Ronan Smyth - STUDENT

L00176857  Computer Science

Devops Lab folio 2

Table of Contents

[1. DevOps 32675 3](#_Toc185778277)

[1.1. DevOps and Agile 3](#_Toc185778278)

[1.2. DevOps Vs Agile 4](#_Toc185778281)

[1.2.1. Similarities 4](#_Toc185778282)

[1.2.2. Differences 4](#_Toc185778283)

[1.3. Benefits of DevOps in Agile Environments 4](#_Toc185778284)

[2. Code 5](#_Toc185778285)

[2.1. Actions 5](#_Toc185778286)

[2.2. Architecture 5](#_Toc185778290)

[2.3. Quality Attributes 6](#_Toc185778291)

[2.4. Resilience 6](#_Toc185778292)

[2.5. Reliability 6](#_Toc185778293)

[2.6. Security 6](#_Toc185778294)

[3. Automated Pipeline 7](#_Toc185778295)

[3.1. Strengths 7](#_Toc185778299)

[3.2. Gaps in the Current Pipeline 7](#_Toc185778304)

[3. Deployment Strategy 8](#_Toc185778307)

[3.1. CI/CD/CO 8](#_Toc185778308)

[3.2. Difference Between Continuous Delivery and Deployment 9](#_Toc185778309)

[3.3. Packaging 10](#_Toc185778310)

[3.4. Why Azure/AWS instead of GitHub in Industry? 10](#_Toc185778311)

[4. Deployment Strategy 11](#_Toc185778315)

[4.1. Blue-Green Strategy 11](#_Toc185778316)

[4.1.1. Process 11](#_Toc185778322)

[4.1.2. Continuous Deployment 13](#_Toc185778323)

[4.1.3. Shortcomings of Blue-Green Deployment 13](#_Toc185778324)

[5. Conclusion 13](#_Toc185778325)

[6. References 15](#_Toc185778326)

Table of Figures

[Figure 1‑1: DevOps SDLC. 2](#_Toc185778210)

[Figure 2‑1: Structure of repository. 4](#_Toc185778211)

[Figure 2‑2: GitHub security overview. 6](#_Toc185778212)

[Figure 3‑1: Code snippet of JUnit testing automation workflow. 8](#_Toc185778213)

[Figure 3‑2: CloudWatch monitoring tool. 9](#_Toc185778214)

[Figure 3‑3: Automate workflow to package code for deployment. 10](#_Toc185778215)

[Figure 3‑4: Part of the workflow which creates the .jar file. 10](#_Toc185778216)

[Figure 4‑1: Workflow for automated JUnit tests using Maven. 12](#_Toc185778217)

[Figure 4‑2: Code snippet for Postman API testing. 12](#_Toc185778218)

[Figure 4‑3: S3 buckets created on AWS CloudFormation platform. 13](#_Toc185778219)

[Figure 4‑4: Errors in push and pull requests. 14](#_Toc185778220)

[Figure 4‑5: Request merged before linking issues or assigning attributes. 14](#_Toc185778221)

# DevOps 32675

ISO/IEC/IEEE 32675 (2021), is the international standard which defines the DevOps methodology. DevOps focuses on defining, controlling, and improving Software Development LifeCycle (SDLC) processes employed during development, packaging, and deployment of software. Among other things, DevOps aims to provide increased security and reliability to software.

The standard emphasizes collaboration and communication among stakeholders, including development, operations, and any other relevant key personnel throughout the SDLC in addition to providing requirements and guidance on implementing DevOps principles and practices.

The standard emphasizes a customer-centric approach which prioritises business goals. It addresses a philosophical change in software development often called a "left-shift" mentality, describing how quality, security, and operational considerations are addressed early in the development process.

The standard emphasises the importance of continuous integration, delivery, deployment of software through continuous testing and feedback from operational processes as can be visualised in figure 1-1.

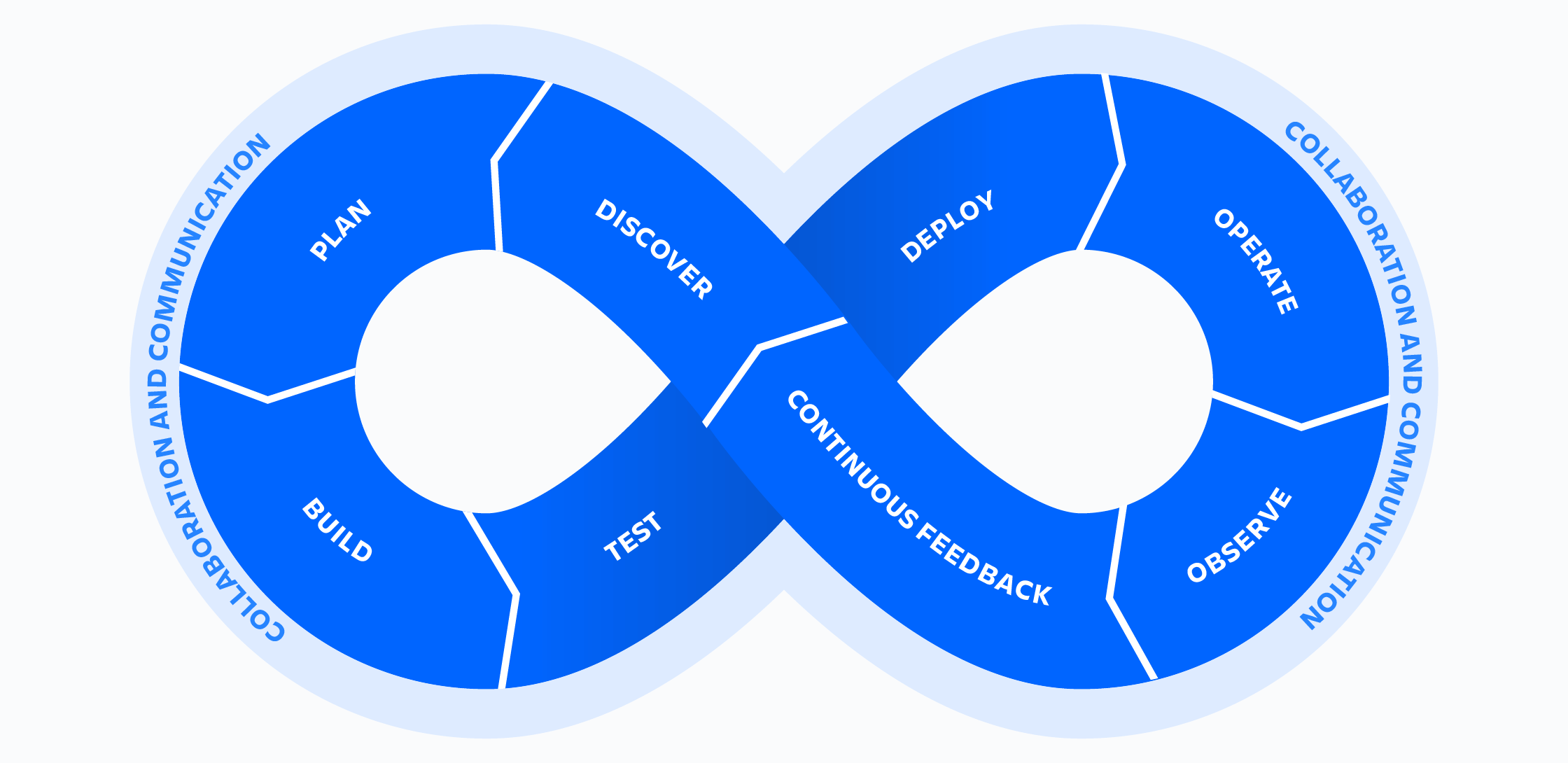


Figure 1‑1: DevOps SDLC.

## DevOps and Agile

According to Hemon et al (2020), DevOps evolved from Agile methods to address a specific bottleneck. While Agile methodologies succeeded in increasing the frequency of software development, poor or difficult communication between development teams and operations teams negatively affected software release timeframes. DevOps emerged to integrate the development and operations functions in order to increase the speed and frequency of software releases. Thus, traditionally separate organizational roles such as developers, architects, scrum masters, product owners, release engineers, and testers on both sides, Dev and Ops, are brought closer through creation of new patterns of collaboration. This new level of interaction between Dev and Ops departs from the traditional collaborations within software development which occur in Agile.



## DevOps Vs Agile

## Similarities

1. **Focus on Collaboration and Efficiency:** Both methodologies prioritize breaking down silos between teams to enhance collaboration and efficiency (AWS, 2024).
2. **Continuous Improvement:** Both Agile and DevOps put emphasis on learning through experience (AWS, 2024).
3. **Customer-Centric Approach:** Agile emphasizes delivering working software in shorter cycles to meet customer demands (1). DevOps builds on Agile’s shorter cycles by striving to ensure that the software is operationally ready and deployed seamlessly to the customer (Hall, 2024).

## Differences

1. **Focus on Collaboration and Efficiency:** Agile emphasizes cross-functional teams for iterative development(AWS, 2024). On the other hand, DevOps integrates development and operations for seamless delivery(Hall, 2024).
2. **Continuous Improvement:** Agile achieves improvement through iterative sprints, reviews and retrospectives (AWS, 2024), while DevOps uses post-deployment monitoring and incident reviews to refine processes (2).
3. **Scope:** Agile focuses on the development phase, iterating through planning, coding, and testing cycles. DevOps extends to include continuous deployment, monitoring, and feedback from operations spanning the entire software lifecycle (AWS, 2024).
4. **Principles:** Agile is built on four values, for instance, individuals over tools and responding to change over following a plan (AWS, 2024). DevOps, in contrast, emphasizes the CALMS framework: Collaboration, Automation, Lean processes, Measurement, and Sharing (Hall, 2024).
5. **Practices:** Agile splits work into smaller increments (sprints) with daily standups and retrospectives (AWS, 2024). DevOps, however, uses Continuous Integration/Continuous Deployment (CI/CD) pipelines, infrastructure as code, and automated monitoring to ensure smooth and frequent releases (Hall, 2024).
6. **Team Integration:** Agile focuses on developers and testers working together (AWS, 2024). DevOps unifies developers, operations, and QA to enable collaboration throughout the delivery pipeline (Hall, 2024).
7. **Automation:** Agile relies on manual processes for sprint planning and progress tracking whereas DevOps heavily incorporates automation for testing, deployment, and monitoring to reduce errors and speed up delivery (AWS, 2024).

## Benefits of DevOps in Agile Environments

Having evolved as the result of observed shortcomings in Agile, DevOps is highly compatible with Agile approaches and results in the following benefits when implemented:

1. Enhanced Collaboration and Communication.
2. Early Focus on Quality and Security ("Left-Shift").
3. Continuous Everything.
4. Improved System Reliability and Stability.
5. Alignment with Business Goals.
6. Increased Automation.

(IEEE Standards and others, 2021)

# Code

A screenshot of a computer

Description automatically generated

Figure 2‑1: Structure of repository.

## Actions

The repository for this development includes key automation-enabling components like the PostmanEcho collection for API testing and pom.xml for Maven-based build management. These elements support DevOps principles by facilitating automated dependency resolution, testing, and consistent builds. The AWS folder provides infrastructure-as-code (IaC) configurations, enabling automated provisioning of EC2 instances and S3 buckets, which can be integrated with CI/CD pipelines. Extending these components with tools like GitHub Actions or Jenkins could further automate testing, deployment, and infrastructure management workflows.



## Architecture

The repository follows a modular architecture with clear separation between application logic (KakaricoRealEstateAgent) and testing (JUnitTest). The KakarikoRealEstateAgent class encapsulates the core domain logic, while the JUnitTest provides unit testing capabilities. CloudFormation templates in the AWS folder enable scalable infrastructure deployment, aligning with a hybrid approach that combines application logic and infrastructure. This structure promotes scalability and ease of integration with the cloud-based services in AWS.

## Quality Attributes

Quality is embedded through comprehensive testing practices and adherence to industry-standard tools like JUnit and Postman. The JUnitTest folder validates core functionalities of the application, while the Postman collection ensures API responses meet functional requirements. Maven's dependency and build management further enhance consistency across environments. The architecture facilitates modular testing, ensuring that changes in one component do not adversely affect others, which improves overall maintainability and performance.

## Resilience

Resilience is achieved through test-driven development (TDD) and the use of infrastructure-as-code in the AWS folder. CloudFormation templates enable repeatable infrastructure provisioning, ensuring rapid recovery in case of failures. The modular code structure and test coverage minimize runtime errors. Adding health checks and retry mechanisms to the EC2 and S3 services could enhance resilience further by enabling graceful degradation during partial failures (AWS, 2024c).

## Reliability

Reliability is implemented through structured unit tests in the JUnitTest/src/test directory and API validation using the Postman collection. The CloudFormation configurations ensure consistent deployment of reliable infrastructure, such as EC2 instances and S3 buckets. Git's version control system enables traceability and rollback capabilities though rollback could also be implemented through s3buckets.

Enhancements such as monitoring tools (e.g., AWS CloudWatch) could provide real-time reliability insights, ensuring proactive issue resolution (AWS, 2024e).

## Security

The repository incorporates a SecurityPolicy.md file, outlining secure development practices. AWS configurations for EC2 and S3 provide a foundation for implementing secure cloud solutions, including identity and access management (IAM). Moreover, the repository automates dependency checking on code via the Dependabot and security key checker.

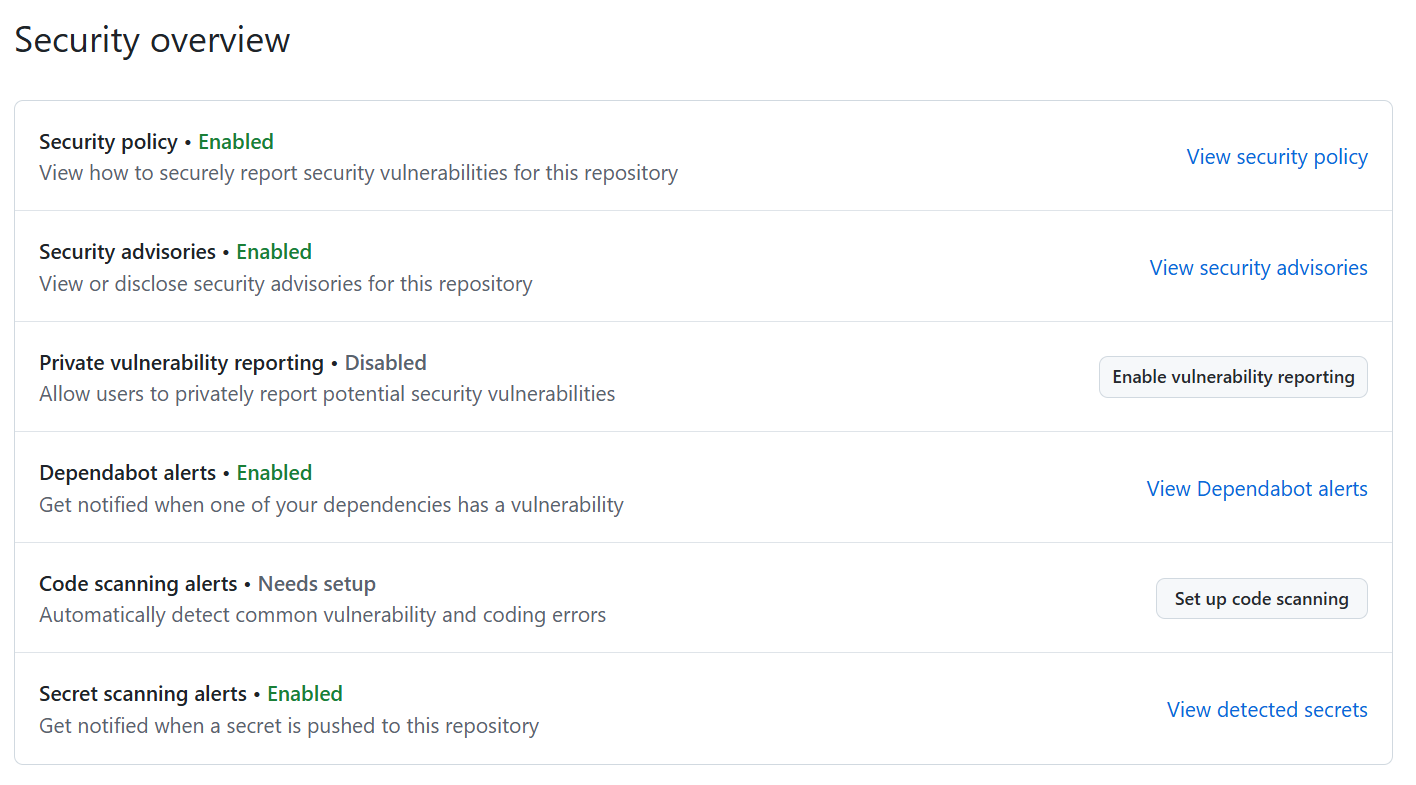


Figure 2‑2: GitHub security overview.

As mentioned in the first lab folio, tools like dependency scanners (e.g., Snyk) and static code analysis (e.g., SonarQube) can be added to augment vulnerability detection.

## Automated Pipeline

A fully automated pipeline is not implemented due to time constraints. Current automation is outlined in the following paragraphs.



## Strengths

1. **Testing Framework:** The repository includes JUnit test cases and a Postman integration for API validation, providing a focus on quality assurance. These tools can be integrated into a CI/CD pipeline to automate testing during every commit or pull request.
2. **Infrastructure as Code (IaC):** The AWS folder contains CloudFormation templates, which are a strong foundation for automating infrastructure provisioning. This aligns with DevOps principles of reproducibility and scalability.
3. **Maven Build Tool:** The pom.xml file provides automated build and dependency management capabilities, ensuring that the project is consistently compiled and tested.

## Gaps in the Current Pipeline

1. **Lack of Explicit CI/CD Integration:** The repository does not currently include any configuration files or scripts (e.g., GitHub Actions .yml files, Jenkinsfile) to automate builds, tests, or deployments. This could be implemented to improve automation.
2. **Postman Integration:** While Postman scripts are included, it has not been integrated into an automated pipeline. Adding Newman (Postman's CLI tool) to run API tests as part of a CI/CD workflow would enhance the automation (Douglas, 2023).
3. **Infrastructure Testing:** Whereas CloudFormation templates are included, automation has not been validated. Tools like AWS CloudFormation Linter (cfn-lint) could be incorporated to ensure the correctness of IaC before deployment (DeJong, 2024).
4. **Monitoring and Feedback:** Monitoring tools have not been integrated with this repository. In theory, monitoring tools would provide feedback from deployments which is crucial for a well-rounded pipeline.

Implementation of Prometheus for end-to-end monitoring would benefit the pipeline by rounding it out. What is more, the Prometheus monitoring tool can be combines wither monitoring tools like Grafana for visualisation and Alertmanager for notifications (BrowserStack, 2024).



# Deployment Strategy

The deployment strategy for this application must align with DevOps best practices, ensuring scalability, reliability, and ease of rollback in case of issues. Bear in mind that Based on your repository's features (Java-based application, AWS CloudFormation templates, and Postman API tests), here’s a recommended deployment strategy:

## CI/CD/CO

**Continuous Integration (CI):** CI is the practice of regularly merging developers' code changes into a shared repository and automatically testing those changes to detect and fix issues early and integrate changes smoothly, without breaking the build or introducing bugs.

This is achieved through the GitHub branching strategy and by using Maven to automate Junit testing.

A screenshot of a computer

Description automatically generated

Figure 3‑1: Code snippet of JUnit testing automation workflow.

**Continuous Delivery (CD):** Continuous Delivery extends CI by ensuring that the codebase is always in a deployable state, with automated workflows that prepare it for release to production via automated packaging and deployment to staging or testing environments

**Continuous Operations (CO):** CO focuses on automating and streamlining post-deployment operations, which include monitoring and feedback, in order to maintain a stable and reliable environment for the deployed application.

Though no monitoring has been applied to the repository, in industry, real-time monitoring and alerting for issues, for example errors or latency, would be integrated using the monitoring tools mentioned or by using CloudWatch from AWS (AWS, 2024e).

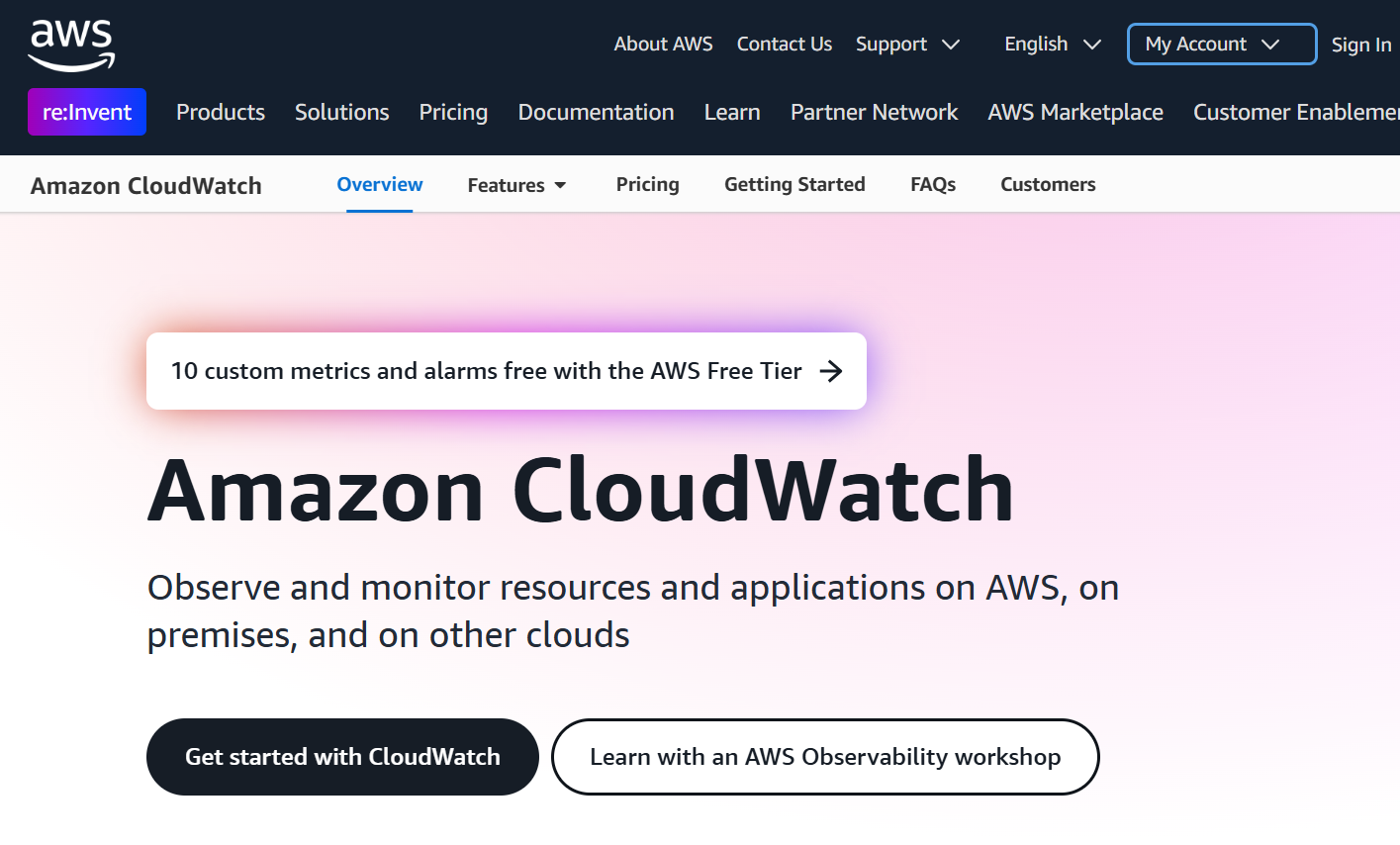


Figure 3‑2: CloudWatch monitoring tool.

Horizontal or vertical scaling, based on demand, can be automated in addition to creation of incident management workflows to resolve problems quickly.

**Continuous Deployment (Another CD):** Automated process of deploying changes to production by verifying intended features and validations to reduce risk.

## Difference Between Continuous Delivery and Deployment

The main difference between continuous delivery and continuous deployment is that continuous delivery stops at the staging or testing environment, while continuous deployment automatically deploys changes all the way to the production environment.

Continuous delivery focuses on ensuring that software is always ready to be deployed, while continuous deployment focuses on actually deploying those changes to production on a regular basis. Continuous deployment requires a high level of automation and confidence in the testing process.

## Packaging

A screenshot of a computer

Description automatically generated

Figure 3‑3: Automate workflow to package code for deployment.

GitHub actions can be used to automate production of a .jar file, since the application is written in Java.

A close-up of a computer screen

Description automatically generated

Figure 3‑4: Part of the workflow which creates the .jar file.

## Why Azure/AWS instead of GitHub in Industry?

While GitHub is an excellent version control tool and provides CI/CD automation via GitHub Actions, cloud platforms like Azure and AWS offer additional features that make them more suitable for deployment in a production-grade industry environment.

1. **Broader Infrastructure and Services:** Infrastructure services, such as, compute, storage, networking in addition to managed services, such as, databases, serverless functions, machine learning.
2. **Scalability and Reliability:** Auto-scaling, load balancing, and global distribution in addition to SLAs for uptime and disaster recovery.
3. **Security and Compliance:** IAM (Identity and Access Management), encryption, and compliance with industry regulations, for instance, GDPR and HIPAA.
4. **DevOps Tool Integration:** Terraform, Jenkins, and Kubernetes for advanced CI/CD workflows and IaC.
5. **Cost Efficiency for Infrastructure:** Price based on usage.

# Deployment Strategy

Deploying an add-on for a video game across the board involves several considerations to ensure smooth delivery, compatibility, and user satisfaction. This chapter contains the chosen strategy for rolling out the deployment.

## Blue-Green Strategy

Simultaneous rollout across all players and servers owing to the fact that all elements must operate on the same version to avoid synchronization issues.

Blue-green deployment also ensures that the new version (Green environment) is fully tested and ready before it replaces the active version (Blue environment) for all players simultaneously.

Since the nature of the product necessitates simultaneous rollout, canary or phased strategies would not be advisable as they may lead to synchronization issues whereby players operating different game versions may not be able to interact.

The Blue-Green strategy provides Zero Downtime as traffic is switched from the old environment to the new one seamlessly.

If issues should arise after deployment, the traffic can be redirected back to the old, blue environment.

The Green environment can be tested thoroughly before making it live. This strategy minimizes the risk of deploying untested code to users.



### Process

#### Pre-Deployment Setup

The blue environment hosting the current live version of the game and the green environment hosting a duplicate of the blue environment, where the new add-on is deployed, will be hosted on technologies offered by AWS, such as, EC2 instances, S3buckets and an Elastic Load Balancer (ELB) to switch traffic between environments.

#### Testing in Green Environment

Automated testing such as unit testing, integration testing, API testing and end-to-end tests are applied using Maven workflow and Postman.

Load tests to simulate high player traffic can also be carried out in addition to QA team involvement to ensure stability and feature correctness.

A screenshot of a computer

Description automatically generated

Figure 4‑1: Workflow for automated JUnit tests using Maven.

A computer code on a white background

Description automatically generated

Figure 4‑2: Code snippet for Postman API testing.

#### Deployment

Once testing confirms the green environment is stable, the ELB will redirect all player traffic to the Green environment. AWS CloudFormation will be used to define and manage blue and green environments consistently.

A screenshot of a computer code

Description automatically generated

Figure 4‑3: S3 buckets created on AWS CloudFormation platform.

#### Rollback Plan

In the event of issues being detected in the green environment the AWS ELB will switch traffic back to the blue environment immediately. Said issues can then be debugged in the green environment without disrupting the live play.

### Continuous Deployment

As deployment takes advantage of so many AWS tools and platforms, monitoring after deployment for infrastructure metrics will be carried out using AWS CloudWatch.

### Shortcomings of Blue-Green Deployment

The Blue-Green strategy, although the best fit for this deployment, is not without drawbacks. The strategy requires infrastructure duplication, that is, double the servers and storage, during the deployment process, which will increase costs. Furthermore, ensuring that databases are backward-compatible or migrated to support the new add-on without impacting gameplay will increase costs, workhours and complexity to the process.

# Conclusion

The current repository in use for this project has strong foundational elements for an automated pipeline. However, it lacks the actual implementation of some necessary CI/CD workflows. By leveraging existing tools like GitHub Actions or Jenkins, you can automate builds, tests, and deployments, significantly enhancing efficiency and aligning with DevOps standards.

Currently, workflows have been set up for Maven unit testing and for packaging before deployment.

The branching strategy, due to the ad hoc nature of information supplied during the DevOps module, was difficult to get right from the beginning. Mistakes were made in the creation of new branches, especially during push and pull requests. Error made were difficult to reverse, hence 2 branches beginning with student number made unintentionally and not removed

A screenshot of a computer

Description automatically generated

Figure 4‑4: Errors in push and pull requests.

This student found operation of GitHub difficult at time, furthermore to the point that errors were difficult to correct, whereby operations carried out in incorrect order were then seemingly impossible to reverse.

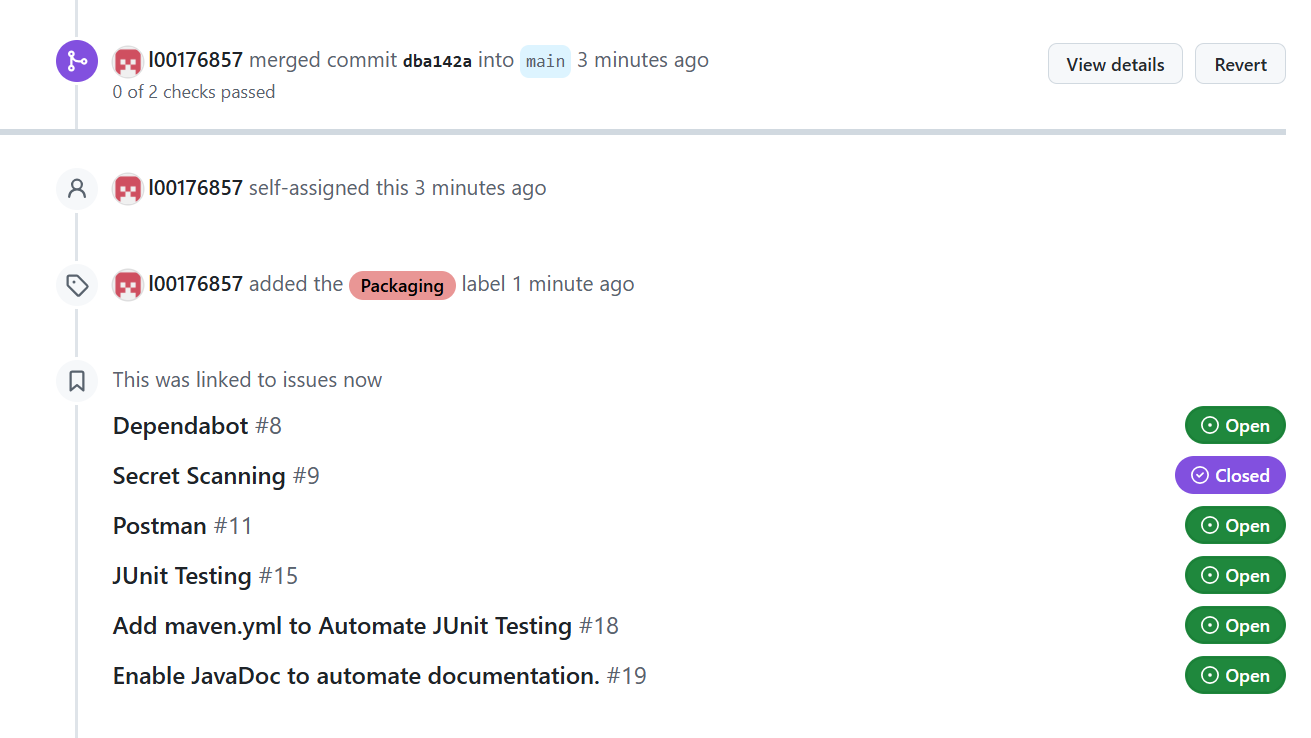


Figure 4‑5: Request merged before linking issues or assigning attributes.

For a game add-on requiring simultaneous rollout, blue-green deployment was selected as the most appropriate strategy to employ. It ensures compatibility across all players, provides zero downtime, and offers a quick rollback mechanism though drawbacks were noted too which include increased costs.

The technologies chosen for this strategy ensure seamless deployment with minimal risk. The combination of IaC tools, CI/CD pipelines and monitoring systems allows for safe and efficient simultaneous rollouts. Time constraints have resulted in not being able to research feature flags although the student is aware of their importance and potential benefit to the pipeline.

# References

1. Atlassian (2024) *What is DevOps?*, *Atlassian*. Available at: <https://www.atlassian.com/devops> (Accessed: 8 December 2024).
2. AWS (2024) *Agile vs DevOps - Difference Between Software Development Practices - AWS*, *Amazon Web Services, Inc.* Available at: <https://aws.amazon.com/compare/the-difference-between-agile-devops/> (Accessed: 20 December 2024).
3. AWS (2024e) *APM Tool - Amazon CloudWatch - AWS*, *Amazon Web Services, Inc.* Available at: <https://aws.amazon.com/cloudwatch/> (Accessed: 20 December 2024).
4. AWS (2024d) *Health checks for instances in an Auto Scaling group - Amazon EC2 Auto Scaling*. Available at: <https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-health-checks.html> (Accessed: 20 December 2024).
5. AWS (2024c) *Implementing health checks*, *Amazon Web Services, Inc.* Available at: <https://aws.amazon.com/builders-library/implementing-health-checks/> (Accessed: 20 December 2024).
6. AWS (2024b) *Security best practices for Amazon S3 - Amazon Simple Storage Service*. Available at: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/security-best-practices.html> (Accessed: 20 December 2024).
7. BrowserStack (2024) *Top 21 Monitoring Tools in DevOps for 2024*, *BrowserStack*. Available at: <https://browserstack.wpengine.com/guide/top-devops-monitoring-tools/> (Accessed: 20 December 2024).
8. DeJong, K. (2024) *AWS CloudFormation Linter (cfn-lint) v1 | AWS DevOps & Developer Productivity Blog*. Available at: <https://aws.amazon.com/blogs/devops/aws-cloudformation-linter-v1/> (Accessed: 20 December 2024).
9. Douglas, I. (2023) *Postman CLI vs. Newman: Choose the Right Tool for You*, *Postman Blog*. Available at: <https://blog.postman.com/postman-cli-vs-newman/> (Accessed: 20 December 2024).
10. Hall, T. (2024) *Agile vs DevOps*, *Atlassian*. Available at: <https://www.atlassian.com/devops/what-is-devops/agile-vs-devops> (Accessed: 20 December 2024).
11. Hemon, A. *et al.* (2020) ‘From agile to DevOps: Smart skills and collaborations’, *Information Systems Frontiers*, 22(4), pp. 927–945.
12. IEEE Standards, I.S. and others (2021) ‘IEEE Standard for DevOps: Building Reliable and Secure Systems Including Application Build, Package, and Deployment: IEEE Standard 2675-2021’.