DevOps Document

Name: Tony Jiang

Semester: 6

Class: RB04

Version	Date	Description
0.1	28 Oct 24	Initial document
1	8 Dec 24	Add version 2
1.1	19 Jan 25	Add version 3

Contents

Introduction 3
CI/CD overview 3
    V3 3
    V2 3
    V1 4
Actual CI/CD pipeline 5

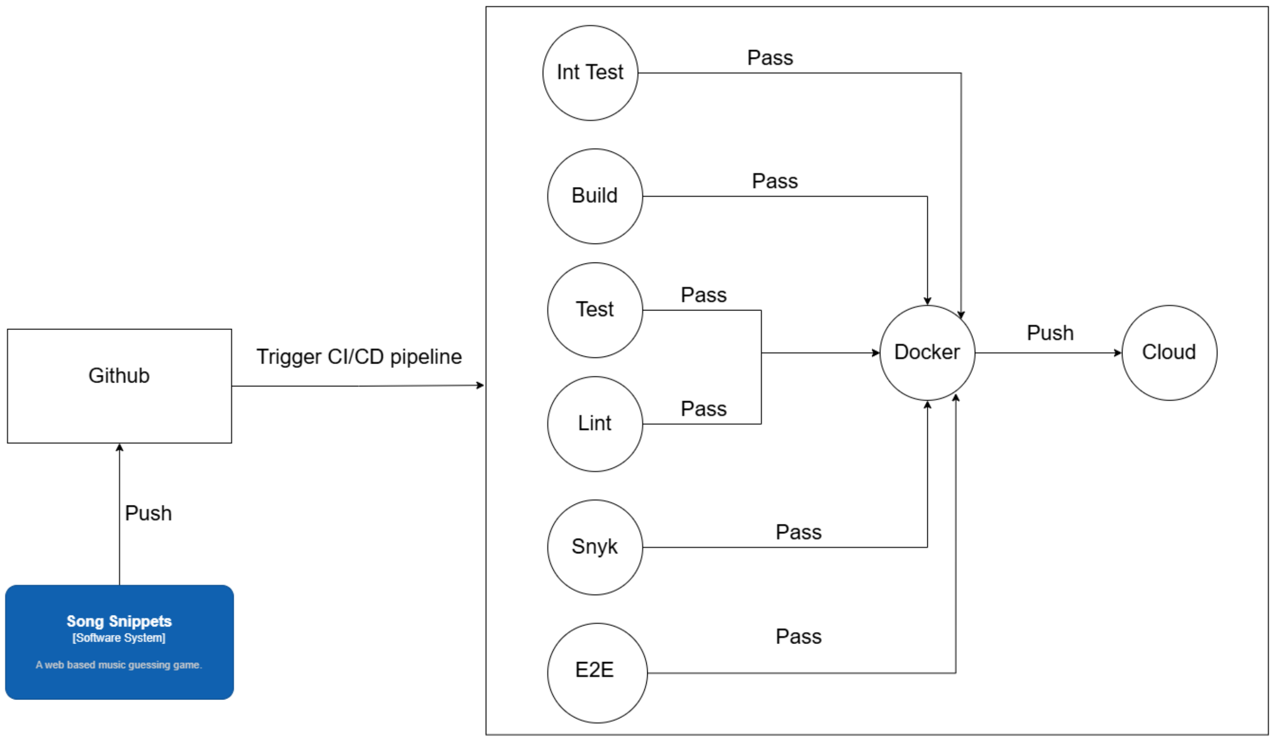
Introduction

This document contains the CI/CD pipeline, how it is set up, and the reasons for its configuration.

CI/CD overview

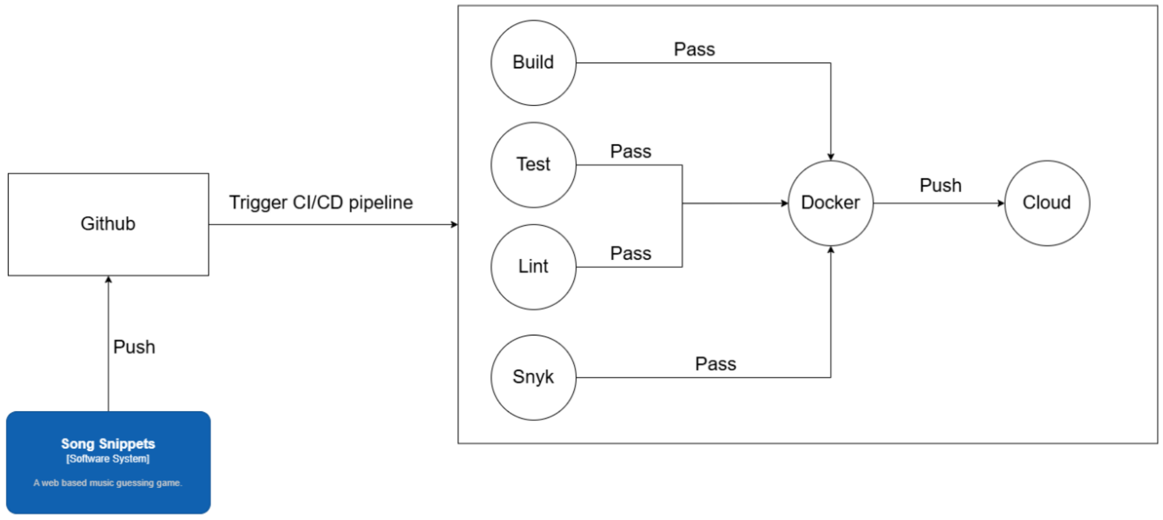
This document provides an overview of the CI/CD pipeline for the project. It explains how the pipeline is set up, implemented, and versioned, leading to the final product version.

V3



In this version, I added integration tests to verify interactions between services, end-to-end tests to validate service endpoints by testing both happy flows and bad flows, and finally, deployment. After publishing the project on Docker Hub, it is automatically deployed to the cloud.

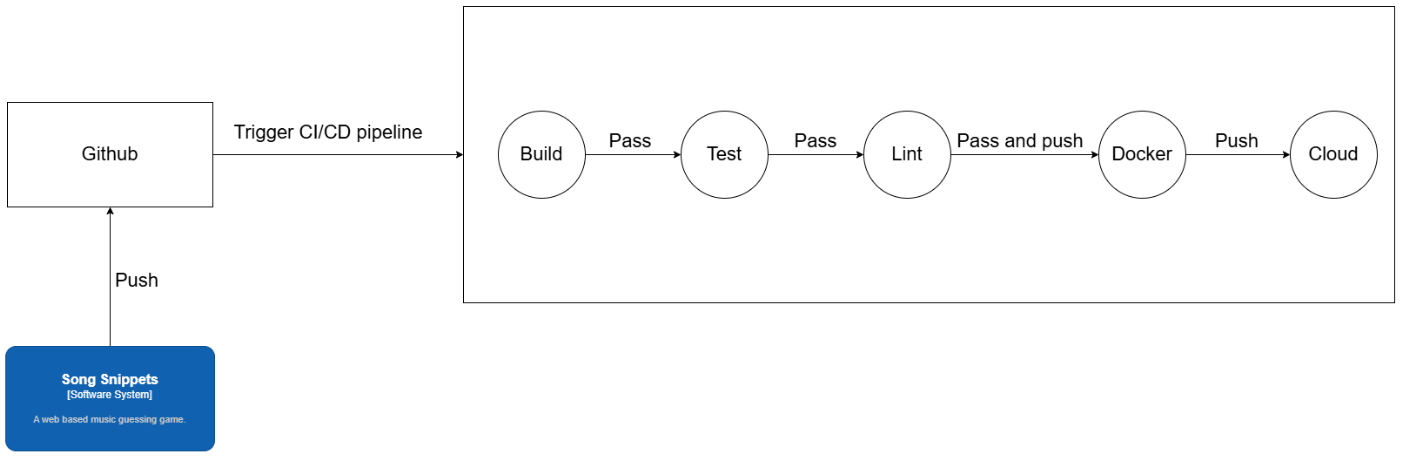
V2



In this version, the design structure is significantly different from the previous one. Build, Test, Lint, and Snyk now run in parallel, making the CI/CD pipeline faster compared to executing them sequentially. If any of these steps fail, the process will not proceed to Docker. If all four steps pass, Docker is executed, and the build is pushed to the cloud.

The Build task compiles the project to ensure there are no build issues. The Test task runs the unit tests to verify that no tests fail. Lint checks the code for any code smells and ensures the project follows best coding practices. Snyk scans for security vulnerabilities and outdated dependencies. Docker builds the Docker image, publishes it to Docker Hub and finally Cloud, It for deploying to project to cloud.

V1

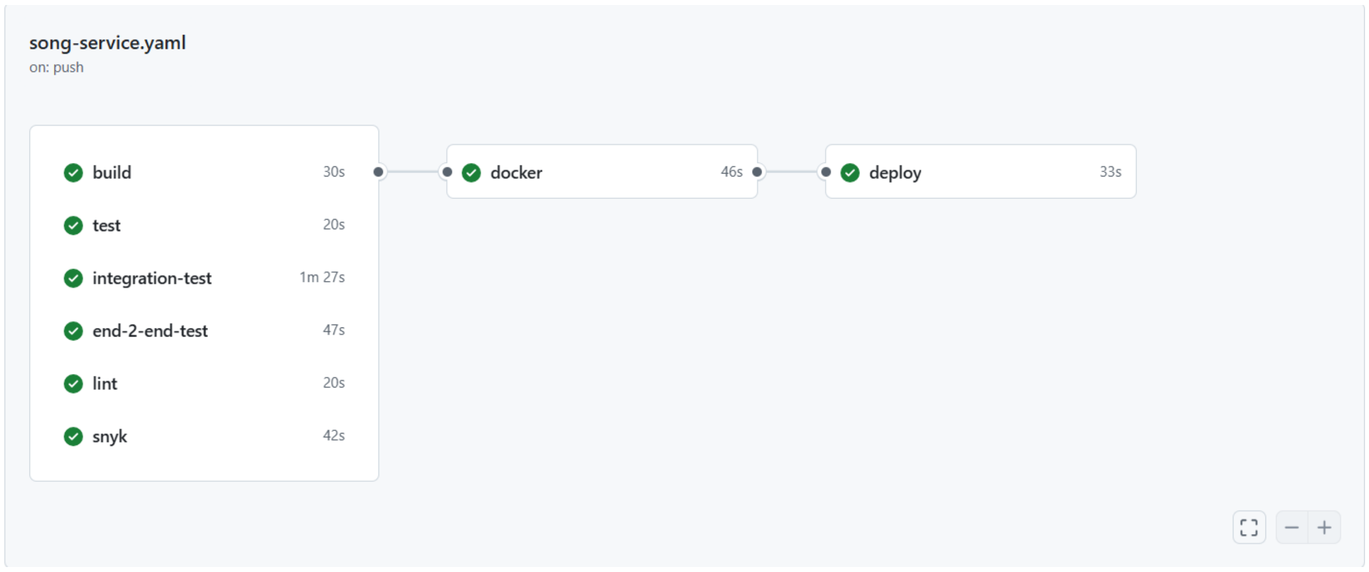


This pipeline is a work in progress and will be enhanced as further research is conducted. The project repository is hosted on GitHub, and GitHub Actions is used to manage the CI/CD pipeline.

The pipeline is triggered whenever changes are pushed to the GitHub repository, regardless of the branch. It begins by building the project and checking for any issues. Next, it runs the existing unit tests to evaluate functionality coverage. Once the tests pass, the pipeline lints the project to identify code issues, code smells, and ensure adherence to best practices.

If all checks are successful, the pipeline pushes the build to Docker and subsequently deploys it to the cloud.

Actual CI/CD pipeline



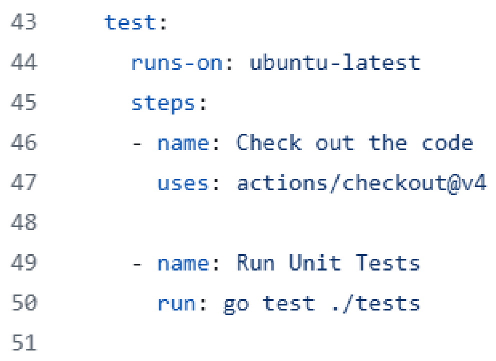
This is the actual CI/CD pipeline for the song service. It follows the CI/CD pipeline design that I planned. All tasks in the CI/CD pipeline were successfully completed. Below is the script for the CI/CD pipeline.

```yaml
name: Song service

on:
  push:
    branches:
      - development
  pull_request:
    branches:
    - development

```

The first task is to build the project to ensure it compiles successfully and to identify any issues during the build process.

```yaml
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    # Check out the code
    - name: Check out the code
      uses: actions/checkout@v4

    # Setup Go
    - name: Set up Go
      uses: actions/setup-go@v5
      with:
        go-version: '1.23.1' # Specify your Go version

    # Cache Go modules
    - name: Cache Go modules
      uses: actions/cache@v4
      with:
        path: |
              ~/.cache/go-build
              ${{ runner.tool_cache }}/go
        key: go-${{ runner.os }}-${{ hashFiles('**/go.sum') }}
        restore-keys: |
          ${{ runner.os }}-go-

    # Install dependencies
    - name: Install dependencies
      run: go mod download

    - name: Build the application
      run: go build -v ./cmd/song-service/

```

The second task is to run the unit tests to check if there are any changes in the functionalities. This is helpful because it eliminates the need for manual testing and alerts you when you need to update the tests or when changes to the functionalities introduce new issues.

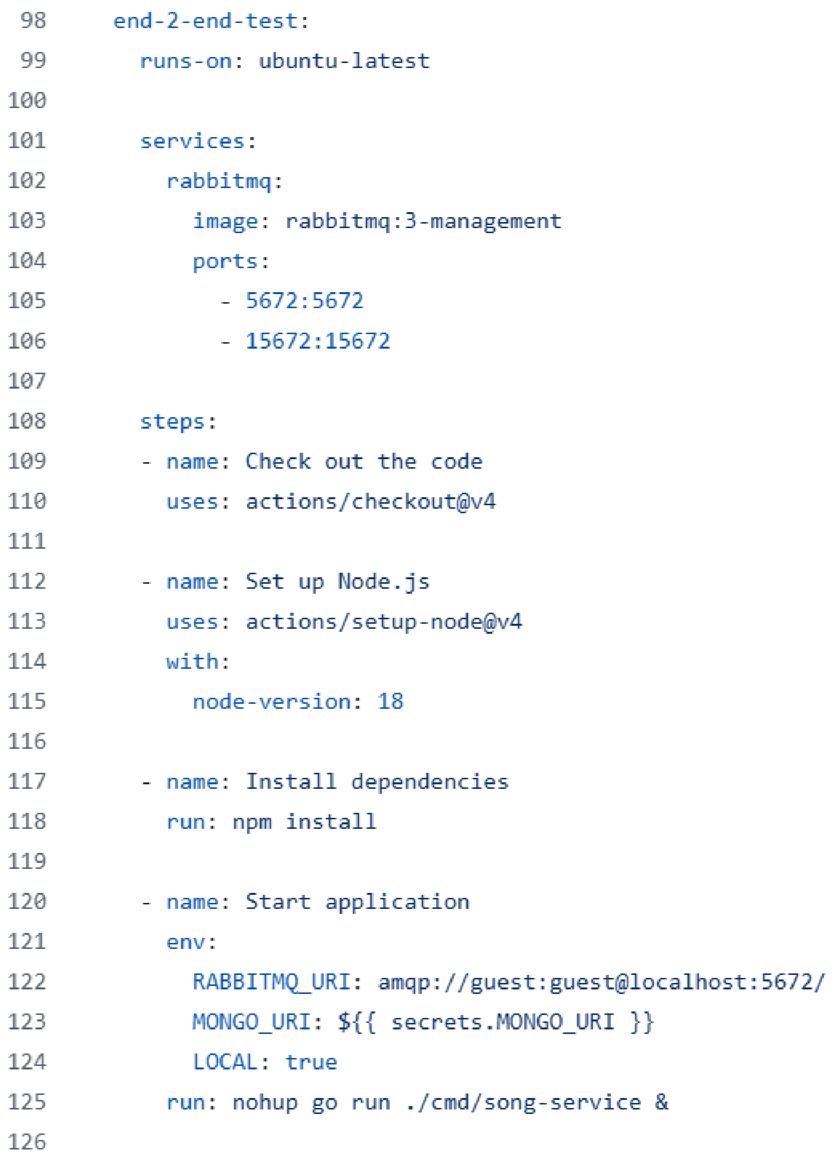


```
  test:
    runs-on: ubuntu-latest
    steps:
    - name: Check out the code
      uses: actions/checkout@v4

    - name: Run Unit Tests
      run: go test ./tests

```

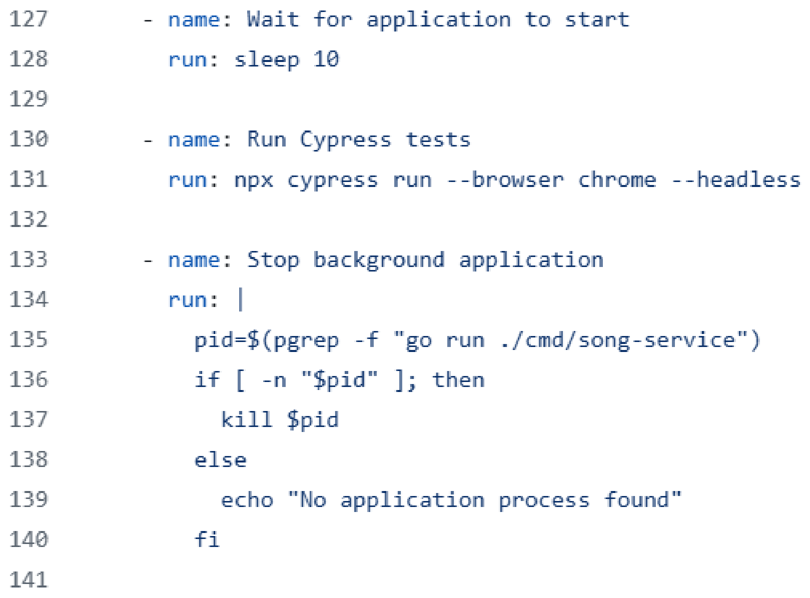
The integration test is used to test communication with other services or databases. Communication with other services can occur via REST API or message queue. In my case, I test the integration with other services using a message queue, specifically RabbitMQ. I need to mock both the database and RabbitMQ, and I use test containers for that. For the integration test, I focus on testing the data that is published. Additionally, the application has to run in the CI/CD pipeline for it to execute the test. It’s good to have the integration test so you don’t have to manually tested every time, it can be tested automatically.

```yaml
  integration-test:
    runs-on: ubuntu-latest

    services:
      rabbitmq:
        image: rabbitmq:3-management
        ports:
          - 5672:5672
          - 15672:15672

    steps:
    - name: Check out the code
      uses: actions/checkout@v4

    - name: Set up Go
      uses: actions/setup-go@v5
      with:
        go-version: '1.23.1'

    - name: Install dependencies
      run: go mod download

    - name: Start application in the background
      env:
        RABBITMQ_URI: amqp://guest:guest@localhost:5672/
        MONGO_URI: ${{ secrets.MONGO_URI }}
        LOCAL: true
      run: nohup go run ./cmd/song-service &

    - name: Wait for application to start
      run: sleep 10

    - name: Run integration tests
      env:
        RABBITMQ_URI: amqp://guest:guest@localhost:5672/
      run: go test -timeout 300s -run ^TestCreateSongPublishIntegration$ github.com/TonyJ3/song-service/integration_test

    - name: Stop background application
      run: |
        pid=$(pgrep -f "go run ./cmd/song-service")
        if [ -n "$pid" ]; then
          kill $pid
        else
          echo "No application process found"
        fi

```

In the end-to-end test, I test the endpoint of my create song API. This test automatically checks the endpoint to ensure it returns the expected result. Also, I have to run the application to test the end to end and this can be done in the pipeline.

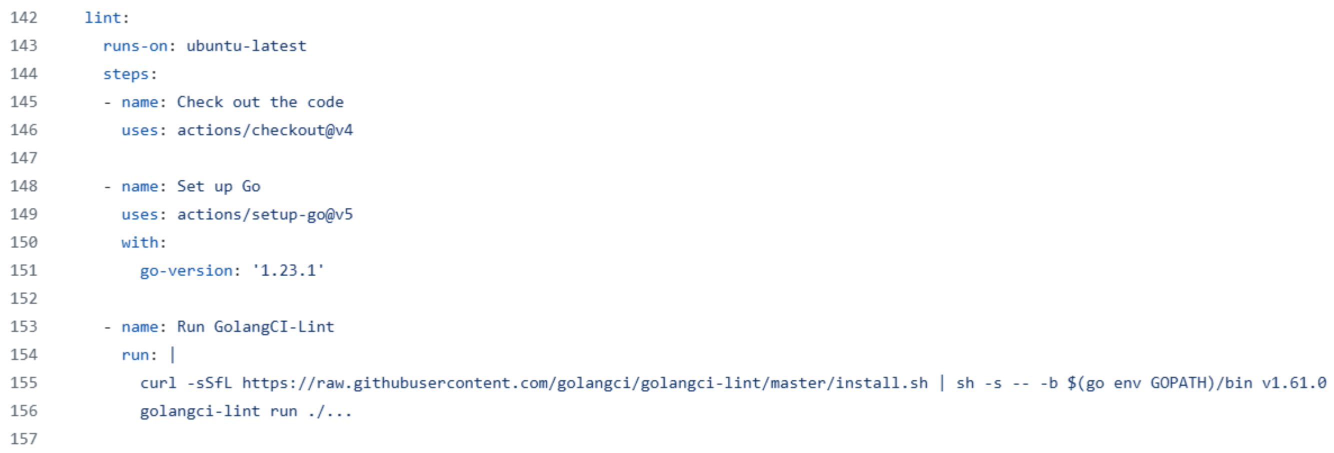


```yaml
  end-2-end-test:
    runs-on: ubuntu-latest

    services:
      rabbitmq:
        image: rabbitmq:3-management
        ports:
          - 5672:5672
          - 15672:15672

    steps:
    - name: Check out the code
      uses: actions/checkout@v4

    - name: Set up Node.js
      uses: actions/setup-node@v4
      with:
        node-version: 18

    - name: Install dependencies
      run: npm install

    - name: Start application
      env:
        RABBITMQ_URI: amqp://guest:guest@localhost:5672/
        MONGO_URI: ${{ secrets.MONGO_URI }}
        LOCAL: true
      run: nohup go run ./cmd/song-service &

```



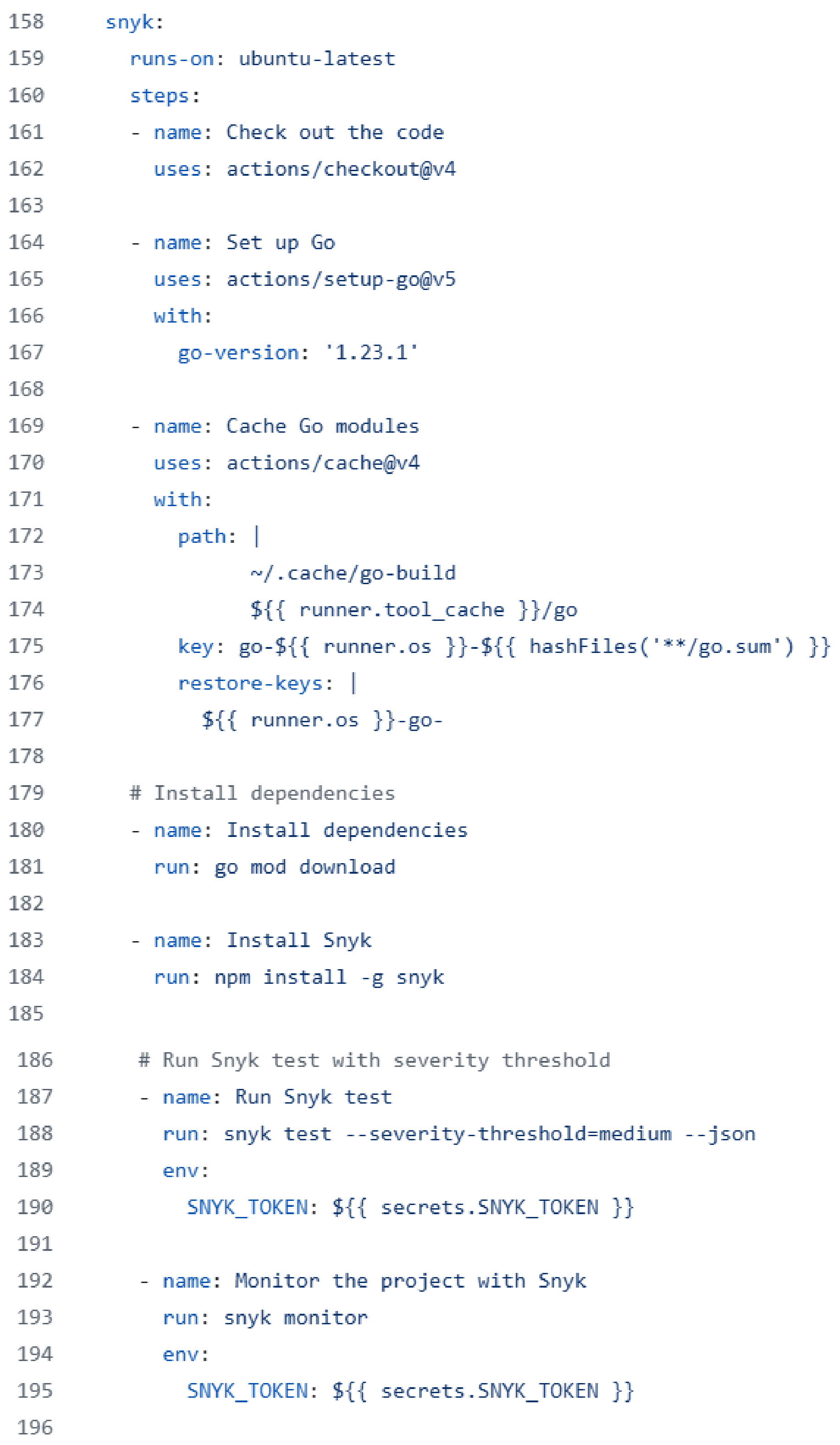
```yaml
      - name: Wait for application to start
        run: sleep 10

      - name: Run Cypress tests
        run: npx cypress run --browser chrome --headless

      - name: Stop background application
        run: |
          pid=$(pgrep -f "go run ./cmd/song-service")
          if [ -n "$pid" ]; then
            kill $pid
          else
            echo "No application process found"
          fi

```

Linting is used to enforce coding best practices. It checks your entire codebase to ensure it follows best practices and identifies any code smells.



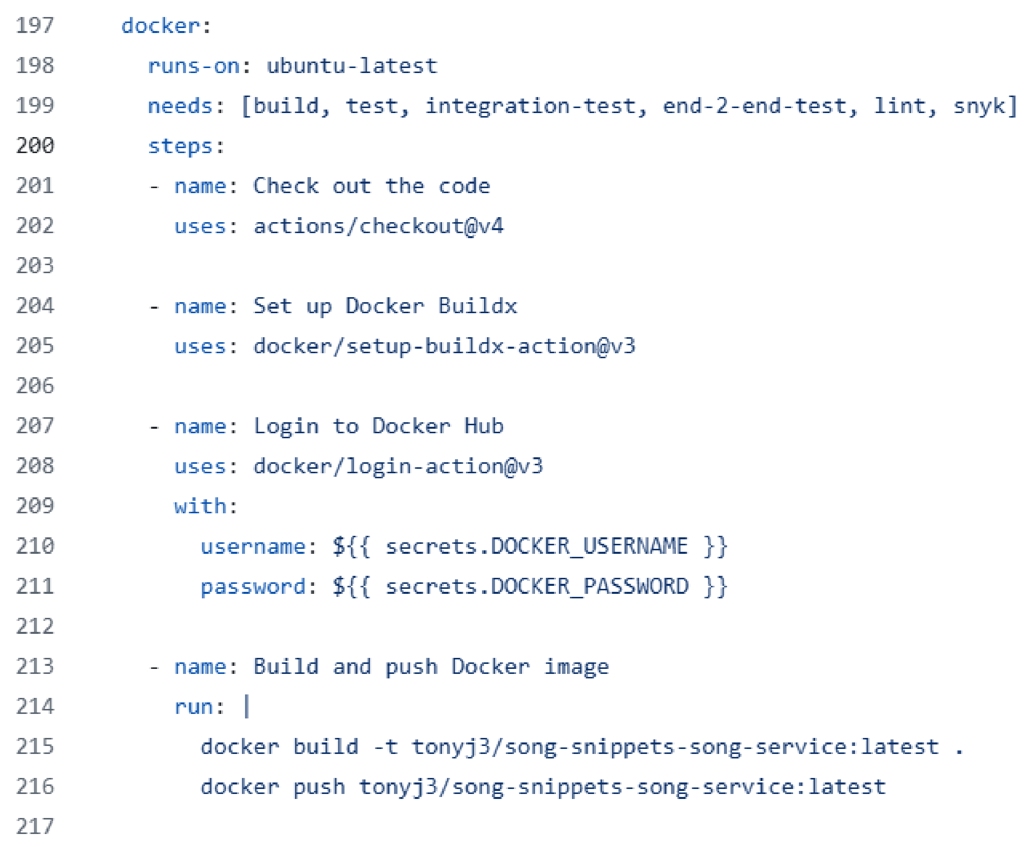
```yaml
  lint:
    runs-on: ubuntu-latest
    steps:
    - name: Check out the code
      uses: actions/checkout@v4

    - name: Set up Go
      uses: actions/setup-go@v5
      with:
        go-version: '1.23.1'

    - name: Run GolangCI-Lint
      run: |
        curl -sSfL https://raw.githubusercontent.com/golangci/golangci-lint/master/install.sh | sh -s -- -b $(go env GOPATH)/bin v1.61.0
        golangci-lint run ./...

```

Snyk is used for checking any outdated dependencies in the application.



```yaml
  snyk:
    runs-on: ubuntu-latest
    steps:
    - name: Check out the code
      uses: actions/checkout@v4

    - name: Set up Go
      uses: actions/setup-go@v5
      with:
        go-version: '1.23.1'

    - name: Cache Go modules
      uses: actions/cache@v4
      with:
        path: |
              ~/.cache/go-build
              ${{ runner.tool_cache }}/go
        key: go-${{ runner.os }}-${{ hashFiles('**/go.sum') }}
        restore-keys: |
          ${{ runner.os }}-go-

    # Install dependencies
    - name: Install dependencies
      run: go mod download

    - name: Install Snyk
      run: npm install -g snyk

    # Run Snyk test with severity threshold
    - name: Run Snyk test
      run: snyk test --severity-threshold=medium --json
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}

    - name: Monitor the project with Snyk
      run: snyk monitor
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}

```

The Docker task builds the Docker image and pushes it to Docker Hub once all tasks are completed, from the build to the Snyk task.



```yaml
  docker:
    runs-on: ubuntu-latest
    needs: [build, test, integration-test, end-2-end-test, lint, snyk]
    steps:
    - name: Check out the code
      uses: actions/checkout@v4

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Login to Docker Hub
      uses: docker/login-action@v3
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}

    - name: Build and push Docker image
      run: |
        docker build -t tonyj3/song-snippets-song-service:latest .
        docker push tonyj3/song-snippets-song-service:latest

```

After the Docker task is completed, the application is deployed. In my case, I'm using AWS Lambda functions. I deploy the “create song” function to AWS Lambda, ensuring that it is updated every time this pipeline is executed.

```yaml
deploy:
 runs-on: ubuntu-latest
 needs: docker
 steps:
 - name: Check out the code
   uses: actions/checkout@v4

 - name: Configure AWS CLI
   uses: aws-actions/configure-aws-credentials@v3
   with:
      aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
      aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      aws-region: eu-central-1

 - name: Build Lambda Function
   run: |
        export GOOS=linux
        export GOARCH=arm64
        export CGO_ENABLED=0
        go build -o bootstrap ./cmd/song-service/main.go
        zip create-song.zip bootstrap

 - name: Deploy to AWS Lambda
   run: aws lambda update-function-code --function-name CreateSong --zip-file fileb://create-song.zip
```