

# DOE522\_FA1

DUE DATE: 2024/08/16

Jancke du Toit

STUDENT NUMBER 20231562

## QUESTION 1:

### 1.1 Key principles of DevOps:

**Collaboration and Communication:** DevOps promotes collaboration between development (Dev) and operations (Ops) teams, breaking down silos and ensuring constant communication throughout the software development lifecycle.

**Automation:** Automation of repetitive tasks like testing, integration, and deployment is a core principle of DevOps. This reduces human error, improves efficiency, and allows teams to focus on innovation.

**Continuous Integration/Continuous Deployment (CI/CD):** DevOps emphasizes continuous integration of code changes into a shared repository and continuous deployment of software updates, making sure that the program is always ready for use.

**Monitoring and Feedback:** DevOps encourages continuous monitoring of applications and infrastructure to get feedback, identify issues early, and improve future development cycles.

**Iterative Improvement:** DevOps encourages an iterative approach to software development, where the feedback is used to make continuous improvements to the product and the development process.

### Benefits of DevOps:

**Increased Efficiency:** By automating routine tasks and continuous integration and deployment, the time lost on manual operations is decreased and resources are used more effectively.

**Improved Quality:** Continuous testing and monitoring help to detect and address issues early in the development process, resulting in higher-quality software.

**Faster Time to Market:** By automating processes and collaboration, DevOps speeds up software releases by reducing the time needed to build, test, and deploy software.

**Greater Innovation:** By saving time through automation, teams can focus more on innovation and developing new features that add value to the product.

**Enhanced Collaboration:** DevOps breaks down the barriers between development and operations teams, leading to better alignment, shared goals, and a stronger focus on the end product.

## **1.2 Enhancing Collaboration:**

- DevOps promotes shared responsibility, where both development and operations teams work together to achieve shared goals. Tools like Slack and Jira facilitate real-time communication, while shared dashboards ensure that everyone is on the same page.

### **Accelerating Software Delivery:**

- The automation of CI/CD pipelines allows for faster integration and deployment of code changes. This means that new features, updates, and bug fixes can be delivered to customers faster. For example, companies using DevOps practices can release updates multiple times a day, compared to traditional methods where releases might happen quarterly or annually.

### **Improving Product Quality:**

- Continuous testing and monitoring results in the early detection of bugs and performance issues, this leads to more reliable and stable software. DevOps practices like Test-Driven Development (TDD) and automated testing ensure that code is thoroughly tested before it reaches production, thereby improving overall quality.

## **1.3 Real world examples:**

**Netflix:** One of the best examples of a DevOps implementation is Netflix. By automating their deployment pipeline and embracing continuous delivery, Netflix can deploy hundreds of updates every day. This has made it possible for them to develop quickly while maintaining a highly reliable streaming service for millions of users.

**Takealot:** As one of South Africa's biggest online retailers, Takealot uses DevOps to manage its complex e-commerce platform. By implementing CI/CD pipelines and automation, Takealot has significantly reduced the time it takes to release new features and updates, ensuring a seamless shopping experience for customers.

**Facebook:** Facebook relies heavily on DevOps practices to manage its massive user base and continuously changing platform. By using continuous integration and deployment strategies, Facebook can release new features, fixes, and improvements several times a day. This agility allows Facebook to stay ahead of the competition and respond quickly to user needs.

## QUESTION 2:

### 2.1 Setting up a version control system for a software development project:

1. **Choose a Version Control System:** Choose between a centralized VCS (like SVN) or a distributed VCS (like Git). Git is more common because of its flexibility and adaptability.
2. **Install the VCS:** Install the chosen VCS on your local machine. You can get Git from the official Git website and install it there.
3. **Initialize a Repository:** In the project directory, create a new repository. In Git, you would use the command `git init` to initialize a new repository.
4. **Set Up a Remote Repository:** If you're working in a team, create a remote repository using GitHub, GitLab, or Bitbucket. Clone the remote repository to your local machine using the `git clone` command.
5. **Configure User Settings:** Configure your VCS with your username and email, which will be linked to your commits. Git offers two commands for configuring users: `git config --global user.name "Your Name"` and `git config --global user.email "youremail@example.com"`.
6. **Add Files and Make Commits:** Add your project files to the repository and make your first commit using `git add .` followed by `g`.

## 2.2 Centralized Version Control Systems (CVCS) - e.g., SVN:

### Advantages:

- **Simplicity:** CVCS are typically easier to set up and manage, making them suitable for smaller teams or projects with less complexity.
- **Single Source of Truth:** With a central repository, all team members work from a single, authoritative version of the project, which makes version management easier.
- **Easier Access Control:** Administrators can easily manage access permissions since all data is stored on a central server.

### Disadvantages:

- **Single Point of Failure:** If the central server goes down, no one can access the repository, potentially stopping development.
- **Limited Offline Work:** Developers have to be connected to the central server to commit changes, limiting their ability to work offline.
- **Performance Issues:** As the project grows, performance can go down because the server becomes a bottleneck for large files or complex operations.

## Distributed Version Control Systems (DVCS) - e.g., Git:

### Advantages:

- **Local Repositories:** Each developer can work offline and commit changes locally as they have a complete copy of the repository.
- **Enhanced Collaboration:** Developers can create branches and merge them back into the main branch without affecting others, allowing for more experimentation and parallel development.
- **Resilience:** With multiple copies of the repository, there's no single point of failure, making DVCS more resilient.
- **Faster Operations:** Operations like commits, branching, and merging are usually faster because they happen locally.

### Disadvantages:

- **Complexity:** DVCS can be more complex to learn and manage, especially for beginners or smaller teams.
- **Possibility of Divergence:** If branches are not properly managed, they may diverge significantly, which will make mergers more difficult.
- **Larger Local Storage:** More local storage is needed because each developer has a complete copy of the repository.

## 2.3 Importance of Code Reviews in a DevOps Environment:

### Enhancing Software Quality:

Code reviews make sure that code is checked for errors, bugs, and potential security vulnerabilities before it is merged into the main branch. This process helps to identify issues early in the development cycle, reducing the possibility of flaws reaching production.

### Improving Team Collaboration:

Code reviews encourage sharing knowledge with team members. By reviewing each other's code, team members learn different coding styles, best practices, and the overall architecture of the project. This leads to a more cohesive and collaborative development environment.

### Maintaining Code Consistency:

Code reviews help maintain coding standards and consistency across the project. By enforcing guidelines, code reviews ensure that the codebase is clean, readable, and manageable, which is crucial for the long-term success of the project.

### Facilitating Continuous Improvement:

In a DevOps environment, code reviews support continuous improvement. Review feedback can lead to better coding practices and a more efficient development process over time.

### Enforcing Compliance:

Code reviews can also guarantee that the code complies with industry regulations, security standards, and company policies. This is particularly important in industries like finance and healthcare, where compliance is essential.

## 2.4 Best Practices for Conducting Effective Code Reviews

**Focus on the Code, Not the Developer:** Recommendations should be objective and focused on improving the code rather than criticizing the individual who wrote it.

**Review Small, Incremental Changes:** By reviewing smaller changes, you can make the process easier to handle and less likely to miss important details.

**Provide Constructive Feedback:** Make suggestions for improvement rather than just pointing out what's wrong. Be specific about what can be improved and why.

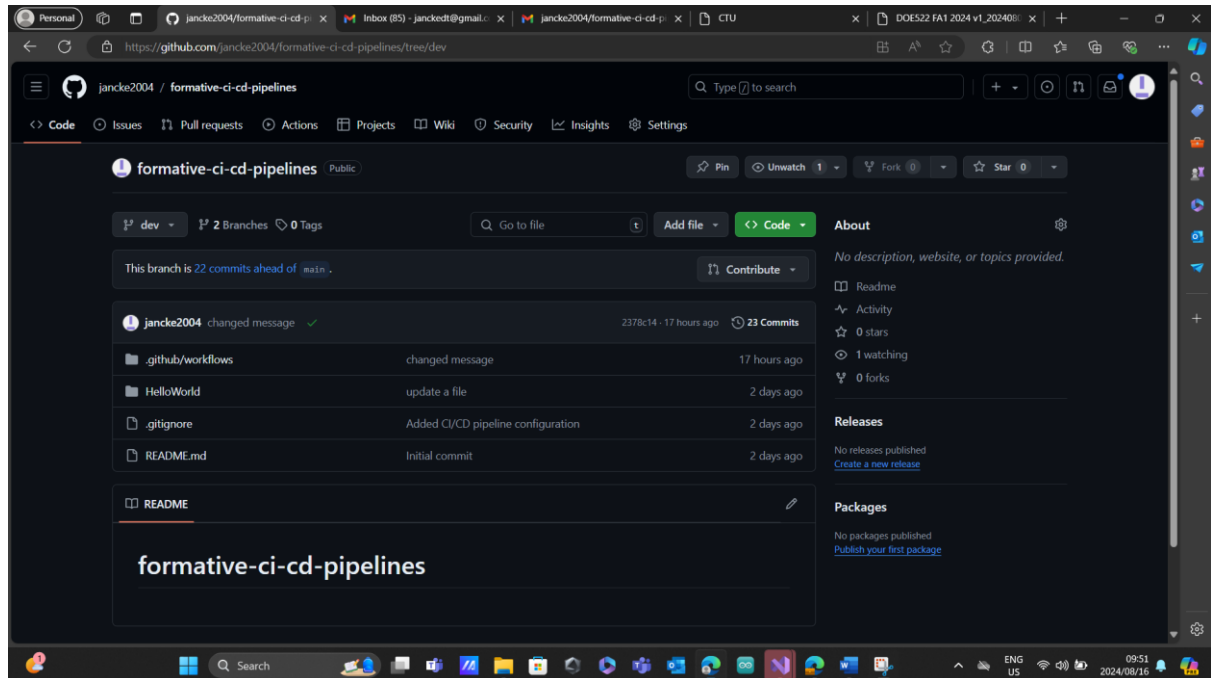
**Use Automated Tools:** Make use of automated code review tools to identify typical problems like style violations or potential bugs. This allows the human reviewer to focus on more complex aspects.

**Set Clear Guidelines:** Establish clear code review guidelines and standards that everyone on the team follows. This guarantees consistency and helps streamline the review process.

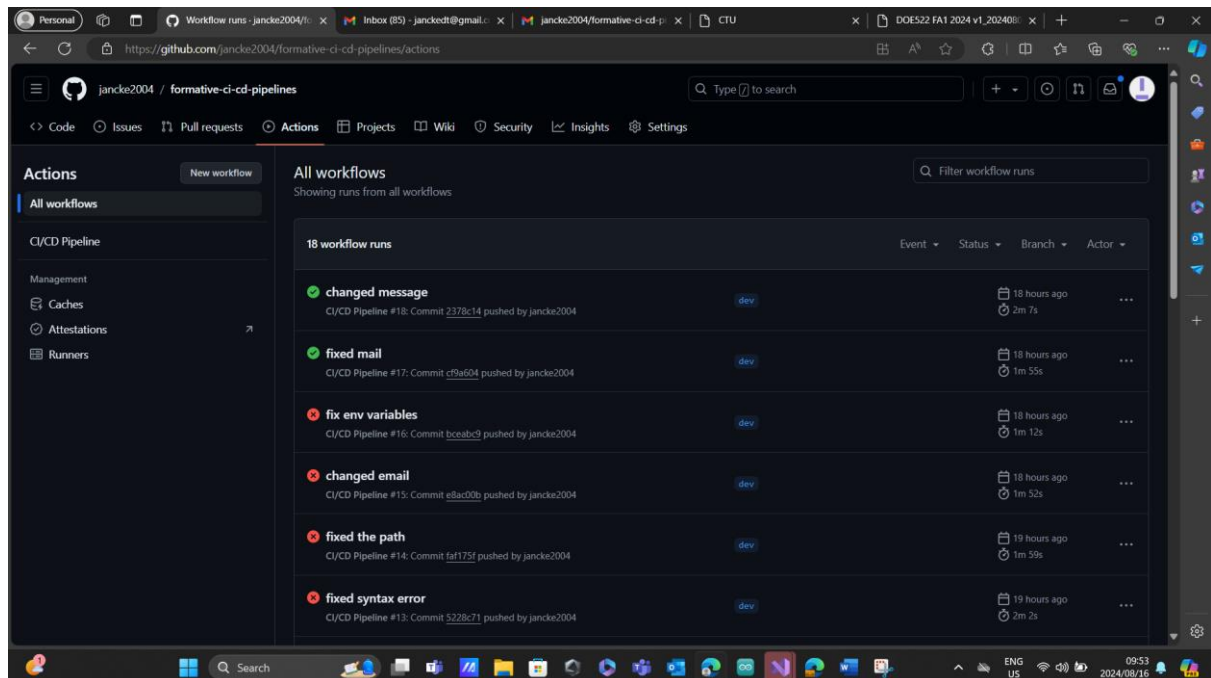
## QUESTION 3:

3.1

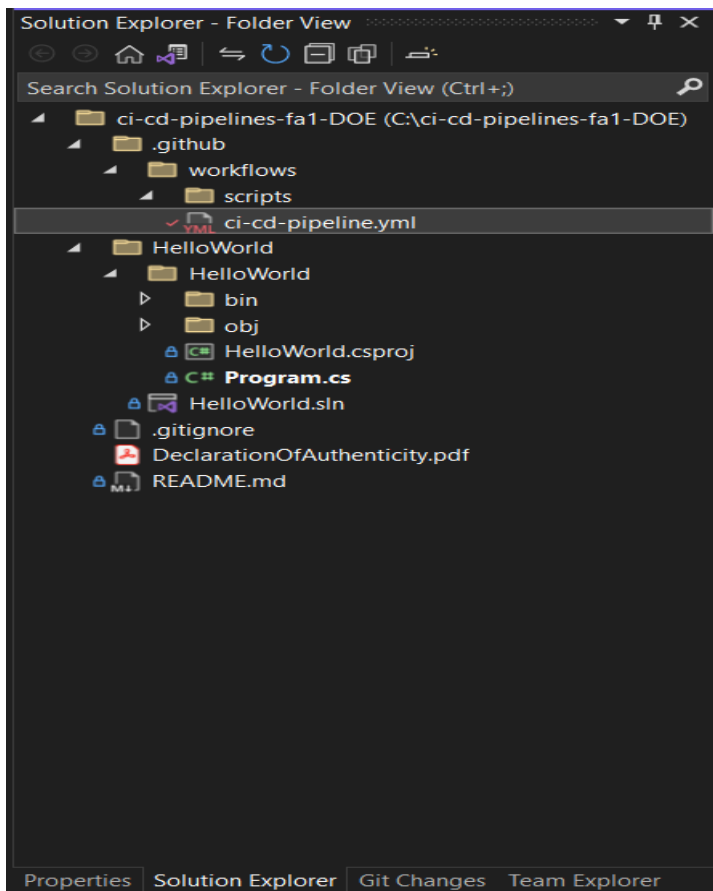
My dev branch in github where everything gets pushed to:



Actions in my github:

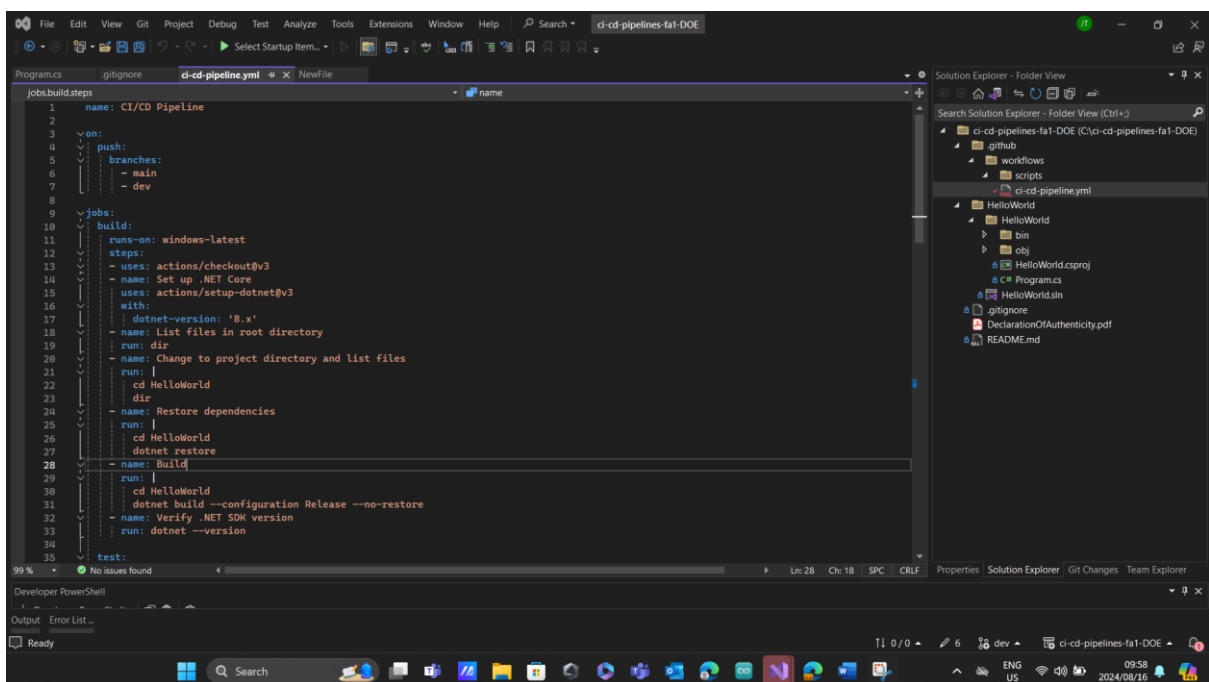


All files in my project:



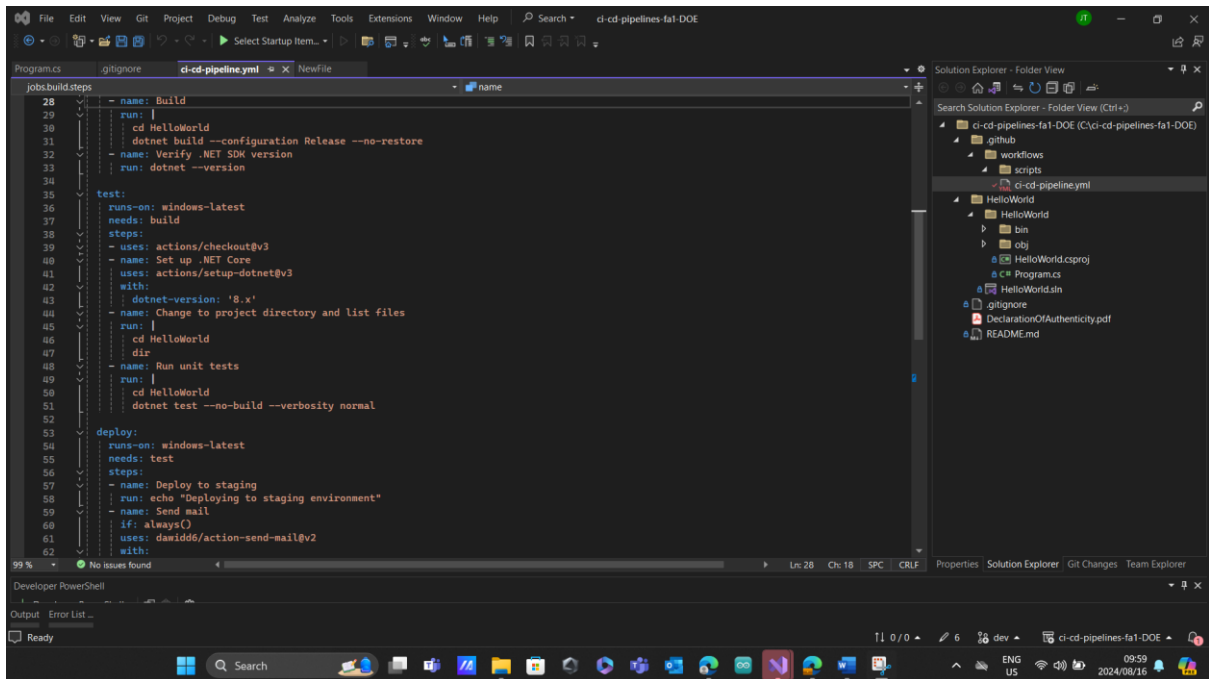
Code :

Build:





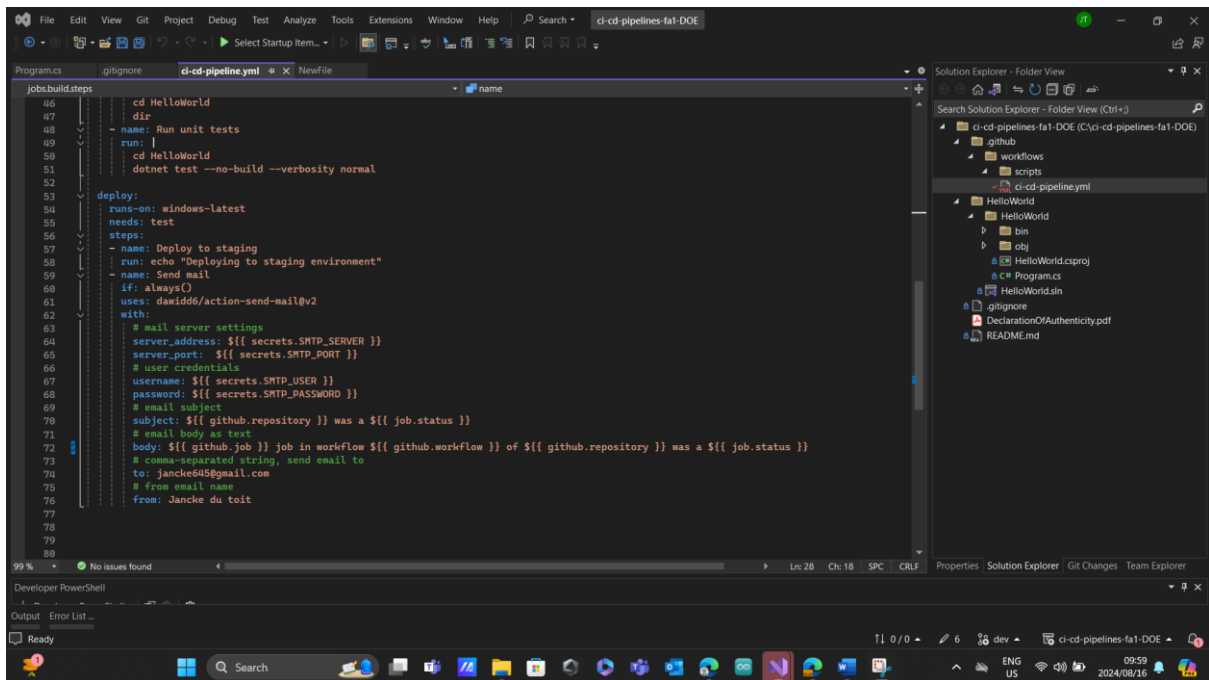
Test:



The screenshot shows the Visual Studio Code editor with a file named `ci-cd-pipeline.yml` open. The file contains a GitHub Actions workflow with three main jobs: `Build`, `test`, and `deploy`. The `test` job is currently selected and expanded, showing its configuration. The `test` job is configured to run on `windows-latest`, needs the `build` job, and uses the `actions/checkout@v3` and `actions/setup-dotnet@v3` actions. It also specifies `dotnet-version: '8.x'`. The `run` steps for the `test` job are: `cd HelloWorld`, `dir`, and `dotnet test --no-build --verbosity normal`. The `deploy` job is also visible, configured to run on `windows-latest`, needs the `test` job, and includes steps for deploying to staging and sending an email.

```
jobs:
  build:
    name: Build
    run: |
      cd HelloWorld
      dotnet build --configuration Release --no-restore
      name: Verify .NET SDK version
      run: dotnet --version
  test:
    runs-on: windows-latest
    needs: build
    steps:
      - uses: actions/checkout@v3
      - name: Set up .NET Core
        uses: actions/setup-dotnet@v3
        with:
          dotnet-version: '8.x'
      - name: Change to project directory and list files
        run: |
          cd HelloWorld
          dir
      - name: Run unit tests
        run: |
          cd HelloWorld
          dotnet test --no-build --verbosity normal
  deploy:
    runs-on: windows-latest
    needs: test
    steps:
      - name: Deploy to staging
        run: echo "Deploying to staging environment"
      - name: Send mail
        if: always()
        uses: dawidd6/action-send-mail@v2
        with:
          # mail server settings
          server_address: ${{ secrets.SMTP_SERVER }}
```

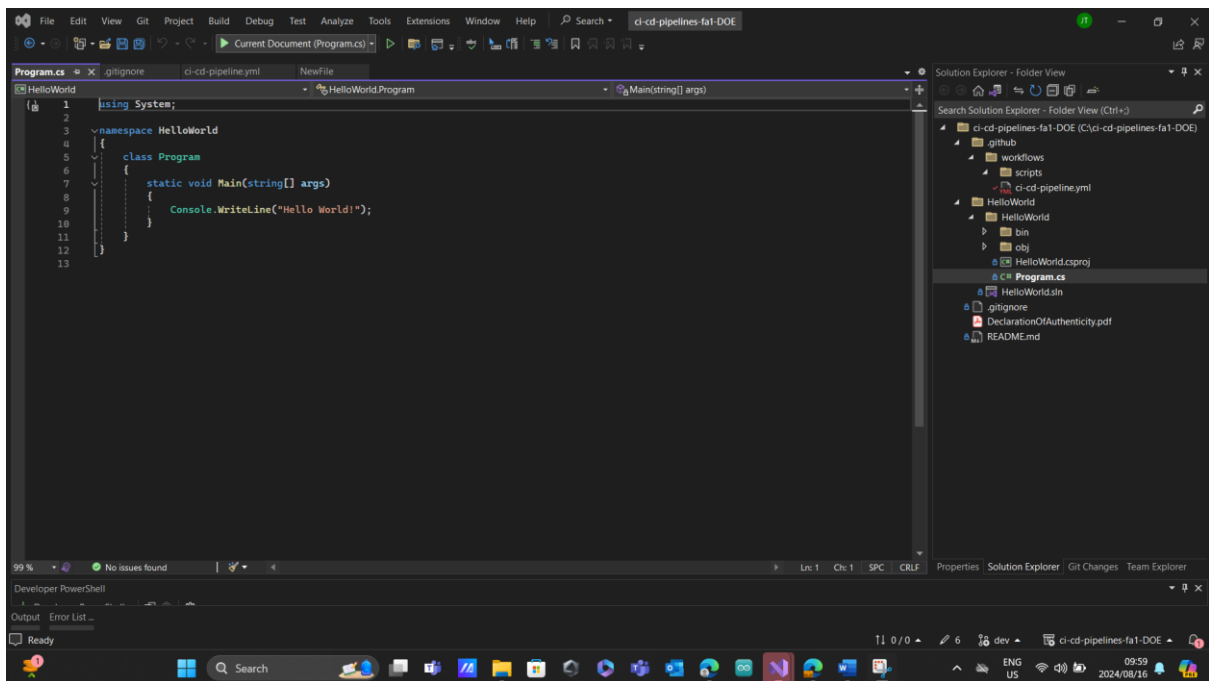
Deploy:



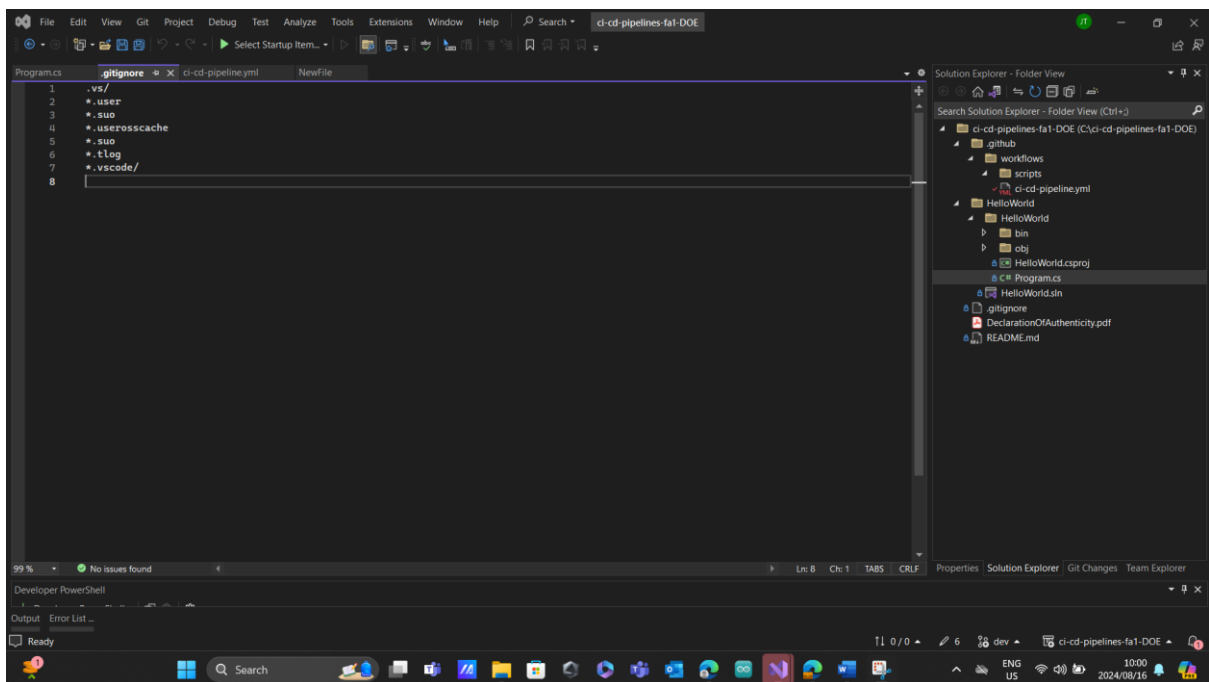
The screenshot shows the Visual Studio Code editor with the same `ci-cd-pipeline.yml` file. The `deploy` job is now selected and expanded, showing its configuration. The `deploy` job is configured to run on `windows-latest`, needs the `test` job, and uses the `actions/checkout@v3` and `actions/setup-dotnet@v3` actions. It also specifies `dotnet-version: '8.x'`. The `run` steps for the `deploy` job are: `cd HelloWorld`, `dir`, and `dotnet test --no-build --verbosity normal`. The `test` job is also visible, configured to run on `windows-latest`, needs the `build` job, and includes steps for deploying to staging and sending an email.

```
jobs:
  build:
    name: Build
    run: |
      cd HelloWorld
      dotnet build --configuration Release --no-restore
      name: Verify .NET SDK version
      run: dotnet --version
  test:
    runs-on: windows-latest
    needs: build
    steps:
      - uses: actions/checkout@v3
      - name: Set up .NET Core
        uses: actions/setup-dotnet@v3
        with:
          dotnet-version: '8.x'
      - name: Change to project directory and list files
        run: |
          cd HelloWorld
          dir
      - name: Run unit tests
        run: |
          cd HelloWorld
          dotnet test --no-build --verbosity normal
  deploy:
    runs-on: windows-latest
    needs: test
    steps:
      - name: Deploy to staging
        run: echo "Deploying to staging environment"
      - name: Send mail
        if: always()
        uses: dawidd6/action-send-mail@v2
        with:
          # mail server settings
          server_address: ${{ secrets.SMTP_SERVER }}
```

HelloWorld file:

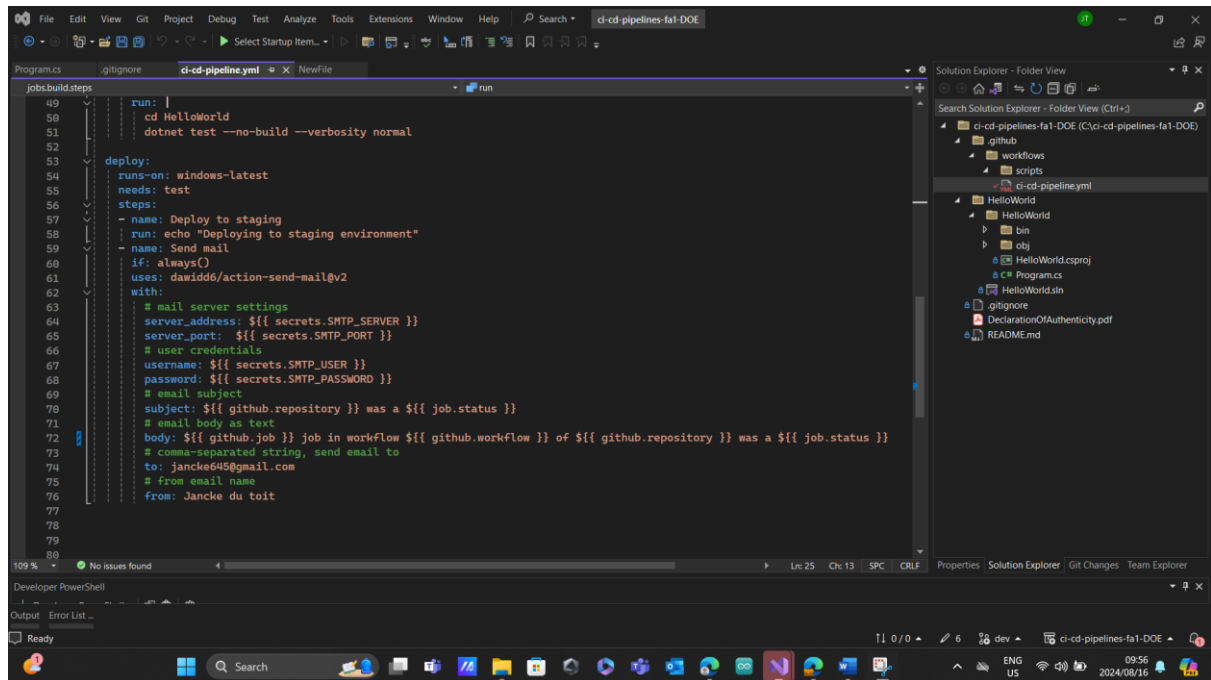


.gitignore file:



### 3.2

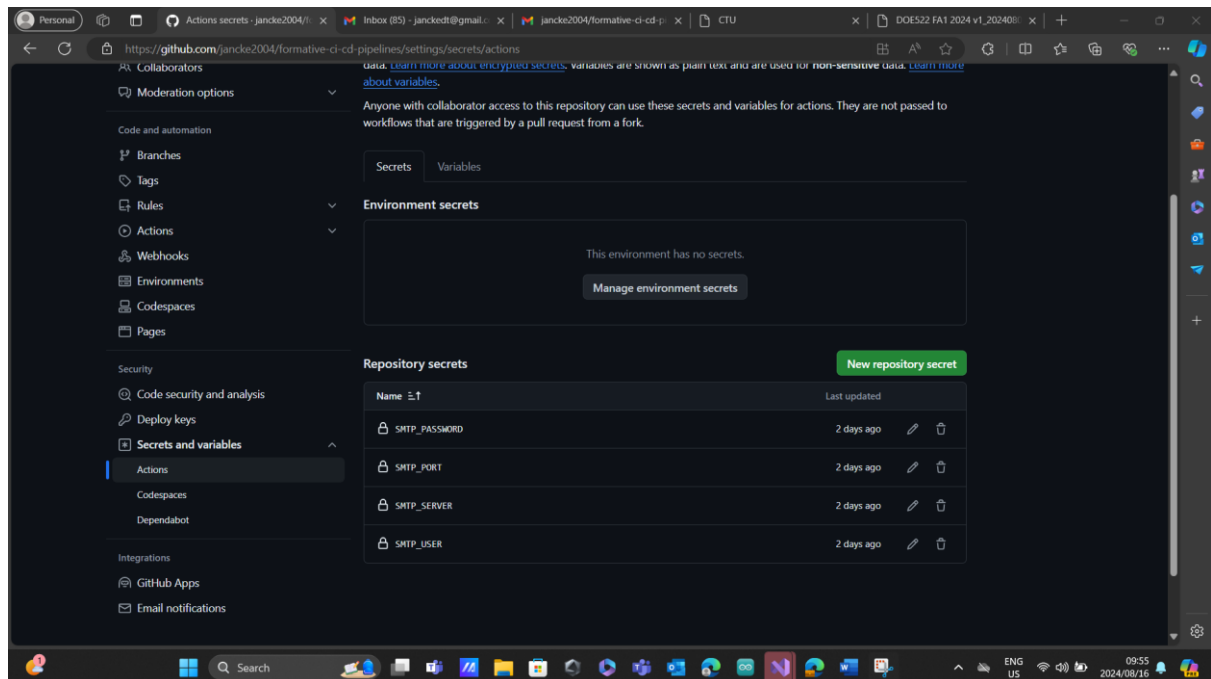
Code for the mail:



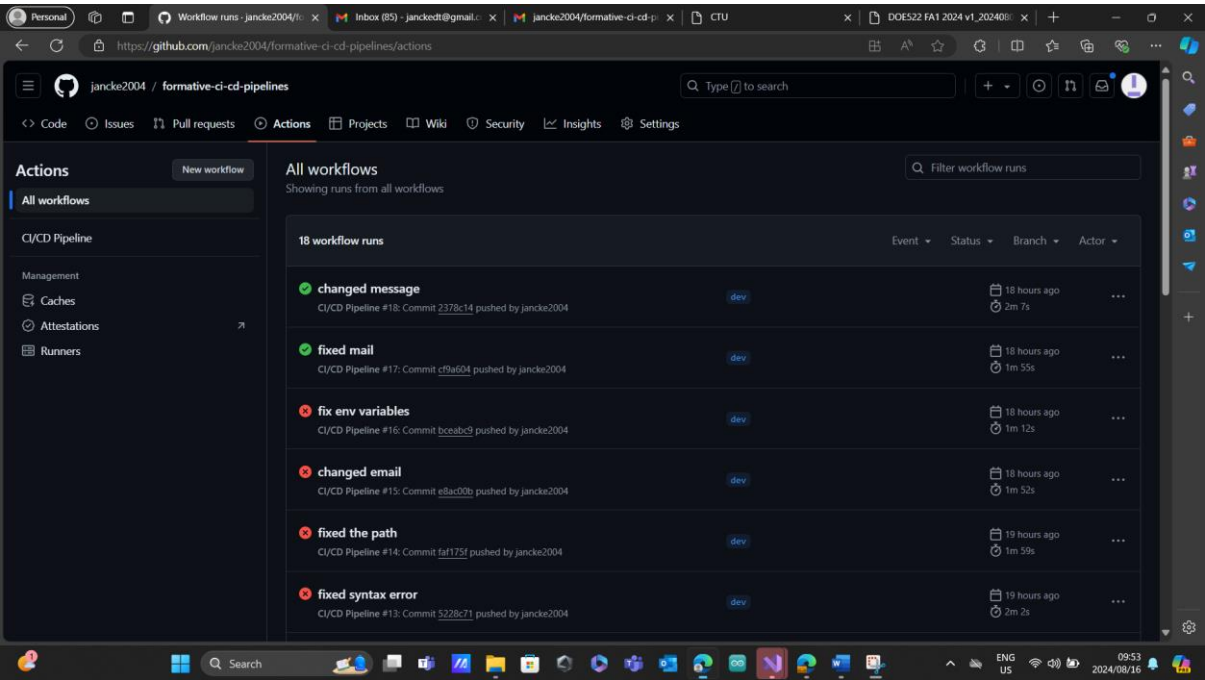
```
jobs:
  build:
    runs-on: windows-latest
    steps:
      - name: HelloWorld
        run: |
          cd HelloWorld
          dotnet test --no-build --verbosity normal

  deploy:
    runs-on: windows-latest
    needs: test
    steps:
      - name: Deploy to staging
        run: echo "Deploying to staging environment"
      - name: Send mail
        if: always()
        uses: dawidd6/action-send-mail@v2
        with:
          # mail server settings
          server_address: ${ secrets.SMTP_SERVER }
          server_port: ${ secrets.SMTP_PORT }
          # user credentials
          username: ${ secrets.SMTP_USER }
          password: ${ secrets.SMTP_PASSWORD }
          # email subject
          subject: ${ github.repository } was a ${ job.status }
          # email body as text
          body: ${ github.job } job in workflow ${ github.workflow } of ${ github.repository } was a ${ job.status }
          # comma-separated string, send email to
          to: jancke645@gmail.com
          # from email name
          from: Jancke du toit
```

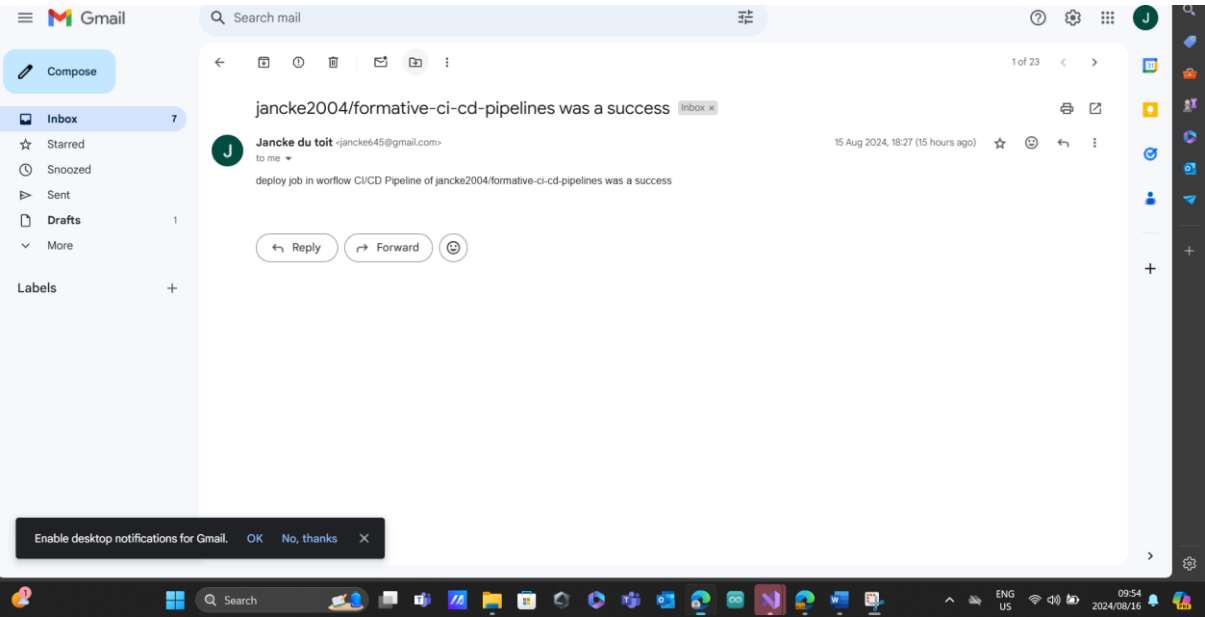
Secrets in github set:



Actions pushed to github



Mail sent to say that the pipelines was a success:



### 3.3 Advantages of Using IaC in a CI/CD Pipeline:

#### 1. Consistency and Standardization:

IaC ensures consistent and standardized infrastructure across all environments, reducing configuration drift. For example, Terraform can apply the same infrastructure configuration to development, testing, and production environments.

#### 2. Automation and Efficiency:

IaC automates infrastructure provisioning and configuration, which speeds up environment setup and lowers manual errors. Tools like Ansible can automate server configurations, making sure that each environment is set up correctly and efficiently.

#### 3. Version Control and Collaboration:

Infrastructure code can be stored in a version control system like Git, allowing teams to track changes, roll back configurations, and collaborate on infrastructure management. This integration supports DevOps processes by improving team collaboration.

### Challenges of Using IaC in a CI/CD Pipeline:

#### 1. Complexity and Learning Curve:

Implementing IaC requires learning new tools and languages, which can be challenging for teams who are unfamiliar with these concepts. For example, learning Terraform or Ansible can take time and requires specialized knowledge.

#### 2. Security Concerns:

Storing infrastructure code in a version control system introduces security risks, like exposing sensitive information like API keys. It is essential to implement best practices like encryption and access controls to mitigate these risks.

### Examples of IaC Tools Integrated into CI/CD Pipelines:

- **Terraform:** Used for provisioning and managing infrastructure across many cloud providers, Terraform can be integrated into CI/CD pipelines to apply infrastructure changes during deployment.
- **Ansible:** Automates server and service configuration, ensuring consistent setups across environments when integrated into CI/CD pipelines.

**Bibliography:**

<https://www.atlassian.com/git/tutorials/what-is-version-control>

<https://guides.github.com/introduction/flow/>

<https://devops.com/the-importance-of-code-reviews-in-devops/>