Assignment 2: CI/CD Pipeline

System Design:
Our CI/CD pipeline works by leveraging github, github actions, jest testing, docker images and dockerhub to continually deploy the latest functional version of our application to our website.

The flow goes roughly as follows:

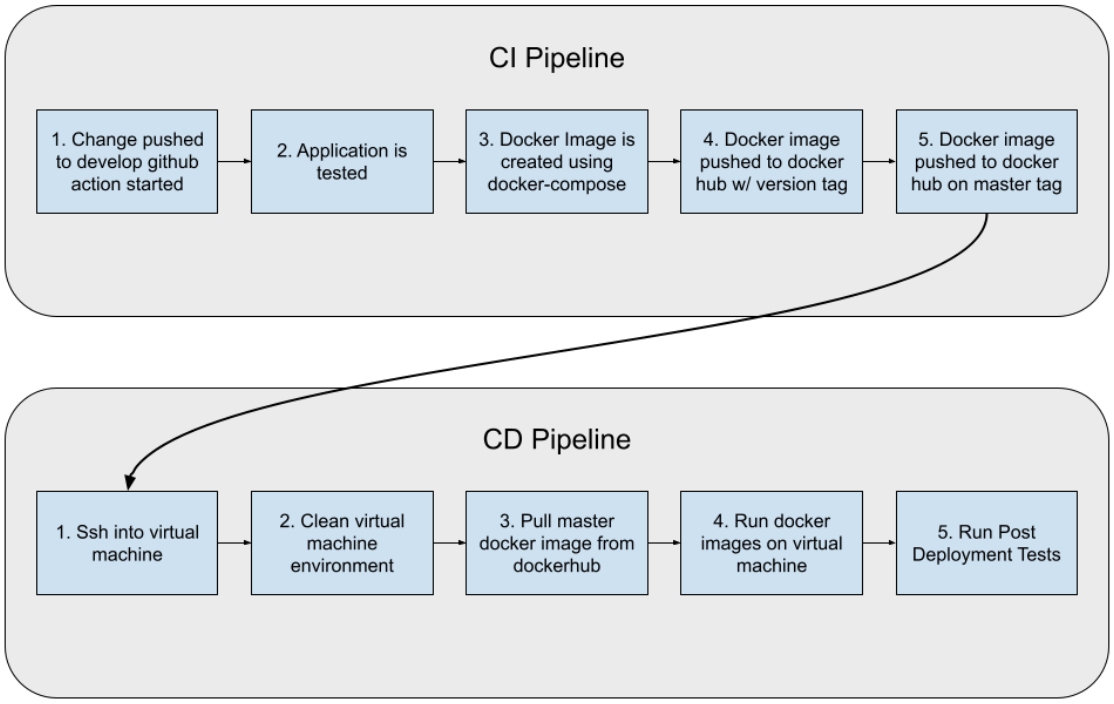
CI Pipeline:

1. A change is pushed to develop where a github action is then started.
2. The github runs tests to see if the currently pushed version passes all steps. If so we move on.
3. We construct a docker image (using docker compose and other dockerfiles) for our application's frontend and backend
4. We push these two images to docker hub on with a version tag that is the github push tag
5. We then push these two images again to docker hub this on the master branch

CD Pipeline:

1. Login to our virtual machine hosting our website
2. Clean virtual machine environment (stopping and removing previous images)
3. Retrieve the master version of our docker images from docker hub and store it our virtual machine
4. We launch our docker images on the virtual machine so it is visible to all users
5. We use jest tests again to ensure post deployment our application is functioning properly

Workflow Diagram:



Tools:

- Github: To store our application with proper version control
- Github actions: To handle the CI/CD workflow
- Jest testing: To ensure our application's functionality
- Docker images: To create our application's images
- Docker Hub: To store and manage all application images

Step by Step Workflow Breakdown:
Firstly, let us go over how we complete the CI Pipeline according to assignment 2

CI Pipeline:
1. Push your code to GitLab, including all necessary dependencies, and prepare it for running tests. (10 marks):
The following is our github actions that does the following. It creates an environment, tests application and if it fails halt workflow, creates .env for our application, and finally construct application in the form of docker images.

```yaml
CI-PIPELINE:
    runs-on: ubuntu-latest

    steps:
      # Step 1: Installs needed dependencies to set up our code
      - name: Checkout Code
        uses: actions/checkout@v3
```

```
      - name: Install Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 18

      # Step 2: Tests code too ensure it passes all tests
      - name: Run frontend tests
        run: cd course-matrix/frontend && npm install && npm run test
      - name: Run backend tests
        run: cd course-matrix/backend && npm install && npm run test

      # Step 3: Set up Docker Buildx
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      # Step 4: Sets our our application's environment
      - name: setup application env
        run: |
          cd course-matrix

          # Update frontend .env
          cd frontend
          echo "VITE_SERVER_URL=\"http://34.130.253.243:8081\"" > .env && \
          echo "VITE_PUBLIC_ASSISTANT_BASE_URL=\"${{
secrets.VITE_PUBLIC_ASSISTANT_BASE_URL }}\"" >> .env && \
          echo "VITE_ASSISTANT_UI_KEY=\"${{ secrets.VITE_ASSISTANT_UI_KEY }}\""
>> .env

          # Update backend .env
          cd ../backend
          echo "NODE_ENV=\"development\"" > .env && \
          echo "PORT=8081" >> .env && \
          echo "CLIENT_APP_URL=\"http://34.130.253.243:5173\"" >> .env && \
          echo "DATABASE_URL=\"${{ secrets.DATABASE_URL }}\"" >> .env && \
          echo "DATABASE_KEY=\"${{ secrets.DATABASE_KEY }}\"" >> .env && \
          echo "OPENAI_API_KEY=\"${{ secrets.OPENAI_API_KEY }}\"" >> .env && \
          echo "PINECONE_API_KEY=\"${{ secrets.PINECONE_API_KEY }}\"" >> .env &&
\
          echo "PINECONE_INDEX_NAME=\"course-matrix\"" >> .env && \
```

```
          echo "BREVO_API_KEY=\"${{ secrets.BREVO_API_KEY }}\"" >> .env && \
          echo "SENDER_EMAIL=\"${{ secrets.SENDER_EMAIL }}\"" >> .env && \
          echo "SENDER_NAME=\"Course Matrix Notifications\"" >> .env

          cd ../

      # Step 5: Logging in to dockerhub
      - name: Log in to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
```

2. Build a Docker image for your master branch and push it to Docker Hub. (10 marks):
Here we tag the create images with master before pushing it, ensuring that master branch only has the latest functional version of our application.

```
      # Step 10: Tags created images for the master branch
      - name: Tag Images for Master Branch
        run: |
          docker tag course-matrix/frontend:latest ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-frontend:master
          docker tag course-matrix/backend:latest ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-backend:master

      # Step 11: Push Docker images to Docker Hub master branch
      - name: Push images to Master Branch
        run: |
          docker push ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-frontend:master
          docker push ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-backend:master
```

3. Build a Docker image that includes all required dependencies and push it to Docker Hub with an appropriate version. (10 marks):
Here we create the docker images for frontend and backend, tag them with version number (which is the git push tag) and push the two to dockerhub.

```
      # Step 6: Build all required docker images
```

```
      - name: Build Docker Image
        run: |
          cd course-matrix
          docker compose build

      # Step 7: Check if images exist before tagging
      - name: List Docker Images (Debugging)
        run: docker images

      # Step 8: Tags created images with version using github commit tags
      - name: Tag Images With Version
        run: |
          docker tag course-matrix/frontend:latest ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-frontend:${{ github.sha }}
          docker tag course-matrix/backend:latest ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-backend:${{ github.sha }}

      # Step 9: Push Docker images version to Docker Hub
      - name: Push Docker images version to Docker Hub
        run: |
          docker push ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-frontend:${{ github.sha }}
          docker push ${{ secrets.DOCKERHUB_USERNAME }}/course-matrix-backend:${{
github.sha }}
```

CD Pipeline:

1. Pull the container from Docker Hub and deploy it on any containerized engine based on your preference. (10 marks):

Here we ssh connect into our virtual machine where we then pull our master branch images from docker hub and run them in our virtual machine.

```
CD-PIPELINE:
    needs: CI-PIPELINE
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Code
        uses: actions/checkout@v3

      # Step 12: Connect to virtual machine
```

```
      - name: Setup SSH Connection
        run: |
          echo "${{ secrets.GCP_SSH_PRIVATE_KEY }}" > private_key
          chmod 600 private_key

      - name: Deploy to Google Cloud VM
        run: |
          ssh -i private_key -o StrictHostKeyChecking=no ${{ secrets.GCP_USERNAME
}}@${{ secrets.GCP_VM_IP }} << 'EOF'
            cd
/home/masahisasekita/term-group-project-c01w25-project-course-matrix || { echo
"Error: Directory /root/myapp does not exist!"; exit 1; }

            # Step 13: Clears deployment environment
            sudo docker stop $(sudo docker ps -q)
            sudo docker rmi -f $(sudo docker images -q)
            sudo docker system prune -a --volumes -f

            # Step 14: Pull the latest images
            sudo docker pull ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-frontend:master
            sudo docker pull ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-backend:master

            # Step 15: Run the docker containers
            sudo docker run -d -p 5173:5173 ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-frontend:master
            sudo docker run -d -p 8081:8081 ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-backend:master
```

2. Set up automated testing for the deployed container to ensure its functionality and performance. (10 marks):

Here we first run docker ps to test if our images are actually deployed and running. We then run our jest tests again to ensure our deployed application is running.

```
            # Step 16: Run post deployment tests
            sudo docker ps
            sudo docker run --rm ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-frontend:master npm test
```

```
            sudo docker run --rm ${{ secrets.DOCKERHUB_USERNAME
}}/course-matrix-backend:master npm test
```