limit flexibility and scalability, making updates and customization challenging. Hardcoded configuration settings cannot adapt to different environments during development, testing, and production without manual changes, increasing friction and risk of errors throughout RBC's integrated DevOps lifecycle. In addition, hardcoding configurations risks exposing sensitive information, which leads to security vulnerabilities. The newly released version of the RBC app, 6.37 Version, allows Dominion Securities, PH&N and Royal Trust clients who have a linked account to view their balances in Personal Banking. This enhancement would require new or updated API endpoint configurations to fetch account balances for Dominion Securities, PH&N, and Royal Trust clients, as well as changes to authenticate and authorize these clients to ensure secure access to their linked accounts. When hardcoded, any changes to API endpoints, authentication mechanisms, or client identification logic in the process would require a new app release, making the system inflexible and hard to maintain. More importantly, the updated version would involve directly embedding URLs and possibly client identifiers or passkeys within the codebase. Embedding sensitive information into the source code can pose significant security risks if the codebase is exposed. which would involve directly embedding URLs and possibly client identification or passwords if RBC developers were to hardcode this.

One alternative for managing system configuration for the RBC mobile app is to use external configuration files. The external configuration files might include API endpoints, authentication keys, and client identification logic. The app can load these external configuration files during startup or on-demand to fetch configuration updates from a server and apply changes dynamically without requiring constant app updates. This allows configurations to be changed without the need to modify the source code, therefore facilitating easier updates and maintenance for the RBC app, and is also beneficial for RBC's DevOps workflow, as it supports the development, testing, and production environments by maintaining separate configuration files for different environments. However, this configuration system requires secure management of the external files to prevent unauthorized access. In addition, it relies on the correct deployment of these external files alongside the application in every environment. The wrong pairing could mean significant application downtime.

Another alternative for managing system configuration is to environment variables. These variables are set in the operating environment where the app or its server components run. This could be on the mobile device's OS for client-side configurations or on the server hosting the app's backend services. The app's source code would be written to fetch these environment variables at runtime. Since the app reads these variables at runtime, changing an environment variable alters the app's behaviour without the need to change the app's code, which allows for dynamic configuration changes. This offers high flexibility for different deployment scenarios, requires no internal selection of configuration files, and keeps sensitive data out of the codebase and version control system at the same time. However, with many environment variables, managing them can become complex, especially across multiple deployment environments. While they simplified the process of scaling the app across environments, developers and configuration managers at RBC must take preventative measures to ensure environment variables are consistently set across DevOps environments to avoid unexpected behaviours.

Appendix



Appendix A. RBC Mobile App Login Environment Configurations

References

Braun, B. (n.d.) *Example Git Branching Diagram*. (n.d.). Gist. https://gist.github.com/bryanbraun/8c93e154a93a08794291df1fcdce6918

Chan, R. (2020, July 5). *156-year-old bank RBC still runs on mainframes, but it's working with IBM on tech updates that have helped speed up its development process by 30%.* Business Insider. https://www.businessinsider.com/rbc-ibm-mainframe-banking-systems-2020-6 *Complete release testing guide to become a 10X engineering team | Zeet.co*. (n.d.). https://zeet.co/blog/release-testing

Hanna, S. (2023, December 27). *A standardized DevOps pipeline at RBC improves the developer experience and drives Global Business Value*. IBM Community. https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/sherri-hanna1/2023/12/27/rbc-standard-devops-pipeline

IBM DBB. (2023). *IBM Dependency Based Build overview*. IBM dependency based Build Overview. https://www.ibm.com/docs/en/dbb/2.0?topic=dependency-based-build-overview

IBM documentation. (2023, December 12). https://www.ibm.com/docs/en/urbancode-build/7.0.0?topic=build-concepts

IBM documentation. (2021, March 8). https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/lifecycle-management/6.0?topic=management-configuration-concepts-capabilities

IBM z/OS. (n.d.). *IBM developer for z/OS*. IBM. https://www.ibm.com/products/developer-for-zos

Inedo. *Hotfix Branch*. (n.d.). https://docs.inedo.com/docs/buildmaster-git-hotfix-branches

Kiruthika, D. (2023, December 27). *What is a Hotfix? | Benefits , Challenges & How to Test?*. Testsigma Blog. https://testsigma.com/blog/what-is-hotfix/

Lange, K. (n.d.). *Release Management in DevOps*. BMC Blogs. https://www.bmc.com/blogs/devops-release-management/

Loeber, Z. (2020, May 20). Understanding DevOps Artifacts and How to Use Them. SPR. https://www.spr.com/understanding-devops-artifacts-and-how-to-use-them/

Majumdar, R., et al. (2017). "Source code management using version control system," *Institute of Electrical and Electronics Engineers*, pp. 278-281, doi: https://ieeexplore.ieee.org/abstract/document/8342438

Marmelab. (n.d.). Configurable Artifacts: Deploy Code Like a Pro. https://marmelab.com/blog/2017/09/12/configurable-artifacts-deploy-code-like-a-pro.html

Microsoft. (2021) *Royal Bank of Canada speeds innovation on-premises with DBaaS based on Azure Arc-enabled data services*. (n.d.). Microsoft Customers Stories. https://customers.microsoft.com/en-us/story/1431733508110961747-royal-bank-canada-speeds-innovation-on-premises-dbaas-based-azure-arc-enabled-data-services

Microsoft. (2022, August 23). Understand releases and deployments in Azure Pipelines. Azure Pipelines | Microsoft Learn. https://learn.microsoft.com/en-us/azure/devops/pipelines/release/releases?view=azure-devops

Raymond, D. (2023, November 26). *Top 10 cons & disadvantages of using github*. Articles for Project Managers - The Project Management Network. https://projectmanagers.net/top-10-cons-disadvantages-of-using-github/#:~:text=Complexity%20and%20Repository%20Size%20Management,reduced%20efficiency%20in%20code%20management.