Continuous Deployment and Testing Strategy for BMI Web Application

Kenn Thomas, Jr.
Net ID: kwt61
GitHub Username: Swagatroni
GitHub Repository Link: https://github.com/Swagatroni/BMI-App
Submission Date: April 7, 2024

Introduction

In the rapidly evolving landscape of software development, the ability to release high-quality, user-centric applications swiftly and efficiently stands as a cornerstone for success. This report outlines the journey and strategic decisions undertaken in extending an existing software system—specifically, a Body Mass Index (BMI) calculation tool—into a web-accessible application. The project's core objective was not only to enhance accessibility and user engagement by transitioning to a web platform but also to establish a robust, continuous deployment pipeline that ensures the delivery of a reliable and high-quality software product.

The report aims to provide insights into the challenges and successes encountered, serving as a comprehensive case study on applying continuous deployment and automated testing strategies in software development. It is a testament to the project's commitment to delivering a tool that not only meets the immediate needs of its users but also adheres to the highest standards of quality and reliability.

Deployment Pipeline

Source Control
Tool: GitHub
Benefits: GitHub is a widely used platform for version control and collaboration, enabling developers to track and manage changes to their codebase efficiently. It supports branch management, pull requests, and code reviews, fostering better code quality and collaboration among team members.
Challenges: New users might struggle with understanding Git concepts like branching, merging, and resolving conflicts. Establishing a consistent workflow, such as Git Flow, requires team agreement and discipline.

Continuous Integration (CI)
Tool: GitHub Actions
Benefits: GitHub Actions facilitates CI by automating the build and test processes every time a commit is made to the repository. This ensures that new changes integrate well with the existing code, helping to catch integration errors quickly and maintaining software quality throughout development.
Challenges: Setting up CI pipelines (e.g., GitHub Actions) requires a good understanding of YAML syntax and the specific platform's configuration options. Initial setup can be time-consuming, especially for complex build and test scripts.

Static Analysis
Tool: Pylint
Benefits: Static analysis tools help in identifying potential errors, code smells, and style issues in the source code without executing it. These tools aid in maintaining a high code quality standard, enforcing coding guidelines, and improving software maintainability.
Challenges: Configuring linters and static analysis tools to match a team's coding standards can be intricate. These tools can generate false positives, requiring time to fine-tune settings or rules. Integrating these tools into the CI pipeline might also require additional scripting.

Automated Unit Tests
Tool: PyTest
Benefits: pytest is a testing framework for Python that allows for the writing of simple and scalable test cases. Automated unit tests validate the correctness of individual units of source code, ensuring that changes to the code do not break existing functionality. This helps in developing a more robust software solution.

Challenges: Writing comprehensive test cases that cover all edge cases can be challenging and time-consuming. It's also crucial to mock external dependencies accurately, which can be complex for beginners.

Automated End-to-End Tests
Tool: Selenium
Benefits: Selenium automates web browsers, enabling developers to simulate user interactions with the web application. Automated end-to-end tests verify the application's behavior from the user's perspective, ensuring that the entire system functions correctly.
Challenges: End-to-end testing frameworks like Selenium require a good deal of setup, especially for complex web applications with many interacting components. These tests can be brittle, breaking with minor UI changes, and might require frequent updates. They also tend to run slower than unit tests, potentially slowing down the CI pipeline.

Automated Deploy to Staging Environment
Tool: Google Cloud Platform (GCP)
Benefits: This platform offers managed services for deploying applications to a staging environment. Automating the deployment to staging allows for the real-world testing of the application in a production-like environment, facilitating the identification and correction of deployment-specific issues before the software reaches actual users.
Challenges: Configuring cloud environments (e.g., AWS, GCP, Heroku) can be complex due to their vast array of services and settings. Automating deployment to these platforms often requires understanding specific CLI tools and APIs. Ensuring consistency between staging and production environments to avoid "it works on my machine" problems is also crucial.

Manual Push to Production
Tool: Manual review process facilitated by GCP
Benefits: A manual push to production, often through a simple command or interface provided by the deployment platform, allows for final human oversight. This step ensures that all automated checks have passed and the application is ready for live users, adding an extra layer of assurance to the deployment process.
Challenges: Human error in the manual review or deployment process can lead to issues in production. There may also be challenges in coordinating among team members, especially in larger teams or those distributed across time zones, to approve and conduct the push to production.

Connect 3rd Party Code Coverage Tool
Tool: Coverage Gutters
Benefits: Coverage Gutters integrates with Visual Studio Code as an extension to track code coverage over time, ensuring that all parts of your code are adequately tested. It provides insights

into which parts of your code are not covered by tests, helping to prevent potential bugs from reaching production. Code coverage tools promote a healthier codebase by encouraging comprehensive testing.
Challenges: Setting up Coverage Gutters may require resolving compatibility issues with test frameworks and coverage formats. Complex projects can necessitate additional configuration for accurate coverage interpretation. Large projects might experience performance impacts in Visual Studio Code due to the extension. Understanding and utilizing coverage data effectively to enhance test comprehensiveness can present a learning curve. Regularly updating coverage reports to maintain current data adds to development workflow overhead.

Each of these steps and tools contributes to a streamlined, efficient, and high-quality software development lifecycle, enabling the rapid and reliable delivery of software products to end users. However, a common challenge is the learning curve associated with each tool and process. Each addition to the pipeline can introduce complexity and require time to set up and maintain. Balancing the depth and breadth of testing and automation with the project timeline and resources is crucial to ensure a smooth and efficient software development lifecycle.

Test Cases for Manual Testing

Test Case 1: Boundary Testing for BMI Classifications
Objective: Verify that the application correctly handles the boundary values for all BMI classifications.

Test Case ID: TC_BMI_01
Test Designed By: Kenn
Test Designed Date: 3-9-24
Test Executed By: Kenn
Test Execution Date: 4-9-24
Test Priority: High

Module: Input Validation
Description: This test verifies the application's response to boundary values of BMI classifications.

Test Steps:
1. Navigate to the BMI calculation code.
2. Pass 18.4 as the *val* to the *classify()* function.
3. Pass 18.5 as the *val* to the *classify()* function.
4. Pass 18.6 as the *val* to the *classify()* function.
5. Repeat steps 2 - 4 for 25, and 30.
    a. Test plus and minus 0.1 for each number.

Expected Results: The application should display the appropriate classification for the given BMI.
Actual Results: The application correctly classified all boundary inputs.
Status: Pass

Test Case 2: Equivalence Partitioning for Height Input
Objective: Ensure the application accurately processes valid height ranges.

Test Case ID: TC_BMI_02
Test Designed By: Kenn
Test Designed Date: 3-9-24
Test Executed By: Kenn
Test Execution Date: 4-9-24
Test Priority: High
Module: Web Interface/Input Validation
Description: This test checks that the application correctly calculates BMI for valid height inputs by dividing the input space into equivalence classes.

Test Steps:
1. Navigate to the BMI calculation page.
2. Enter a valid weight.
3. For height, enter a value that represents a typical valid height (e.g., 5 feet 6 inches).
4. Submit the form.
5. Repeat steps 2 - 4 with invalid inputs. (ie. negative integers, > 12 inches)

Expected Results: The application calculates and displays the BMI based on the provided valid height and weight and asks the user to try again if inputs are invalid, indicating proper handling of height inputs.
Actual Results: The application handled the invalid and valid inputs accordingly.
Status: Pass

Automated Unit Testing

Automated unit testing is a critical component of the development process, especially for applications undergoing continuous integration and deployment. It involves writing tests for the smallest testable parts of an application which are often functions or methods.

Key Aspects of Automated Unit Testing

Tool: PyTest
Isolation: Each test should focus on a single "unit" of code. This ensures that tests are not dependent on each other and can pinpoint exactly where a problem occurs.
Repeatability: Tests should produce the same results every time they are run, assuming no changes have been made to the code.
Automated Execution: Tests should be able to run automatically without manual intervention, which is crucial for integrating with continuous integration (CI) systems.

Implementing Unit Tests for the BMI Application:

Testing *calculate_bmi* Function:

The *calculate_bmi* function takes weight in pounds, and height in feet and inches, then calculates and returns the BMI. Tests should cover various input combinations, including boundary conditions and invalid inputs to ensure the function behaves as expected.

- Normal Inputs: Verify that the function calculates the correct BMI for valid inputs.
- Boundary Conditions: Test for edge cases, such as very low or high height and weight.
- Invalid Inputs: Ensure the function raises an appropriate exception for invalid inputs.
  - (e.g., negative values, zero, or non-numeric types)

Testing *classify* Function:

The *classify* function takes a BMI value and returns a string indicating the BMI category. Tests should verify that it returns the correct classification for a range of BMI values.

- Classification: Ensure there's at least one test for each possible classification.
- Invalid Input: Verify that the function raises an exception for invalid BMI values.
  - (e.g., negative numbers or non-numeric types)

Integration with CI Tools:

Integrating these unit tests into a CI pipeline (e.g., GitHub Actions) ensures that tests are automatically run every time code is pushed to the repository. This helps in identifying and fixing issues early in the development cycle. The CI tool can be configured to run `pytest` and report the results, blocking any pull requests that fail tests from being merged until the issues are resolved.

Automated unit testing, as outlined, not only aids in ensuring code quality and correctness but also facilitates a more efficient and reliable development workflow, particularly in agile and continuous delivery environments.

Tool Description

In the deployment pipeline for your BMI Web App, several tools play crucial roles in ensuring code quality, integration, and deployment efficiency. Here's a brief overview of each tool, its functionality, ease of use, and how it integrates into the pipeline.

GitHub

Purpose and Functionality: GitHub is a cloud-based hosting service that lets you manage Git repositories. It provides a platform for version control and collaboration, allowing multiple contributors to work on projects simultaneously.
Ease of Use and Setup: GitHub is user-friendly, with extensive documentation and community support making it accessible for beginners and professionals alike. Setting up a new repository is straightforward, with GUI and CLI options.
Recommendation: Highly recommended for teams of any size due to its robust features, including issue tracking, code review, and integration with many CI/CD tools.
Integration: Serves as the foundation of the deployment pipeline, where code changes are pushed, reviewed, and merged. It triggers the CI/CD pipeline on actions like push or pull requests.

GitHub Actions

Purpose and Functionality: GitHub Actions enables the automation of workflows directly within your GitHub repository. It can be used for continuous integration (CI) and continuous deployment (CD), running tests, building and pushing images, deploying to various environments, and more.
Ease of Use and Setup: It integrates seamlessly with GitHub repositories, reducing the need for external CI/CD tools. Setting up workflows is done through YAML files, allowing for customizable pipelines.
Recommendation: Recommended for projects hosted on GitHub due to its direct integration and versatility. It's suitable for both simple and complex workflows.
Integration: It's triggered by GitHub events, such as push, pull requests, or manual triggers. It can run tests, deploy to staging, and push to production, making it a central part of the CI/CD pipeline.

Pytest

Purpose and Functionality: Pytest is a testing framework for Python that allows you to write simple and scalable test cases for your applications. It supports fixtures, assets, and plugins for extending its capabilities.
Ease of Use and Setup: Pytest is straightforward to set up and use, especially for those familiar with Python. Its detailed documentation and active community support ease the learning curve.
Recommendation: Recommended for Python-based projects due to its flexibility, powerful features, and ease of integration with other tools and CI/CD pipelines.
Integration: This can be integrated into GitHub Actions workflows to run tests automatically upon code pushes or pull requests, ensuring that only passing code is merged.

Coverage Gutters

Purpose and Functionality: Coverage Gutters is a Visual Studio Code extension designed to visually display code coverage in the editor, highlighting lines of code covered by tests and those that are not.
Ease of Use and Setup: The extension is straightforward to install through the Visual Studio Code marketplace and configure. It works by reading coverage reports and visually annotating the code editor with coverage information. The setup process involves specifying the location of the coverage report files within the extension's settings.
Recommendation: Coverage Gutters is recommended for teams seeking to improve their code quality through comprehensive testing. The ease of identifying uncovered code directly within the code editor streamlines the testing process, making it essential for any team focused on maintaining high code quality.
Integration: It integrates seamlessly into the development workflow by working alongside testing frameworks that generate coverage reports, such as Jest, Mocha, or Istanbul. The extension reads these reports and displays the coverage data directly in the editor.

Selenium

Purpose and Functionality: Selenium is a portable framework for testing web applications. It provides tools for writing functional tests without the need to learn a test scripting language.
Ease of Use and Setup: It has a learning curve, especially for those new to testing or web development. However, extensive documentation and community resources are available.
Recommendation: Recommended for projects requiring thorough end-to-end testing of web interfaces. It's versatile and supports multiple programming languages and browsers.
Integration: Selenium tests can be integrated into CI/CD pipelines using tools like GitHub Actions. They can be set to run automatically on code pushes or periodically, ensuring the application's user-facing components work as intended.

Integration with the Pipeline

Each tool integrates into the deployment pipeline to automate and streamline the process from code commit to deployment. The flow typically begins with a code push to GitHub, triggering GitHub Actions workflows. These workflows can run unit tests with Pytest, end-to-end tests with Selenium, and report coverage statistics with Coverage Gutters. Upon successful completion of tests and checks, the workflow can automatically deploy the application to a staging environment and, with manual approval, to production. This automation ensures a smooth, reliable, and efficient deployment process, minimizing human error and speeding up release cycles.

Cloud Platform Recommendation: Google Cloud Platform (GCP)

Challenges:

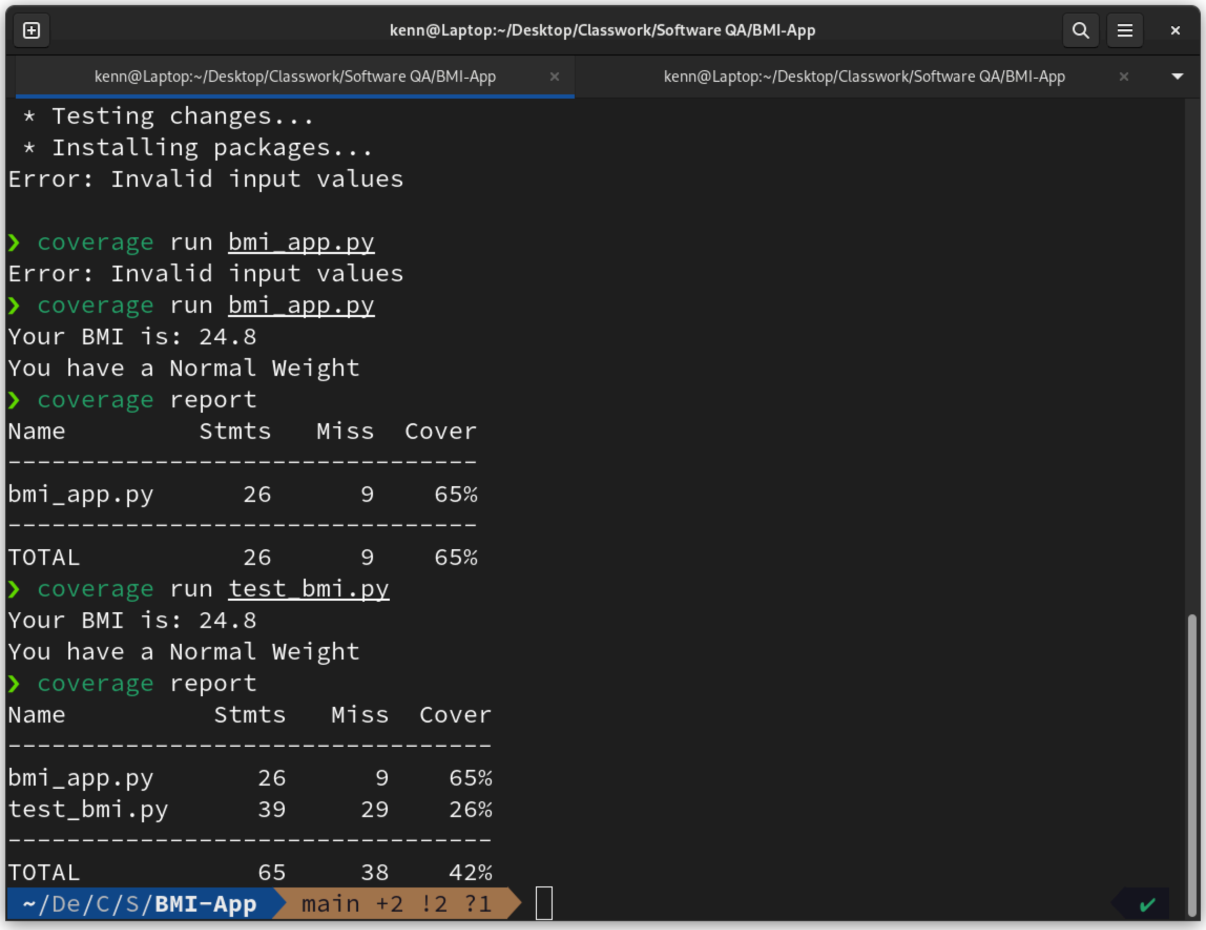
1. **Learning Curve**: Similar to other cloud platforms, GCP has its own set of tools and services, which may require new learning for users, especially those migrating from other platforms. The vast array of options and best practices can be overwhelming initially.
2. **Billing Complexity**: GCP's billing can be intricate due to the granularity of its services. Users need to understand the pricing details of various services to manage costs effectively and avoid unexpected charges.
3. **Integration with Existing Tools**: While GCP provides extensive integrations, organizations with existing investments in tools not natively supported by GCP might face challenges in integration and may need to rely on third-party solutions or custom development.
4. **Resource Management**: Efficiently managing resources in GCP to avoid underutilization or overprovisioning requires a good understanding of the platform's capabilities and continuous monitoring.

Benefits:

1. **Highly Scalable and Reliable**: GCP offers global scalability and reliability, with a robust infrastructure that supports automatic scaling, load balancing, and a global network that ensures fast access and high availability.
2. B**ig Data and Analytics**: GCP excels in big data and analytics solutions, providing powerful tools such as BigQuery for data warehousing, and AI and machine learning capabilities that are deeply integrated into the platform.
3. **Security**: GCP is built on Google's private global fiber network and inherits the same security model that protects services like Gmail and Google Search, with extensive security features, compliance certifications, and best practices.
4. **Cost-Effectiveness**: GCP provides a competitive pricing model, including sustained use discounts and custom machine types, to optimize costs without upfront investments or lengthy commitments.
5. **Open Cloud**: GCP supports an open-cloud approach, facilitating portability and open standards. It's committed to supporting open-source technologies and provides easy integration with various open-source tools.

Given these challenges and benefits, GCP is a highly recommended platform for deploying web applications, particularly for projects that leverage big data, analytics, and machine learning. Its scalable infrastructure, commitment to security, and cost-effective pricing model make it an attractive choice for businesses of all sizes. Overcoming the challenges involves careful planning, cost monitoring, and taking advantage of GCP's extensive documentation and training resources.

Coverage Report



kenn@Laptop:~/Desktop/Classwork/Software QA/BMI-App
kenn@Laptop:~/Desktop/Classwork/Software QA/BMI-App
kenn@Laptop:~/Desktop/Classwork/Software QA/BMI-App

```
 * Testing changes...
 * Installing packages...
Error: Invalid input values

❯ coverage run bmi_app.py
Error: Invalid input values
❯ coverage run bmi_app.py
Your BMI is: 24.8
You have a Normal Weight
❯ coverage report
Name          Stmts   Miss  Cover
---------------------------------
bmi_app.py       26      9    65%
---------------------------------
TOTAL            26      9    65%
❯ coverage run test_bmi.py
Your BMI is: 24.8
You have a Normal Weight
❯ coverage report
Name           Stmts   Miss  Cover
----------------------------------
bmi_app.py        26      9    65%
test_bmi.py       39     29    26%
----------------------------------
TOTAL             65     38    42%
 ~/De/C/S/BMI-App  main +2 !2 ?1 
```

Conclusion

In conclusion, the development and deployment of the BMI Calculator web application demonstrated the effectiveness of integrating advanced software engineering tools and methodologies. By navigating challenges such as tool configuration and code quality maintenance, the project underscored the importance of continuous integration, automated testing, and cloud deployment. Utilizing Google Cloud Platform for deployment and Coverage Gutters for code quality analysis proved essential for ensuring the application's reliability and scalability. This project serves as a valuable model for future endeavors, emphasizing the critical role of automation and precise tool selection in enhancing software development processes. Ultimately, it highlights the project's success in delivering a high-quality, efficient, and user-friendly application.

References

OpenAI. (2024). *ChatGPT* (3.5) [Large language model]. https://chat.openai.com