Joakim Edvardsen

CI/CD Pipeline

Table of Contents

Motivation

Today, when working with a software product, you often want a fully functional and automated pipeline that works from code commit to production/test environment. But what are the motivations behind this?

- By having a working pipeline you can be more confident that every feature of your software product that is pushed to production is tested and working correctly.

- A pipeline makes it much easier to work with continuous integration and continuous delivery (CI/CD) by having everything automated. Everything a developer has to do when he is finished implementing a new feature, is to press a single button that will trigger the pipeline. When the pipeline is complete, the developer and every one else on the team will either get confirmation that everything is still working or that something broke an has to be fixed.

- And one of the main reasons we want to have CI/CD is to rappidly deliver new features and refactor our code to make it better and more efficient. This will also yield in more rappidly feedback from the end user.

Side note: *When to push new features to production should often be a business decision. That's why for most pipelines you dont want to automatically deploy it to production. Rather to a test environment where it can be tested to confirm everything is working and have a working product with all the new features on stand-by ready to be deploy to production when the decision is made.*

Tools - Which tools and why

- Spring boot: To create a basic web application that can be deployed and tested.

- Sonar Cloud: Analyze the project for security, code smells, test coverage etc... Gives good feedback on the code with a clear dashboard.
- Docker: Containerize the application, both the spring boot web app but also the postgreSQL database the spring boot app should be connected to.
- GitHub Actions: I'm a big fan of GitHub an use it for every project I make. Wanted to explore more of what it has to offer.
- Azuer: Production environment to host the application. Tested multiple cloud service providers earlier in the semester and ended up liking azure the most. Has a clean dashboard, ease to understand walk-throughs when setting up resources and a lot of documentation for when you are stuck.

Pipeline Location

The pipeline file are store under `.github/workflows/build.yml`

Pipeline Triggers

The pipeline is triggered whenever a change is pushed to the main branch. You can also configure it to be triggered whenever a merge request to the main branch is opened. This is ususally how teams work, by creating a new branch where a single developer can work on a singel new feature. When it's implemented, the developer opens up a merge requrest to the main branch, which triggers the pipeline as well as other developers can review the new changes.

Since I've worked on this projec alone, I didn't find any reasons to work like this. Instead I just pushed every change direclty to the main branch.

```
on:
  push:
    branches:
      - main
```

Pipeline trigger config

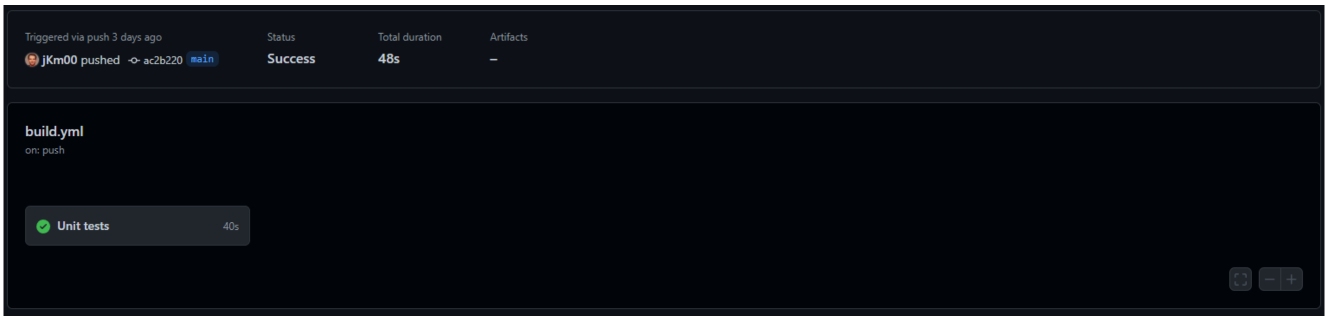
Pipeline Jobs

The pipeline consits of 4 jobs:

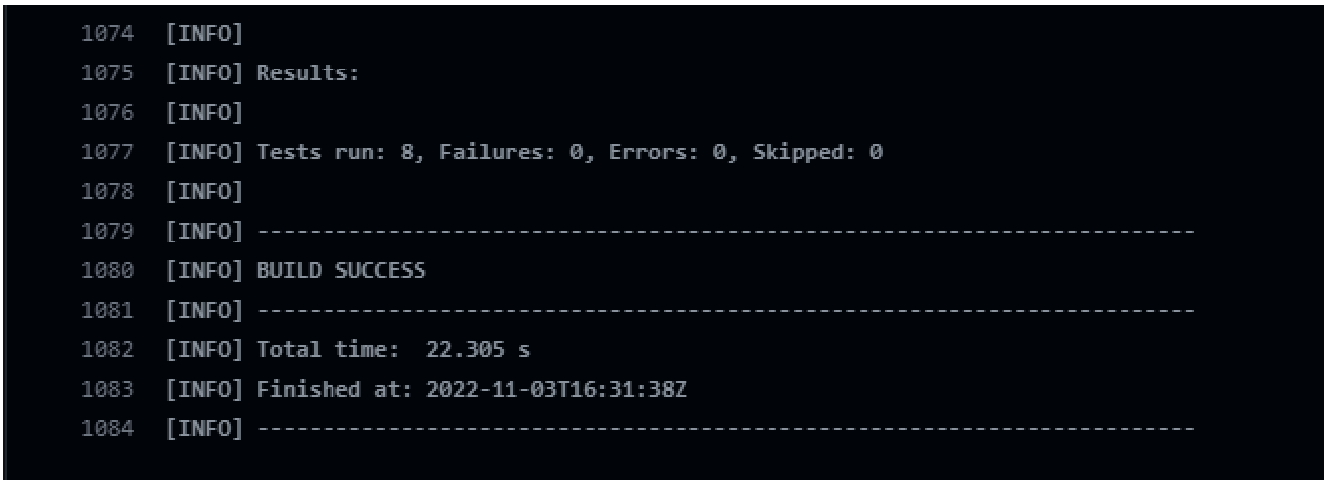
1. Unit testing
2. Sonar Cloud analyzing
3. Building
4. Deploying

Unit test

The unit test job makes sure all the unit test specified in the test folder of the application is executed and the job is succesfull only if all the unit test passes. If not the pipeline is canceled with logs of which test failed.



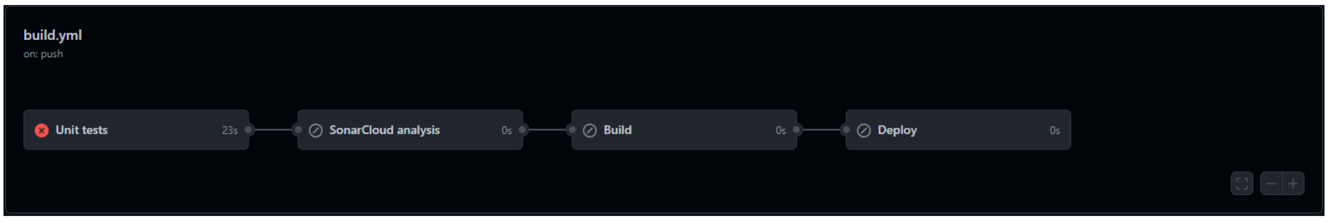
Job: Unit testing, succesfull!



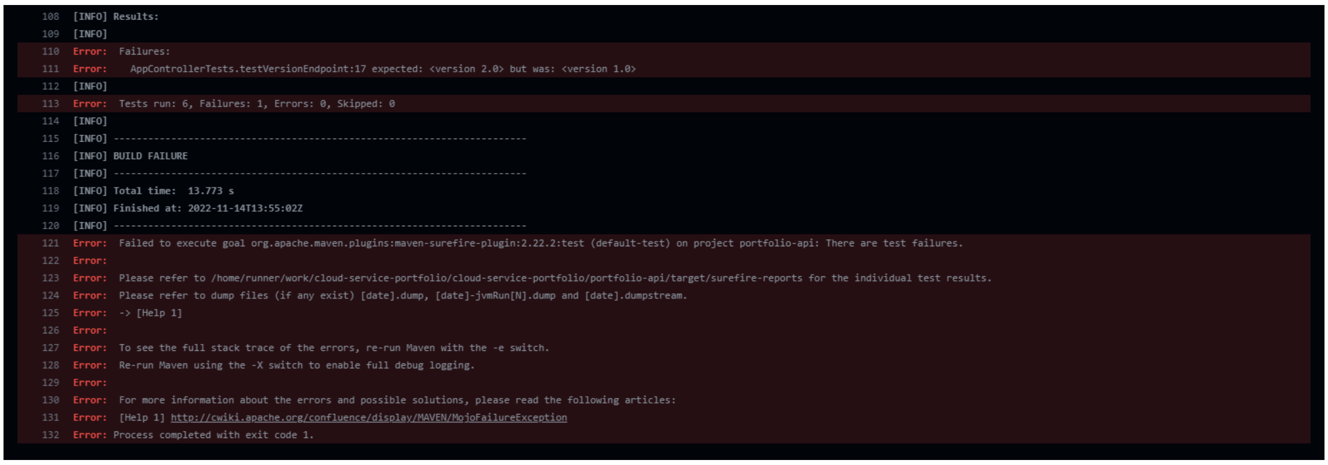
```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  22.305 s
[INFO] Finished at: 2022-11-03T16:31:38Z
[INFO] ------------------------------------------------------------------------
```

Test results fron unit test job

If one or more tests fails, the pipeline is cancelled and the test result is logged in the `test` step of the pipeline. You can navigate your way to it and have a look at what went wrong to make it easier to pin-point what needs to be fixed.



Pipeline canceled when on or more tests fails



```
[INFO] Results:
[INFO]
Error:  Failures:
Error:    AppControllerTests.testVersionEndpoint:17 expected: <version 2.0> but was: <version 1.0>
[INFO]
Error:  Tests run: 6, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  13.773 s
[INFO] Finished at: 2022-11-14T13:55:02Z
[INFO] ------------------------------------------------------------------------
Error:  Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.2:test (default-test) on project portfolio-api: There are test failures.
Error:  
Error:  Please refer to /home/runner/work/cloud-service-portfolio/cloud-service-portfolio/portfolio-api/target/surefire-reports for the individual test results.
Error:  Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date].dumpstream.
Error:  -> [Help 1]
Error:  
Error:  To see the full stack trace of the errors, re-run Maven with the -e switch.
Error:  Re-run Maven using the -X switch to enable full debug logging.
Error:  
Error:  For more information about the errors and possible solutions, please read the following articles:
Error:  [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
Error: Process completed with exit code 1.
```

Log output when some tests fails

```yaml
tests:
  name: Unit tests
  runs-on: ubuntu-latest
  steps:
    # Check out repo
    - uses: actions/checkout@v1
    # Set up JDK
    - name: Set up JDK
      uses: actions/setup-java@v1
      with:
        java-version: '17'
    # Set up maven
    - name: Cache Maven packages
      uses: actions/cache@v1
      with:
        path: ~/.m2
        key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
        restore-keys: ${{ runner.os }}-m2
    # Run tests
    - name: Run Tests
      run: mvn -B test
      working-directory: ./portfolio-api
```

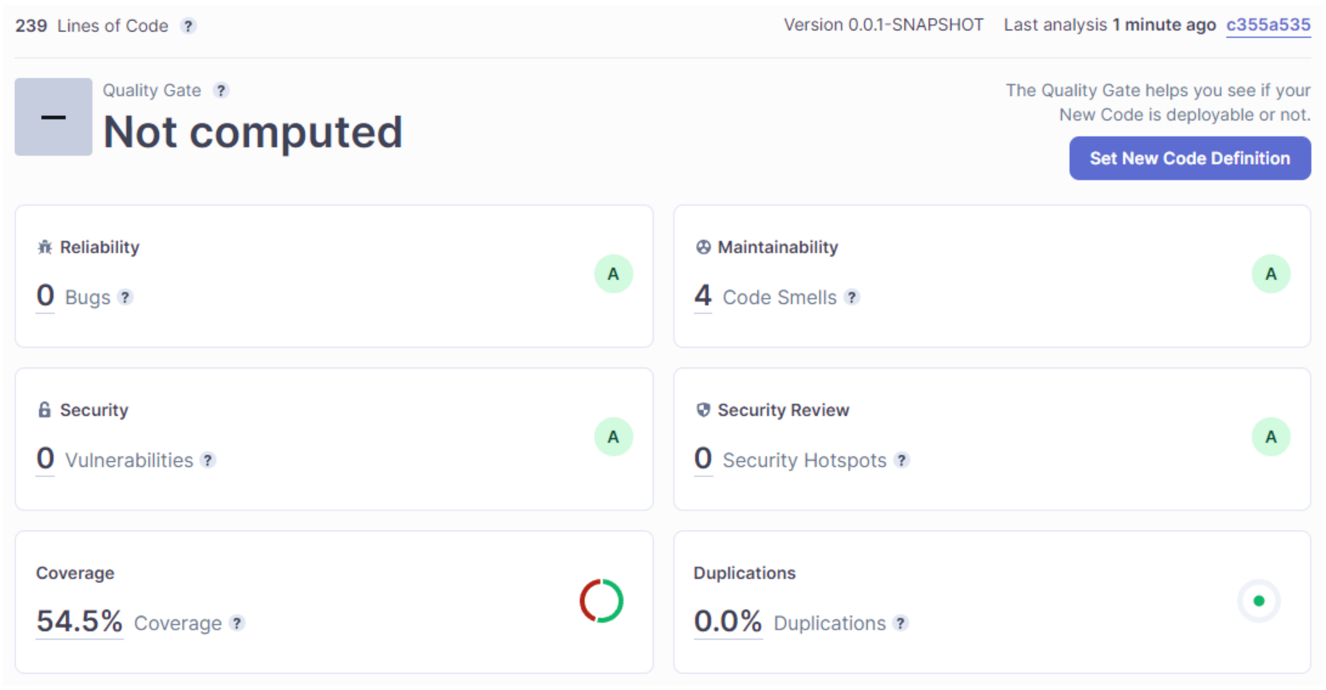
Pipeline config for unit testing

For the unit test job, an ubunt VM is booted up, a JDK is downloaded and initialized as well as maven, before the application is executed with the maven test command.

***Note:** Need to specify working directory when executing the tests because the appliaction does not live in the root of the repository.*

Sonar Cloud analyzing

Runs the appliaction with Sonar Cloud to analyze the project and given feedback on bugs, security issues, test coverage, maintainability, etc... A summary of the appliaction state could be found on by logging into Sonar Cloud.


239 Lines of Code ? Version 0.0.1-SNAPSHOT Last analysis 1 minute ago c355a535

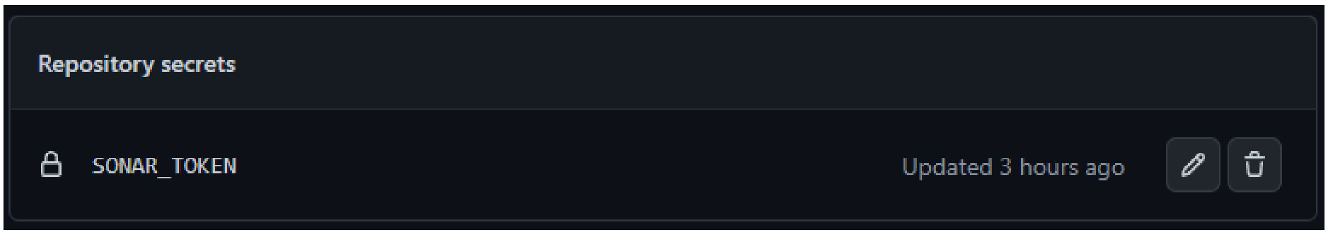
Quality Gate ? — Not computed

The Quality Gate helps you see if your New Code is deployable or not.

Reliability	0 Bugs	A
Maintainability	4 Code Smells	A
Security	0 Vulnerabilities	A
Security Review	0 Security Hotspots	A
Coverage	54.5% Coverage	
Duplications	0.0% Duplications	

Sonar Cloud Dashboard

For github to get access to Sonar Cloud a token is stored as a repository secret: `settings -> secrets -> actions`. This token is used directly in the pipeline job. Only admins of the repository have access to read, edit or delete this the token.


Repository secrets

SONAR_TOKEN Updated 3 hours ago

Sonar Cloud token stored as repository secret

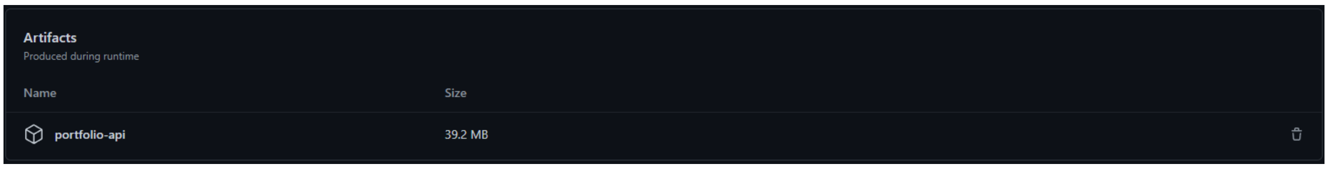
```
sonar:
  # Make sure test job is succesfull
  needs: tests
  name: SonarCloud analysis
  runs-on: ubuntu-latest
  # Step's required for the job
  steps:
    # Check-out repository
    - uses: actions/checkout@v2
      with:
        fetch-depth: 0 # Shallow clones should be disabled for a better relevancy
of analysis
    # Sets up JDK
    - name: Set up JDK
      uses: actions/setup-java@v1
      with:
        java-version: '17'
    # Sets up SonarCloud cache
```

```
      - name: Cache SonarCloud packages
        uses: actions/cache@v1
        with:
          path: ~/.sonar/cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
      # Sets up Maven cache
      - name: Cache Maven packages
        uses: actions/cache@v1
        with:
          path: ~/.m2
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
          restore-keys: ${{ runner.os }}-m2
      # Uses SonarCloud to analyze the project
      - name: Build and analyze
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }} # Needed to get PR information,
if any
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        run: mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -
Dsonar.projectKey=jKm00_cloud-service-portfolio -Dspring.profiles.active=test
        working-directory: ./portfolio-api
```

Pipeline config for Sonar Cloud analyze

Building

Before the app can be deployed, it needs to be built. That's what this job does. After the job has build the application, the `.jar` file is uploaded as an artifact, with name `portfolio-api`, so it can be shared across the pipeline jobs.



Name	Size
portfolio-api	39.2 MB

Uploaded artifact by pipeline job *(can be downloaded and executed)*

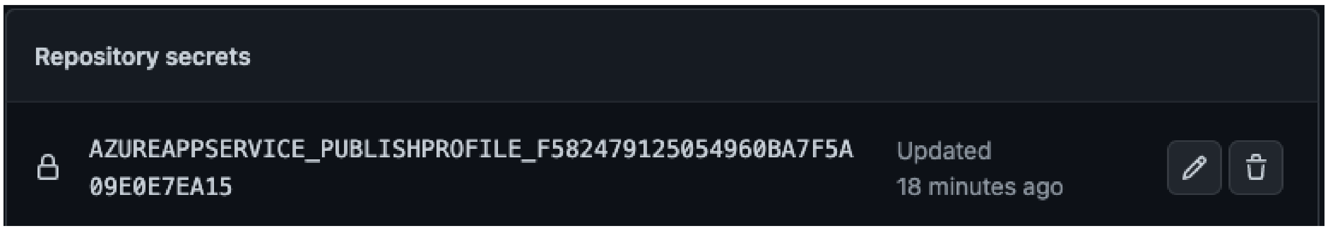
```
build:
  # Make sure sonar job is succesfull
  needs: sonar
  name: Build
  runs-on: ubuntu-latest
  steps:
    #Check-out repository
    - uses: actions/checkout@v1
    #Set up JDK
    - name: Set up JDK
      uses: actions/setup-java@v1
      with:
        java-version: '17'
    #Set up Maven cache
    - name: Cache Maven packages
      #This action allows caching dependencies and build outputs to improve
workflow execution time.
      uses: actions/cache@v1
      with:
        path: ~/.m2
        key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
        restore-keys: ${{ runner.os }}-m2
    #Build the application using Maven
    - name: Build with Maven
      run: mvn -B package -DskipTests --file pom.xml
      working-directory: ./portfolio-api
    # Upload build version of application
    - name: Upload JAR
      #This uploads artifacts from your workflow allowing you to share data
between jobs and store data once a workflow is complete.
      uses: actions/upload-artifact@v2
      with:
        name: portfolio-api
        #From this path
        path: portfolio-api/target/portfolio-api-1.0.jar
```

Pipeline config for building

Deploying

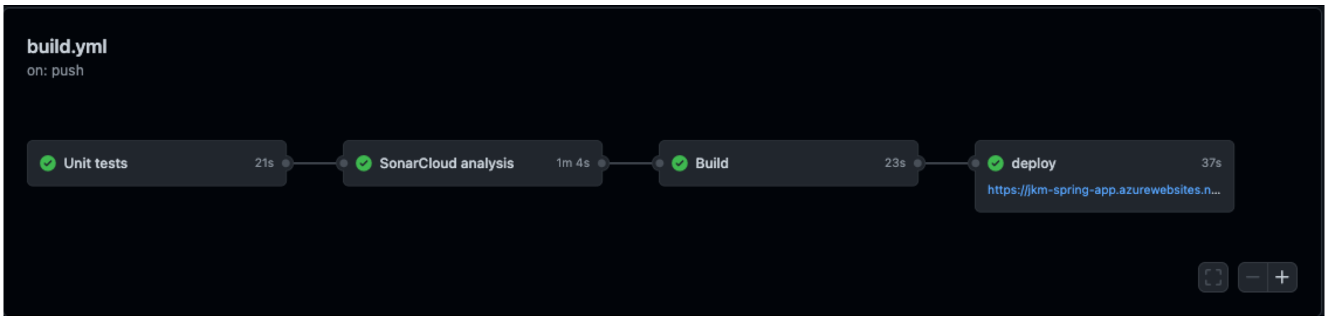
The job responsible for deploying the spring boot application from the repository to the azure web app. Uses the artifact generated from the previous job (build) and uploads it to the azure web app before it's ran.

For github to get access to the azuer web application another secret has to be stored in the repository:

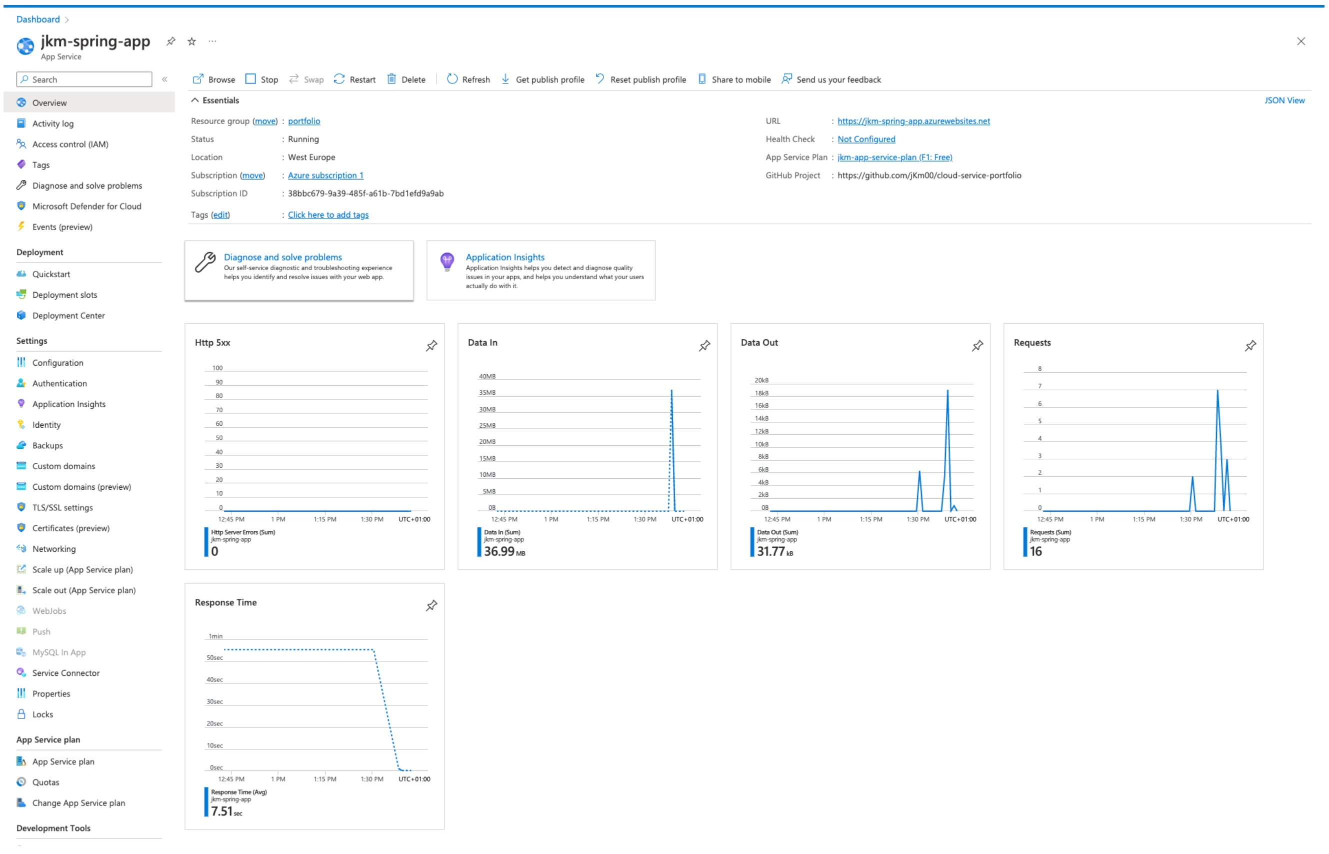

Repository secrets

AZUREAPPSERVICE_PUBLISHPROFILE_F582479125054960BA7F5A09E0E7EA15 Updated 18 minutes ago

Azure secret stored in repository


build.yml
on: push
Unit tests 21s
SonarCloud analysis 1m 4s
Build 23s
deploy 37s
https://jkm-spring-app.azurewebsites.n...

Succesfull Pipeline with Azure Deployment



Essentials

Resource group (move) : portfolio
Status : Running
Location : West Europe
Subscription (move) : Azure subscription 1
Subscription ID : 38bbc679-9a39-485f-a61b-7bd1efd9a9ab
Tags (edit) : Click here to add tags

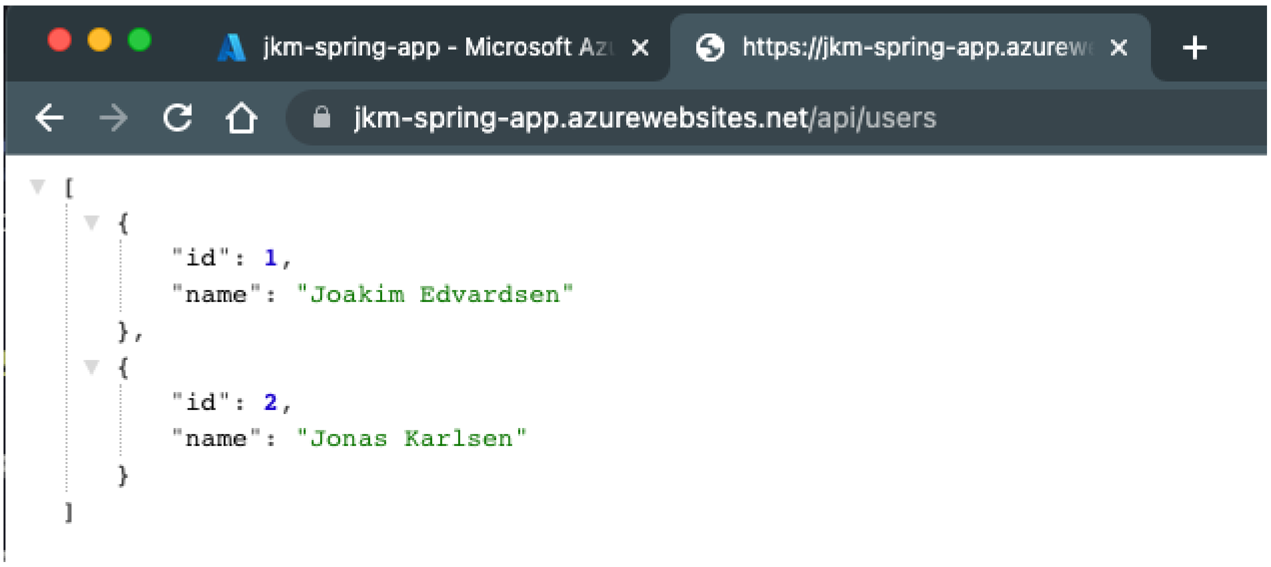
URL : https://jkm-spring-app.azurewebsites.net
Health Check : Not Configured
App Service Plan : jkm-app-service-plan (F1: Free)
GitHub Project : https://github.com/jKm00/cloud-service-portfolio

Diagnose and solve problems
Our self-service diagnostic and troubleshooting experience helps you identify and resolve issues with your web app.

Application Insights
Application Insights helps you detect and diagnose quality issues in your apps, and helps you understand what your users actually do with it.

Http Server Errors (Sum) jkm-spring-app 0
Data In (Sum) jkm-spring-app 36.99 MB
Data Out (Sum) jkm-spring-app 31.77 kB
Requests (Sum) jkm-spring-app 16
Response Time (Avg) jkm-spring-app 7.51 sec

Azure Wen App Overview



jkm-spring-app.azurewebsites.net/api/users

```
[
  {
    "id": 1,
    "name": "Joakim Edvardsen"
  },
  {
    "id": 2,
    "name": "Jonas Karlsen"
  }
]
```

Accessing api endpoint from azure web app

```
deploy:
  needs: build
  name: Deploy
  runs-on: ubuntu-latest
  environment:
    name: 'Production'
    url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}

  steps:
    - name: Download artifact from build job
```

```
        uses: actions/download-artifact@v2
        with:
          name: portfolio-api

      - name: Deploy to Azure Web App
        id: deploy-to-webapp
        uses: azure/webapps-deploy@v2
        with:
          app-name: 'jkm-spring-app'
          slot-name: 'Production'
          publish-profile: ${{
secrets.AZUREAPPSERVICE_PUBLISHPROFILE_F582479125054960BA7F5A09E0E7EA15 }}
          package: '*.jar'
```

Pipeline config for deployment

Iterations / Experiences

1. First I created the basic spring boot applicaiton with some endpoints. I wanted to connect the application to a database to challenge myself to have more than one service. In the beginning I created the app with an in memory database with the tought of moving this to a postgreSQL later in production. I would anyways need the in memory database configuration for testing to make sure I would have consistant test results.

2. Next I started creating the pipeline. The first thing I wanted to complete was automated testing. For this I also needed to implement some unit test that would be run in the pipeline. The pipeline configuration was pretty straight forward. Just needed to setup a VM where I could execute all the tests.

3. After doing some research, I found an iteresting tool, Sonar Cloud, that we were introduced to in the first year, but a tool that I had forgot about. After rediscovering this I wanted to implement it in the pipeline to get automated analyzing of the applicaiton. There were some challenges with this step like getting a token and storing it as a secret as well as actually using that token in the pipeline itself.

4. The next and final step I wanted to complete was deploying the application, however I needed to build the application first. So this became a naturall step by itself. The main challenge with this step were to figure out how to store the built applicaiton so I could use it later to deploy.

5. Now I could takle the finally step, deploying. I had tested some cloud providers earlier in the semester and found that I liked azure the most. That's why I ended up with using azure for this portfolio as well. Before I could configure the pipeline to automatically deploy, I had to configure a web application in azure that I could deploy my app to. This was done using azures portal interface. When this was created, azure automatically genereted a workflow file that I could just merge into my own.

One extremly good reasen to use azure is their documentation. I had some challeneges along the way, but looking it up always yeilded a result where my pipeline ended up with a green checkmark.

Everything was done with version controll, meaning you can go to the github repository and view all the commits along the way as well as a history of all the executed workflows under the actions tab.

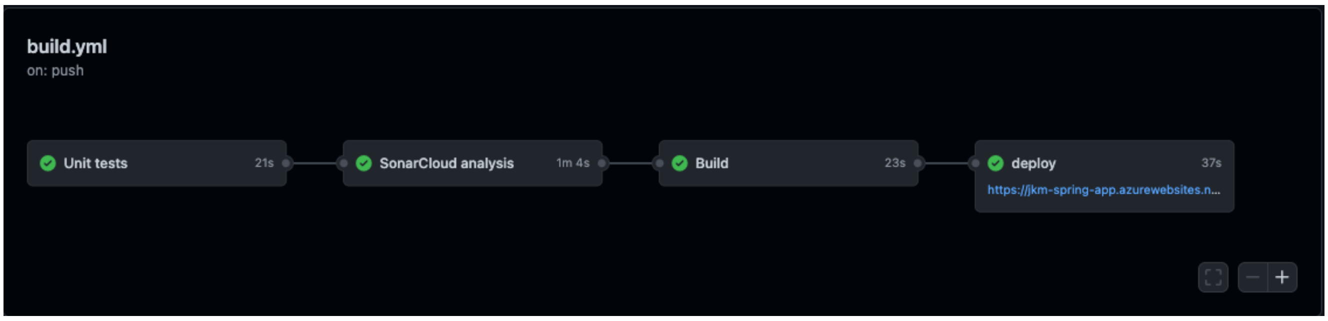
Further Works

With more time working with this pipeline, I would have containerized both the spring boot application and a postgreSQL database and ran it using a `docker-compose.yml` on azure, instead of just executing the `.jar` file created from the build job of the pipeline (which runs an in memory database).

Complete Pipeline Configuration

```
name: CI/CD Pipeline
on:
  push:
    branches:
      - main

jobs:
  # Test the application
  tests:
    name: Unit tests
    runs-on: ubuntu-latest
    steps:
      # Check out repo
      - uses: actions/checkout@v2
      # Set up JDK
      - name: Set up JDK
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'
          cache: 'maven'
      # Run tests
      - name: Run Tests
        run: mvn -B test
        working-directory: ./portfolio-api

  # Sona's job
  sonar:
    # Make sure test job is succesfull
    needs: tests
    name: SonarCloud analysis
    runs-on: ubuntu-latest
    # Step's required for the job
    steps:
      # Check-out repository
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0 # Shallow clones should be disabled for a better
relevancy of analysis
      # Set up JDK
      - name: Set up JDK
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'
          cache: 'maven'
      # Sets up SonarCloud cache
      - name: Cache SonarCloud packages
        uses: actions/cache@v1
        with:
          path: ~/.sonar/cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
      # Uses SonarCloud to analyze the project
      - name: Build and analyze
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }} # Needed to get PR
information, if any
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        run: mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar
-Dsonar.projectKey=jKm00_cloud-service-portfolio -Dspring.profiles.active=test
        working-directory: ./portfolio-api

  # Build the application for production
  build:
    # Make sure sonar job is succesfull
    needs: sonar
    name: Build
    runs-on: ubuntu-latest
    steps:
      #Check-out repository
      - uses: actions/checkout@v2
      # Set up JDK
      - name: Set up JDK
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'
          cache: 'maven'
      #Build the application using Maven
      - name: Build with Maven
        run: mvn -B package -DskipTests --file pom.xml
        working-directory: ./portfolio-api
      # Upload build version of application
      - name: Upload JAR
        #This uploads artifacts from your workflow allowing you to share data
between jobs and store data once a workflow is complete.
        uses: actions/upload-artifact@v2
        with:
          name: portfolio-api
          #From this path
          path: portfolio-api/target/portfolio-api-1.0.jar

  # Deploy application to azure
  deploy:
    needs: build
    name: Deploy
    runs-on: ubuntu-latest
    environment:
      name: 'Production'
      url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}

    steps:
      - name: Download artifact from build job
        uses: actions/download-artifact@v2
        with:
          name: portfolio-api

      - name: Deploy to Azure Web App
        id: deploy-to-webapp
        uses: azure/webapps-deploy@v2
        with:
          app-name: 'jkm-spring-app'
          slot-name: 'Production'
          publish-profile: ${{
secrets.AZUREAPPSERVICE_PUBLISHPROFILE_F582479125054960BA7F5A09E0E7EA15 }}
          package: '*.jar'
```

Config file


build.yml
on: push
Unit tests 21s
SonarCloud analysis 1m 4s
Build 23s
deploy 37s
https://jkm-spring-app.azurewebsites.n...

Complete Pipeline Summary

Sources & Resources

- Pipeline configuration guide
- GitHub Actions: Java with maven guide
- Sonar Cloud Github Actions
- Azure / GitHub Pipeline
- GitHub Repository