Web-Based Weather Application DevOps Pipeline

Project Brief

SEPTEMBER 27, 2024

CLASS-5 FRIDAY 2:30 TEAM 4

Chowdhury Mohammad Jarif Ferdous #103792481, Hossain Muntasir # 103798748, Hridoy Ahmed # 103798793, Sazed Ur Rahman #103815346

Table of Contents

1. Introduction 2
2. Background of the Project 2
    DevOps Server Pipeline and Deployment 2
3. Deployment Workflow: 4
4. Project Goals and Objectives 6
5. Selected Tools and Their Justification 7
        1. GitHub (Version Control) 7
        2. GitHub Actions (CI/Build Server) 7
        3. AWS Elastic Beanstalk (Production Server) 7
        4. AWS CloudWatch (Monitoring) 8
        5. Automated Scaling (AWS Lambda) 8
6. Success Metrics 9
7. Project Scope and Exclusions 10
        Project Scope 10
        Project Exclusions 10
8. Project Deliverables 11

Table of Figures

Figure 1 Deployment Workflow 4

1. Introduction

In today's fast-paced software development landscape, the need for reliable and efficient applications is more important than ever. DevOps practices like continuous integration (CI) and continuous deployment (CD) have transformed how development and operations teams work together, making it easier to automate processes and streamline deployments. As applications become more complex, ensuring smooth collaboration between teams is key to maintaining consistent integration, testing, and deployment.

This project focuses on building a DevOps pipeline for a web-based weather application that uses the OpenWeather API to retrieve and display real-time weather data. The pipeline will include essential components such as version control, CI/build, testing, and a production environment. It will demonstrate continuous development and integration through automated processes triggered by code changes, ensuring that the application is always up to date. The pipeline will also incorporate automated testing and performance monitoring to ensure the application remains stable and performs well in production.

The following sections will delve into the design of this pipeline, the tools selected, and the steps involved in implementing a scalable and maintainable solution for future development.

2. Background of the Project

This project aims to develop a web-based weather application that serves as a practical demonstration of the DevOps software development lifecycle. While the application itself is simple, it presents ample opportunities to showcase essential DevOps principles and tools, especially in the context of continuous integration and continuous deployment (CI/CD). The focus will be on designing an efficient, automated server pipeline that integrates various DevOps components, from code versioning to deployment and monitoring.

DevOps Server Pipeline and Deployment

The core of this project is the creation of a robust CI/CD pipeline that automates the processes of building, testing, and deploying code. By eliminating manual intervention, this pipeline ensures that any changes pushed to the codebase are automatically verified and deployed to the production environment

For this project, we are planning to use the following components of DevOps pipeline.

- **Version Control (GitHub):**
  GitHub will serve as the primary version control platform, hosting the code for the weather application. Any code changes pushed to the repository will act as the trigger point for the CI/CD process, ensuring that updates are versioned and traceable.
- **CI/Build Server (GitHub Actions):**
  GitHub Actions will automate the building and testing of the application. Upon each code push, GitHub Actions will compile the code and run automated tests to ensure no errors are introduced. This integration validates every commit, ensuring that only stable, functional code proceeds to deployment.
- **Test Server (GitHub Actions):**
  As part of the CI pipeline, GitHub Actions also handles automated testing. This includes running unit tests on the codebase to verify that the application behaves as expected. Any failed tests will halt the deployment process, ensuring only error-free code is deployed to production.
- **Production Server (AWS Elastic Beanstalk):**
  Once the application passes all the tests, it is automatically deployed to AWS Elastic Beanstalk. Elastic Beanstalk manages the infrastructure for deploying and running the web-based calculator, including provisioning the required resources, handling scaling, and monitoring the application's health.
- **Monitoring (AWS CloudWatch):**
  After deployment, AWS CloudWatch monitors the performance of the application, tracking metrics such as CPU utilization, memory usage, and response time. This ensures the application runs efficiently and provides insights into potential performance bottlenecks.
- **Automatic Scaling (AWS Lambda):**
  To handle changes in traffic, AWS Lambda will be used to automatically scale the application based on metrics from AWS CloudWatch. This ensures that the system can dynamically adjust to demand, maintaining performance without manual oversight.

3. Deployment Workflow:

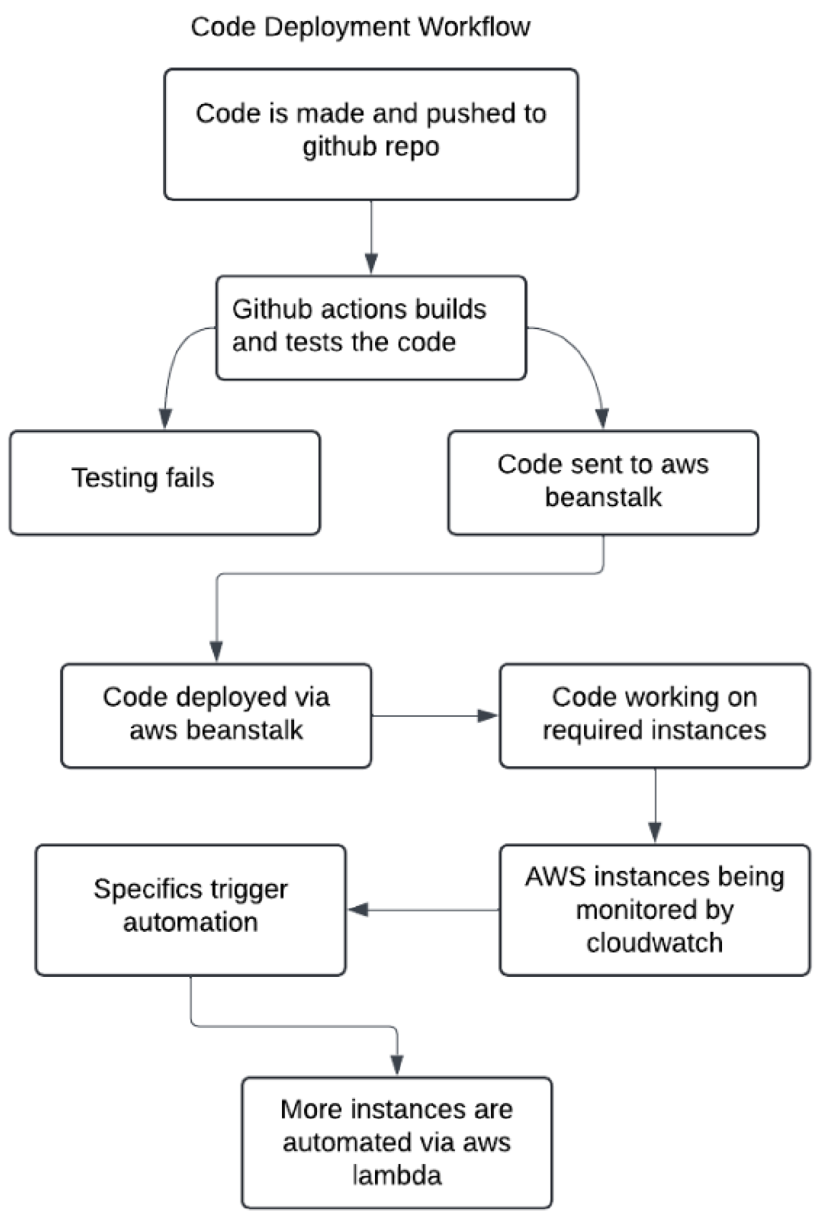
Code Deployment Workflow
Code is made and pushed to github repo
Github actions builds and tests the code
Testing fails
Code sent to aws beanstalk
Code deployed via aws beanstalk
Code working on required instances
AWS instances being monitored by cloudwatch
Specifics trigger automation
More instances are automated via aws lambda

Figure 1 Deployment Workflow

The code deployment workflow illustrated in the diagram represents an efficient DevOps pipeline integrating GitHub Actions and AWS services. When new code is pushed to the GitHub repository, GitHub Actions automatically trigger a build and runs tests to ensure code quality. If tests fail, deployment is stopped, preventing unstable releases. Upon successful testing, the code is deployed to AWS Elastic Beanstalk.

The application runs on AWS instances, continuously monitored by AWS CloudWatch for performance and health. If specific conditions are met, AWS Lambda automatically scales the application by provisioning more instances, ensuring it can handle varying workloads. This pipeline ensures continuous delivery, automated scaling, and monitoring, creating a robust, scalable production environment.

- Code Development & Push:
  Developers write and push code to the GitHub repository.

- GitHub Actions Initiates Build:
  Automated build process is triggered by GitHub Actions upon code push.

- Automated Testing:
  GitHub Actions runs tests to validate the code.
  If tests fail, deployment halts, ensuring only stable code is released.

- Successful Code Deployment:
  On passing tests, code is sent to AWS Elastic Beanstalk for deployment.

- Application Runs on AWS Instances:
  The application runs on AWS-provisioned instances.

- Continuous Monitoring via AWS CloudWatch:
  AWS CloudWatch monitors the health and performance of the deployed instances.

- Automatic Scaling via AWS Lambda:
  Based on CloudWatch metrics, AWS Lambda provisions additional instances if necessary to handle increased demand.

- High Availability & Responsiveness:
  The system maintains performance and scalability with automatic resource adjustments, ensuring consistent availability.

4. Project Goals and Objectives

The primary goal of this project is to develop and deploy a fully functional web-based weather application using a comprehensive DevOps pipeline that automates the entire software lifecycle, from code updates to production deployment, while also ensuring scalability and performance monitoring.

Objectives:

- Develop a web-based weather application that retrieves real-time weather data from the OpenWeather API.
- Establish a CI/CD pipeline using GitHub Actions to automate the process of building, testing, and deploying the application.
- Automate the deployment process so that any code changes are tested and seamlessly deployed to the production environment using AWS Elastic Beanstalk.
- Monitor the performance and health of the deployed application through AWS CloudWatch, tracking key metrics for optimization.
- Implement AWS Lambda to automatically scale the application in response to workload demands, ensuring the system can dynamically adjust based on usage patterns without manual intervention.

5. Selected Tools and Their Justification

1. GitHub (Version Control)

GitHub is our chosen version control platform, widely known for its ease of use and extensive collaboration features. It allows us to track code changes, collaborate as a team, and integrate seamlessly with GitHub Actions for automated workflows.

Justification: GitHub is a standard tool for version control, enabling collaboration and version tracking with ease. Its direct integration with GitHub Actions allows for streamlined automation of the CI/CD pipeline, reducing the need for additional tools.

2. GitHub Actions (CI/Build Server)

GitHub Actions automates our continuous integration and deployment processes. Every time code is pushed to the repository, it automatically triggers builds, runs tests, and prepares the app for deployment.

Justification: GitHub Actions is built directly into GitHub, making it an ideal choice for automating the build and testing process. Its easy setup and direct integration with our repository save time and reduces complexity, ensuring a smooth CI/CD process without requiring additional external tools.

3. AWS Elastic Beanstalk (Production Server)

AWS Elastic Beanstalk simplifies the deployment process by automatically managing infrastructure tasks such as server provisioning, scaling, and monitoring. It integrates directly with GitHub Actions for continuous deployment.

Justification: AWS Elastic Beanstalk allows for fast and automated deployment without requiring manual infrastructure management. Its scalability and ease of use make it an excellent choice for deploying our web-based calculator, allowing us to focus on development instead of server management.

4. AWS CloudWatch (Monitoring)

AWS CloudWatch provides real-time monitoring and performance metrics for our deployed application. It tracks CPU usage, memory, and response times, allowing us to ensure the app is running smoothly and efficiently.

Justification: AWS CloudWatch is fully integrated with AWS services, making it the perfect tool for monitoring our app. Its real-time metrics and alerting system help us catch performance issues early, ensuring that our app runs efficiently in production.

5. Automated Scaling (AWS Lambda)

AWS Lambda is a serverless compute service that allows code to run in response to specific events. In this pipeline, Lambda will be triggered when CloudWatch detects high resource usage, such as increased CPU or memory consumption.

Justification: AWS Lambda automates the process of scaling up by provisioning additional Elastic Beanstalk instances, ensuring the application can handle increased load without downtime. The serverless nature of Lambda makes it a cost-efficient and scalable solution for automating resource management in response to real-time conditions

The DevOps pipeline ensures continuous integration, automated testing, seamless deployment, and dynamic scaling—all with minimal human intervention. This infrastructure allows for efficient development, reliable performance, and the ability to adapt to changing demands in real time.

6. Success Metrics

The success of the project will be measured based on the following criteria:

- **Successful Deployment:** The web-based Weather Application is successfully deployed and accessible from a public URL hosted on AWS Elastic Beanstalk.
- **Deployment Frequency:** To measure how often new code changes are successfully deployed to the production environment. Higher deployment frequency indicates a more efficient CI/CD pipeline.
- **Automated Testing and Deployment:** Code changes in the GitHub repository automatically trigger the CI/CD pipeline, including building the application, running tests, and deploying to production without manual intervention.
- **Performance Monitoring:** AWS CloudWatch metrics confirm that the application is running efficiently, with no performance bottlenecks, and that response times are within acceptable limits.
- **Uptime and Availability:** The application will be monitored for 99.9% uptime, ensuring it is always available to users.
- **Automatic Scaling Efficiency:** To measure the responsiveness and effectiveness of the automatic scaling process triggered by AWS Lambda and monitored by AWS CloudWatch. This includes tracking how quickly additional instances are provisioned in response to traffic spikes and how effectively the application maintains performance during these periods.

7. Project Scope and Exclusions

Project Scope

- **Web-Based Weather Application:**
  - Develop an application that retrieves and displays current weather data using the OpenWeather API.
- **User Interface Design:**
  - Create a user-friendly and responsive interface for various devices.
- **DevOps Pipeline Implementation:**
  - Establish a CI/CD pipeline using GitHub Actions for automated build, testing, and deployment.
- **Version Control:**
  - Use GitHub as the primary version control system for code management.
- **AWS Services Integration:**
  - Deploy the application to AWS Elastic Beanstalk and monitor performance with AWS CloudWatch.
- **AWS Lambda for Scaling:**
  - Implement serverless functions to enable automatic scaling based on application load.
- **Performance Monitoring:**
  - Continuously track application metrics to ensure reliability and efficiency.

Project Exclusions

- **Advanced Features:**
  - Exclude complex functionalities like historical weather data and third-party alerts.
- **Extensive Performance Optimization:**
  - No focus on advanced optimization techniques beyond standard practices.
- **User Authentication:**
  - Exclude user account management or authentication features.
- **In-Depth Security Compliance:**
  - No extensive security measures or regulatory compliance beyond basic best practices.

8. Project Deliverables

This project will deliver the following components:

❖ **Web-Based Weather Application:**
A fully functional application that retrieves and displays current weather data from the OpenWeather API.

❖ **User Interface Design:**
A responsive and intuitive user interface optimized for desktop and mobile devices.

❖ **CI/CD Pipeline Documentation:**
Comprehensive documentation outlining the setup and functioning of the CI/CD pipeline using GitHub Actions.

❖ **Source Code Repository:**
A well-organized GitHub repository containing all application code, including version control history.

❖ **Deployment on AWS Elastic Beanstalk:**
A deployed version of the application running on AWS Elastic Beanstalk with appropriate configurations.

❖ **Performance Monitoring Setup:**
Integration of AWS CloudWatch for real-time monitoring of application performance metrics.

❖ **AWS Lambda Implementation:**
Serverless functions configured for automatic scaling based on application demand.

❖ **Testing Report:**
A report summarizing the results of automated tests conducted during the CI/CD process, including any issues encountered.

❖ **User Manual:**
A guide for end-users detailing how to use the application, including features and troubleshooting tips.

❖ **Project Closure Report:**
A final report summarizing project outcomes, lessons learned, and recommendations for future enhancements.