GetawayGo

DevOps

Fontys University of Applied Sciences

GetawayGo

Anna Kadurina
18/10/2024

Version	Date	Author(s)	Changes	State
1.0	05/10/2024	Anna Kadurina	Creation of the document	Initial version
2.0	02/11/2024	Anna Kadurina	Update of the document	In progress
2.1	03/11/2024	Anna Kadurina	Update of the document	In progress
2.2	08/12/2024	Anna Kadurina	Update of the document	In progress
3.0	18/01/2025	Anna Kadurina	Update of the document	Final version

Table of Contents

Introduction

DevOps

DevOps is a set of practices that integrates software development (Dev) and IT operations (Ops) with the goal of improving collaboration, automating processes, and delivering higher-

quality software faster. In GetawayGo, DevOps plays a crucial role in ensuring efficient, scalable and secure deployment of microservices.

DevOps emphasizes continuous integration and continuous delivery (CI/CD), enabling frequent updates, rapid feedback loops, and enhanced collaboration with stakeholders.

Key elements of DevOps include:

- CI/CD pipelines: Continuous Integration (CI) and Continuous Delivery (CD) automate building, testing, and deploying code changes.
- Infrastructure as Code (IaC): Using tools like Azure Resource Manager (ARM) to define cloud infrastructure in a code-like manner.
- Monitoring and Feedback Loops: Continuous monitoring of system performance, security and user feedback. Tools such as Azure Monitor and Application Insights can be used.
- Collaboration Tools: Tools like Azure DevOps facilitate sprint management, version control, and tracking progress of PBIs (Product Backlog Items).

DevSecOps

DevSecOps is the evolution of DevOps to include security as an integral part of the development and operations processes. Instead of security being a separate part that occurs at the end of the development cycle, it is integrated from the beginning. This helps reduce vulnerabilities and ensure compliance with industry standards such as GDPR.

Key DevSecOps practices include:

- Security by Design: Implementing security controls from the beginning of the development process, ensuring that vulnerabilities are minimized.
- Automated Security testing: Tools such as OWASP ZAP or Snyk can be integrated into the CI/CD pipeline of GetawayGo to perform security checks at every build.
- Continuous Monitoring and Incident Response: Real-time monitoring of application behaviour.

How to adopt DevSecOps

Adopting DevSecOps in GetawayGo involves several steps:

- Integrating Security Tools in CI/CD pipelines: Use security testing tools like OWASP ZAP or Snyk to scan code for vulnerabilities during the CI.
- SonarQube in pipelines for Code Quality.
- Automated Testing and Validation: Implement automated unit, integration, and security testing into the pipelines.
- Regular Vulnerability Assessments: Perform tests to identify vulnerabilities that may not be detected in automated testing.

GetawayGo DevSecOps

Code Repositories

For the GetawayGo project, I will utilize the Polyrepo model of having multiple repositories. I have created the GetawayGo project in Azure DevOps where I am managing the repositories, as well as the backlog and boards, and pipelines.

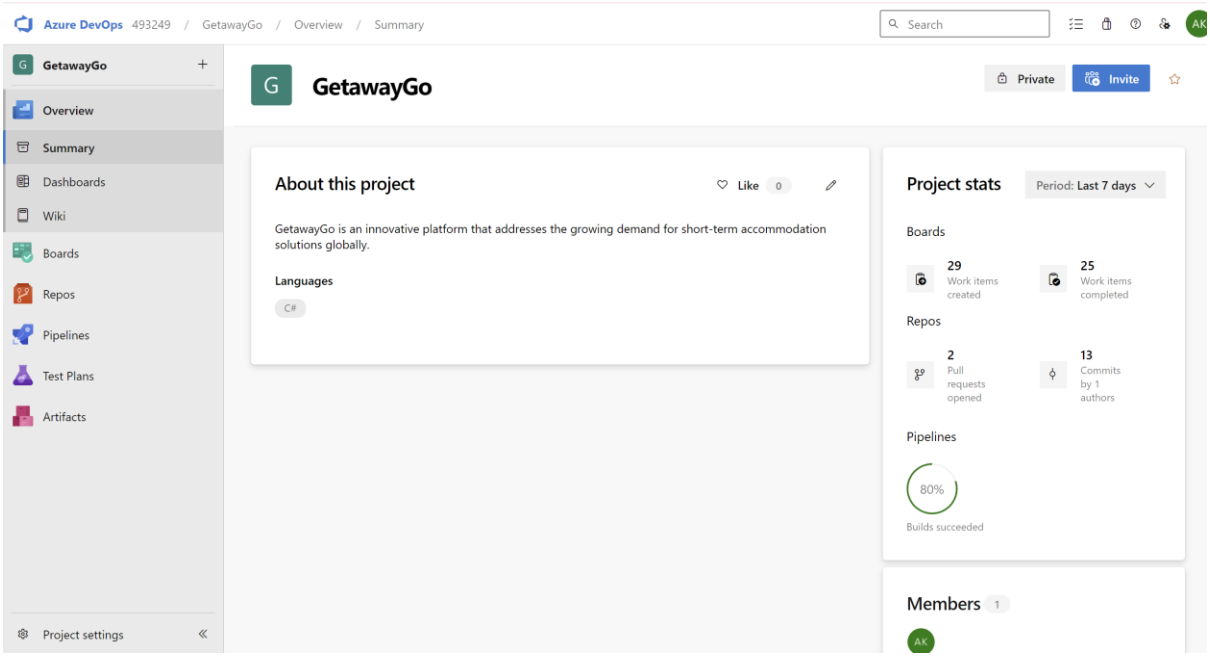


Figure 1 – GetawayGo project

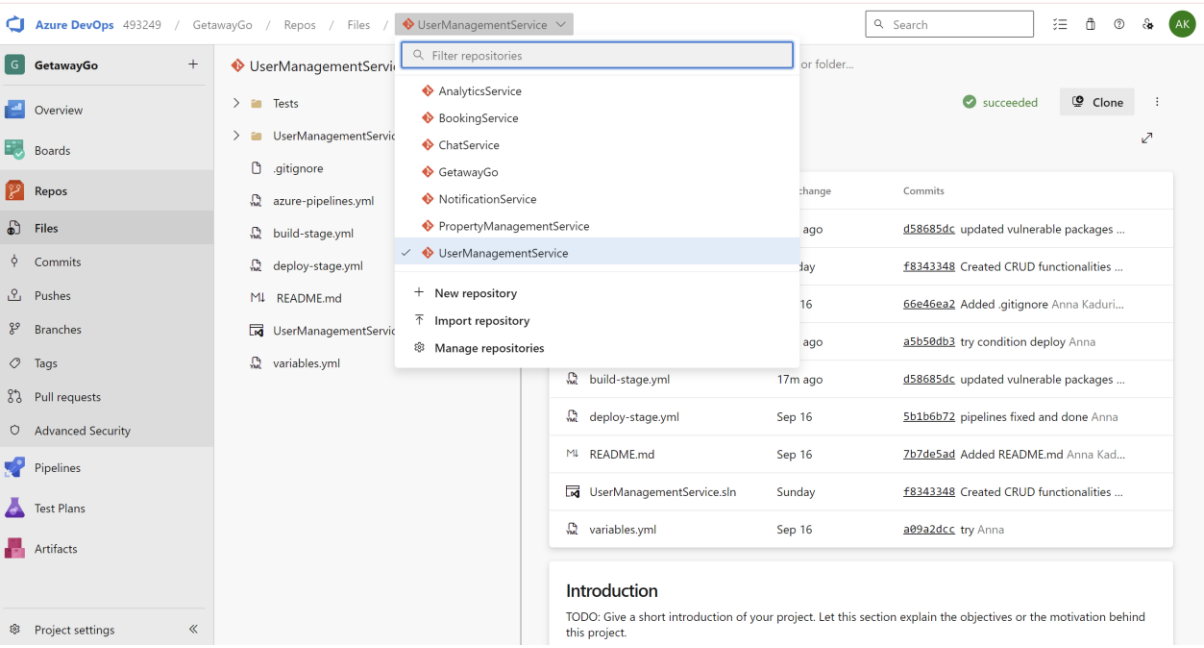


Figure 2 - Repositories

Pipelines

For each repository, I am utilizing a CI/CD pipeline. I have decided on dividing the pipeline logically.

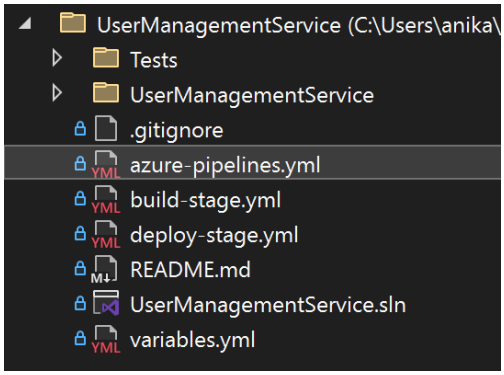


Figure 3 – Pipelines in UserManagementService

As you can see, I have decided on having a yaml file for Build, Deploy and one combined that runs the actual pipeline in Azure DevOps. I am also storing the relevant variables in a separate file.

The Build pipeline builds and tests the code. Also, I have added Snyk Scan to test for vulnerabilities and ensure that the application is secure to be deployed. Furthermore, I incorporated OWASP ZAP Scan to perform further risk assessment of the application. If any of these steps fail, the pipeline fails as well.

azure-pipelines.yml | build-stage.yml | Tests.csproj | UserRepository.cs | UserMapper.cs | Startup.cs | UserController.cs | appsettings.Development.json

steps

```yaml
parameters:
  - name: solution
    type: string
    default: '**/*.sln'
  - name: buildPlatform
    type: string
    default: 'Any CPU'
  - name: buildConfiguration
    type: string
    default: 'Release'
  - name: targetUrl
    type: string
    default: 'https://userservicegetawaygo.azurewebsites.net'

steps:
  - task: NuGetToolInstaller@1

  - task: NuGetCommand@2
    inputs:
      restoreSolution: ${{ parameters.solution }}

  - task: VSBuild@1
    inputs:
      solution: ${{ parameters.solution }}
      msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:PackageLocation="$(build.artifactStagingDirectory)"'
      platform: ${{ parameters.buildPlatform }}
      configuration: ${{ parameters.buildConfiguration }}

  - task: VSTest@2
    inputs:
      platform: ${{ parameters.buildPlatform }}
      configuration: ${{ parameters.buildConfiguration }}

  - task: SnykSecurityScan@1
    inputs:
      serviceConnectionEndpoint: 'SnykConnection'
      testType: 'app'
      targetFile: UserManagementService.sln
      monitorWhen: 'always'
      failOnIssues: true
      jsonFileOutput: '$(Build.ArtifactStagingDirectory)/snyk-report.json'

  - script: |
      docker run --rm -v $(Build.ArtifactStagingDirectory):/zap/wrk/:rw -t zaproxy/zap-stable zap.sh -cmd -quickurl ${{ parameters.targetUrl }} -quickout /zap/wrk/zap_report.html
    displayName: 'Run OWASP ZAP Scan'

  - task: PublishPipelineArtifact@1
    inputs:
      artifactName: 'OWASPZAPReports'
      targetPath: '$(Build.ArtifactStagingDirectory)'

  - task: PublishBuildArtifacts@1
    inputs:
      PathtoPublish: '$(build.artifactStagingDirectory)'
      ArtifactName: 'UserServiceGetawayGo'
```

Figure 4 – Build pipeline first way

variables.yml | deploy-stage.yml | azure-pipelines.yml | **build-stage.yml** | BookingController.cs

inputs

```yaml
parameters:
  - name: solution
    type: string
    default: '**/*.sln'
  - name: buildPlatform
    type: string
    default: 'Any CPU'
  - name: buildConfiguration
    type: string
    default: 'Release'
  - name: targetUrl
    type: string
    default: 'https://bookingservicegetawaygo.azurewebsites.net'

steps:
  - task: NuGetToolInstaller@1

  - task: NuGetCommand@2
    inputs:
      restoreSolution: ${{ parameters.solution }}

  - task: DotNetCoreCLI@2
    displayName: Build
    inputs:
      command: build
      projects: '**/*.csproj'
      arguments: '--configuration $(buildConfiguration)'

  - task: DotNetCoreCLI@2
    inputs:
      command: test
      projects: '**/*Tests/*.csproj'
      arguments: '--configuration $(buildConfiguration)'

  - task: SnykSecurityScan@1
    inputs:
      serviceConnectionEndpoint: 'SnykConnection'
      testType: 'app'
      targetFile: BookingService.sln
      monitorWhen: 'always'
      failOnIssues: true
      jsonFileOutput: '$(Build.ArtifactStagingDirectory)/snyk-report.json'

  - script: |
      docker run --rm -v $(Build.ArtifactStagingDirectory):/zap/wrk/:rw -t zaproxy/zap-stable zap.sh -cmd -quickurl ${{ parameters.targetUrl }} -quickout /zap/wrk/zap_report.ht
    displayName: 'Run OWASP ZAP Scan'

  - task: PublishPipelineArtifact@1
    inputs:
      artifactName: 'OWASPZAPReports'
      targetPath: '$(Build.ArtifactStagingDirectory)'

  - task: DotNetCoreCLI@2
    inputs:
      command: publish
      publishWebProjects: True
      arguments: '--configuration $(BuildConfiguration) --output $(Build.ArtifactStagingDirectory)'
      zipAfterPublish: True

  - task: PublishBuildArtifacts@1
    inputs:
      PathtoPublish: '$(build.artifactStagingDirectory)'
      ArtifactName: 'BookingServiceGetawayGo'
```

Figure 5 – Build pipeline second way

Also, I added SonarQube Cloud to check the Code Quality and provide me with insights on Build.

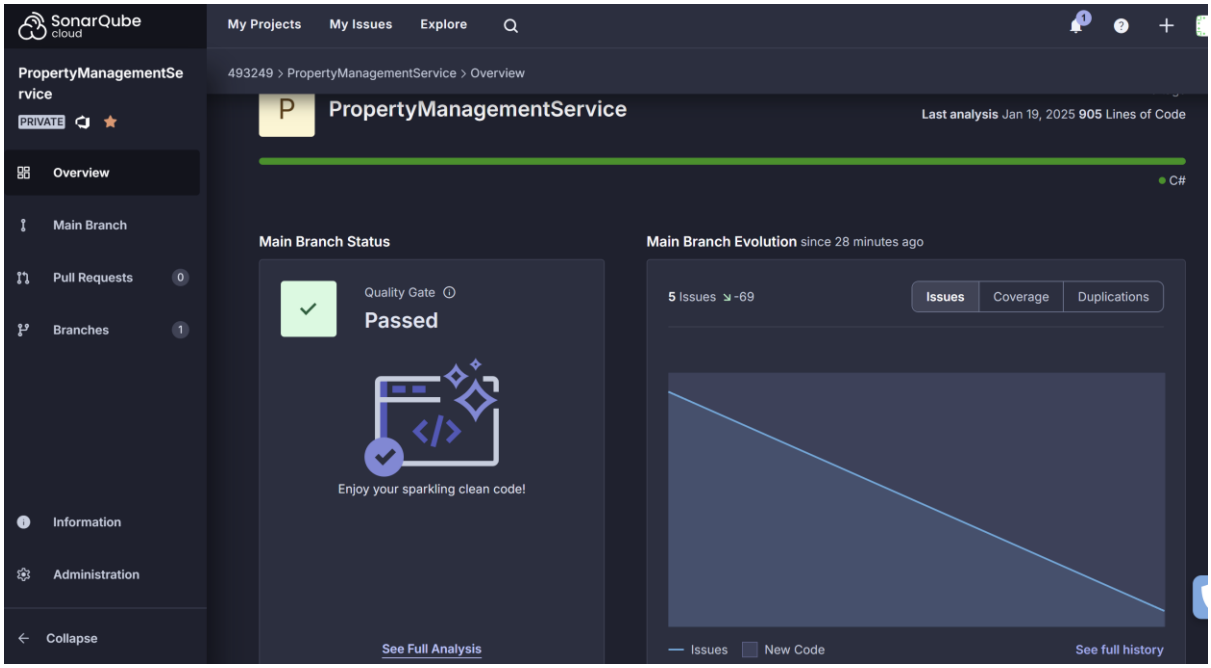
```yaml
- task: SonarQubePrepare@5
  inputs:
    SonarQube: 'SonarQubeCloudProperties'
    scannerMode: 'MSBuild'
    projectKey: '493249_PropertyManagementService'
    projectName: 'PropertyManagementService'
    projectVersion: '1.0'
    extraProperties: |
      sonar.organization=493249
      sonar.sources=.
      sonar.language=csharp

- task: DotNetCoreCLI@2
  displayName: Build
  inputs:
    command: build
    projects: '**/*.csproj'
    arguments: '--configuration $(buildConfiguration)'

- task: SonarQubeAnalyze@5

- task: SonarQubePublish@5
  inputs:
    pollingTimeoutSec: '300'
```

When the report is ready, I can see it on SonarQube Cloud in my organization.



When the Snyk task is finished, I get the results from it and can evaluate if there is anything wrong. If there are vulnerabilities, the pipeline fails and I can check what the problem is and can fix it.

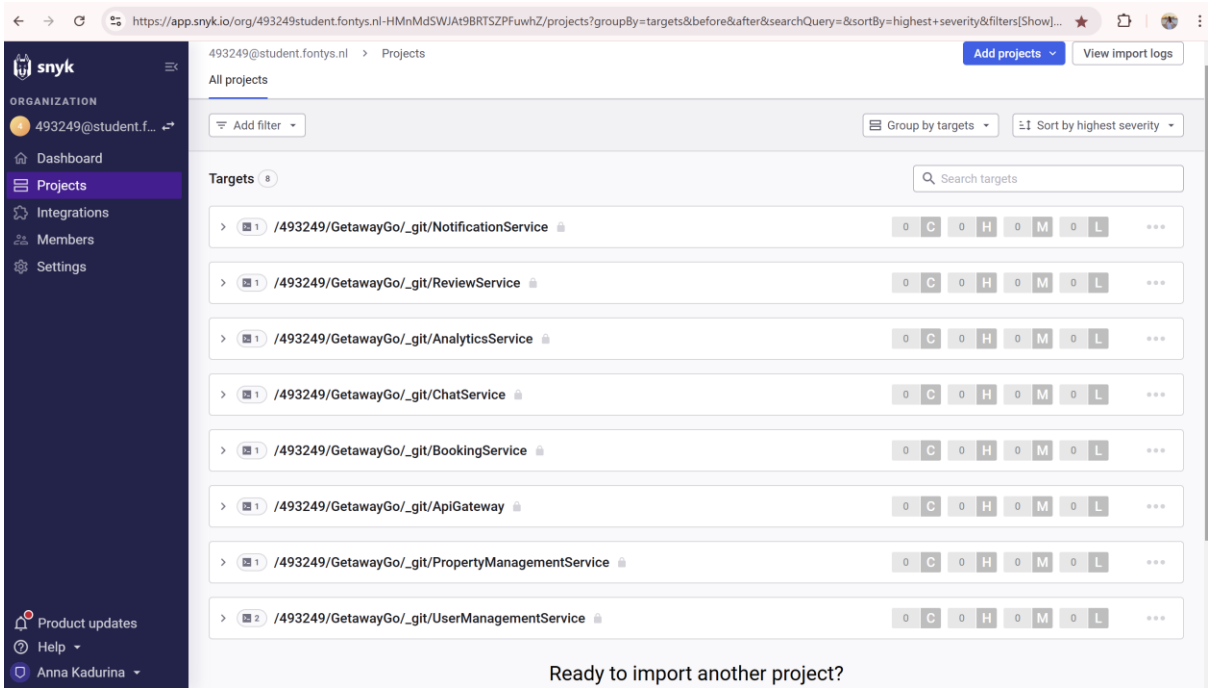


Figure 6 – Snyk Projects

Afterwards, the OWASP ZAP Scan is executed. When it is done, I get a report that I publish as an artifact and can access it via Azure DevOps.

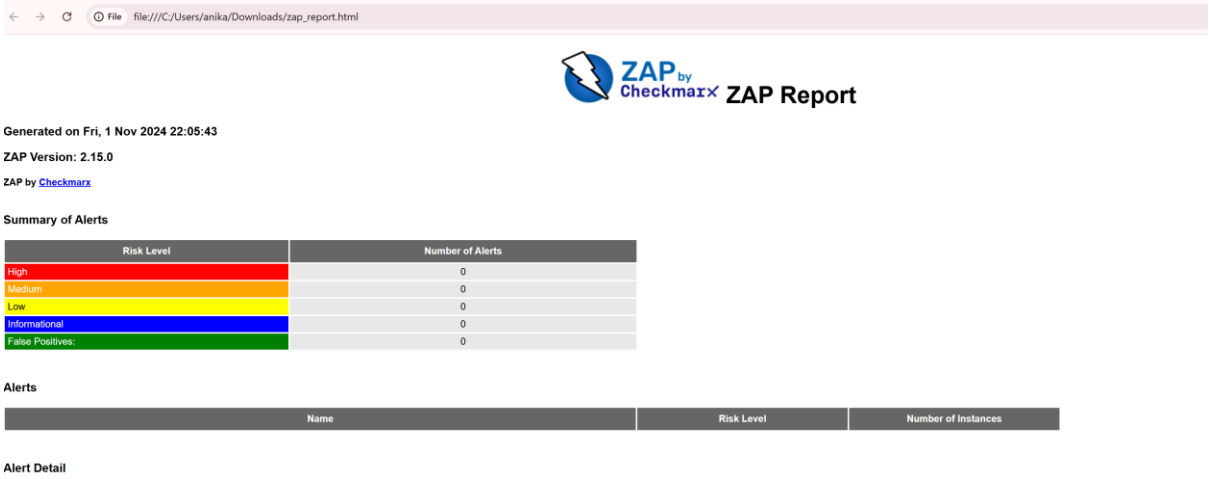


Figure 7 – OWASP ZAP Report

If the Build pipeline succeeds and I am on the main branch, I can start the Deploy pipeline. This pipeline deploys the code to the relevant App Service in Azure. I have created an Environment Production that requires an approval before deployment.

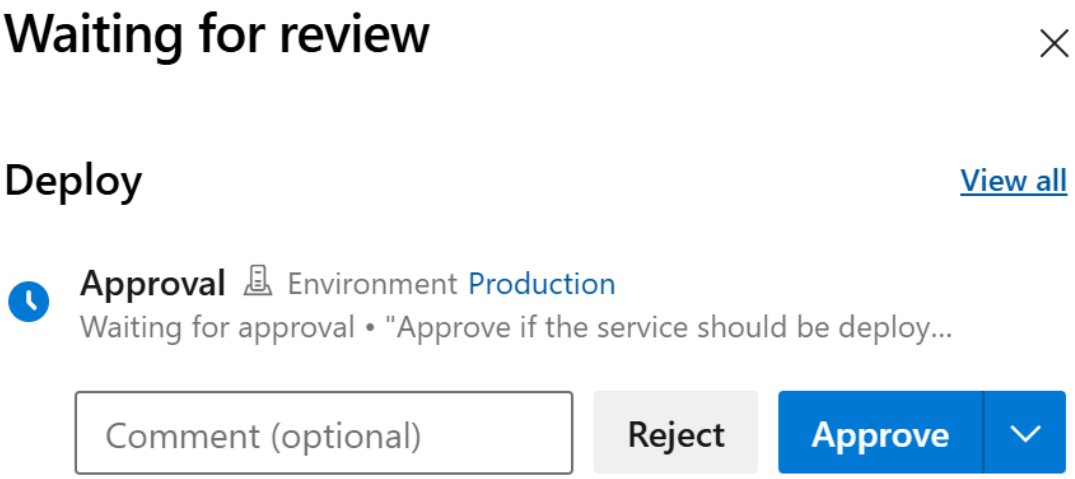


Figure 8 – Pre-deployment approval for Production

```yaml
steps:
- task: DownloadBuildArtifacts@0
  inputs:
    artifactName: 'PropertyServiceGetawayGo'
    downloadPath: '$(System.DefaultWorkingDirectory)'

- task: AzureRmWebAppDeployment@4
  inputs:
    azureSubscription: $(azureSubscription)
    appType: 'webApp'
    WebAppName: $(appServiceName)
    deployToSlotOrASE: false
    ResourceGroupName: $(resourceGroup)

```

Figure 9 – Deploy pipeline – first way

```yaml
steps:
- task: DownloadBuildArtifacts@0
  inputs:
    artifactName: 'APIGateway'
    downloadPath: '$(System.DefaultWorkingDirectory)'

- script: |
    echo "Listing files in $(System.DefaultWorkingDirectory)"
    ls $(System.DefaultWorkingDirectory)
  displayName: 'List Published Files'

- task: AzureCLI@2
  inputs:
    azureSubscription: $(azureSubscription)
    scriptType: bash
    scriptLocation: inlineScript
    inlineScript: |
      az webapp deployment source config-zip \
        --resource-group $(resourceGroup) \
        --name $(appServiceName) \
        --src $(System.DefaultWorkingDirectory)/**/*.zip
```

Figure 10 – Deploy pipeline – second way

I have combined the Build and Deploy steps in the azure-pipelines.yml and this is the file that the pipeline runs on. The Deploy step depends on the Build and if it is successful. It also depends on the branch that is being currently used. If it is a feature branch, the Deploy will not run. The pipeline runs on a self-hosted agent.

azure-pipelines-1.yml | **azure-pipelines.yml** | build-stage.yml | Tests.csproj | UserRepository.cs

stages.jobs.strategy.runOnce.deploy.steps

```yaml
trigger:
- main

variables:
- template: variables.yml

stages:
- stage: Build
  jobs:
  - job: Build
    pool:
      name: Default
      demands:
        - agent.name -equals AgentAnna
    steps:
    - template: build-stage.yml

- stage: Deploy
  dependsOn: Build
  condition: and(succeeded(), eq(variables['Build.SourceBranchName'], 'main'))
  variables:
  - group: userservice-prod-vg
  jobs:
  - deployment: DeployJob
    displayName: Deploy App Service
    environment:
      name: Production
    strategy:
      runOnce:
        deploy:
          pool:
            name: Default
            demands:
              - agent.name -equals AgentAnna
          steps:
          - template: deploy-stage.yml
```

Figure 11 – Main pipeline

As you can see from the pipeline, I am utilizing a variable group where I can keep my connection strings and other secrets safe.

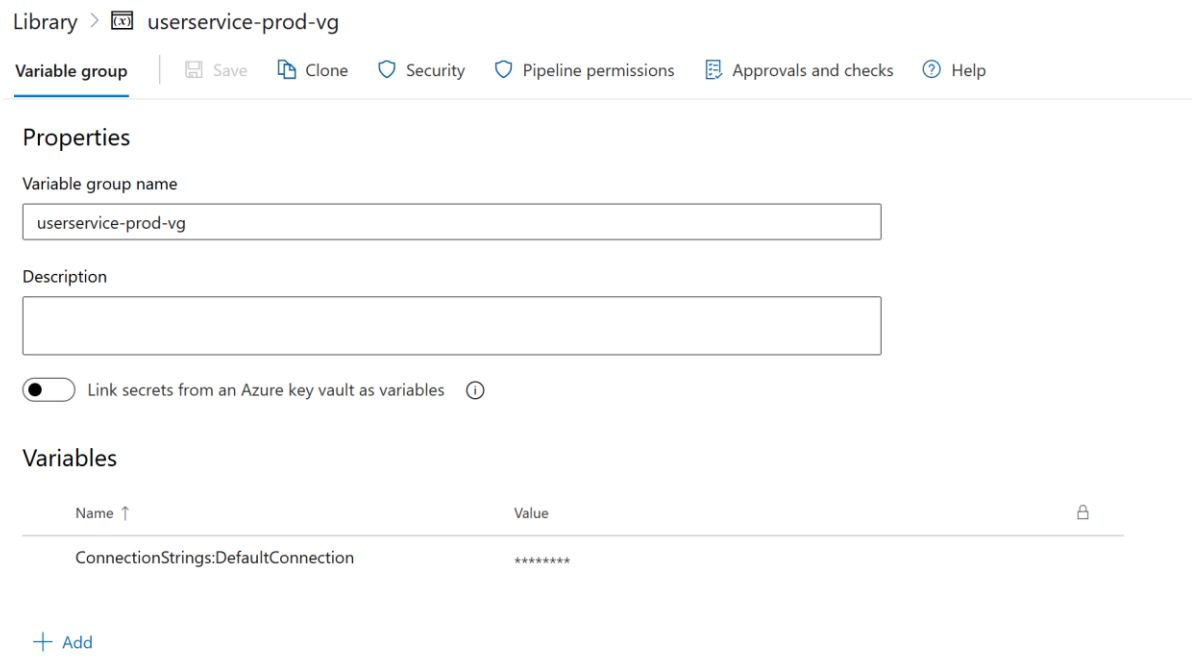


Figure 12 – Variable group

The frontend is also utilizing a CI/CD approach. It is being deployed in a static web app in Azure and I used the built-in functionality of Azure to automatically create all resources for the deployment.

```yaml
name: Azure Static Web Apps CI/CD

pr:
  branches:
    include:
      - main

trigger:
  branches:
    include:
      - main

resources:
  repositories:
    - repository: frontendRepo
      type: git
      name: GetawayGo/Frontend
      ref: refs/heads/main

jobs:
- job: build_and_deploy_job
  displayName: Build and Deploy Job
  condition: or(eq(variables['Build.Reason'], 'Manual'), or(eq(variables['Build.Reason'], 'PullRequest'), eq(variables['Build.Reason'], 'IndividualCI')))
  pool:
    vmImage: ubuntu-latest
  variables:
    - group: Azure-Static-Web-Apps-nice-mushroom-09ebf5f03-variable-group
  steps:
    - checkout: frontendRepo

    - script: |
        npm install
        npm run build
      displayName: 'Install and Build React App'

    - script: |
        npm install cypress --save-dev
        npx cypress verify # Verify Cypress installation
        npm run build # Ensure the app is built
        npx cypress run --headless --browser chrome
      displayName: 'Run Cypress Tests'
      continueOnError: false

    - task: AzureStaticWebApp@0
      inputs:
        azure_static_web_apps_api_token: $(AZURE_STATIC_WEB_APPS_API_TOKEN_NICE_MUSHROOM_09EBF5F03)
        app_location: "/"
        api_location: ""
        output_location: "build"
```

Figure 13 – Frontend pipeline

The frontend pipeline uses a combination of 2 repos – the one where the frontend is and the one Azure automatically created with the yml file. The frontend also utilizes the variables groups where a token for the deployment has been placed as a secret. The pipeline also executes the end-to-end tests in the project with Cypress. It is configured to stop execution if the tests fail.

And here is an overview of all the pipelines in the GetawayGo project.

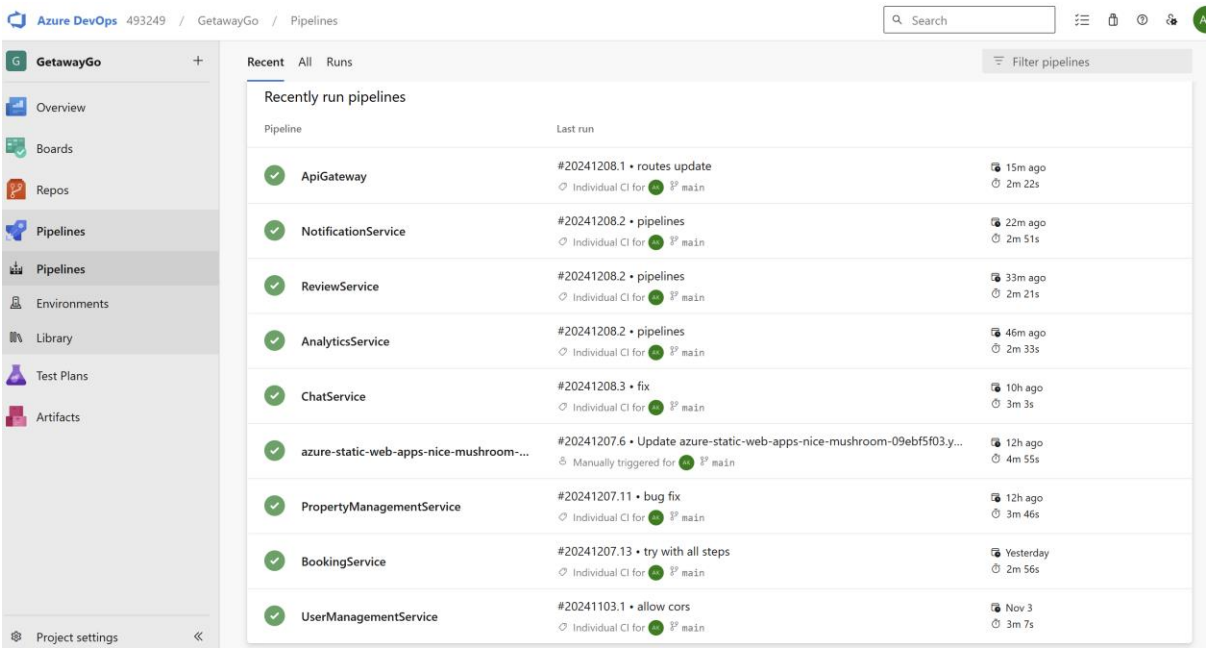


Figure 14 – All pipelines

Monitoring and Testing

Monitoring

At this stage, I have utilized Azure Application Insights for monitoring of the microservices. If there is a problem, and email is sent to me.

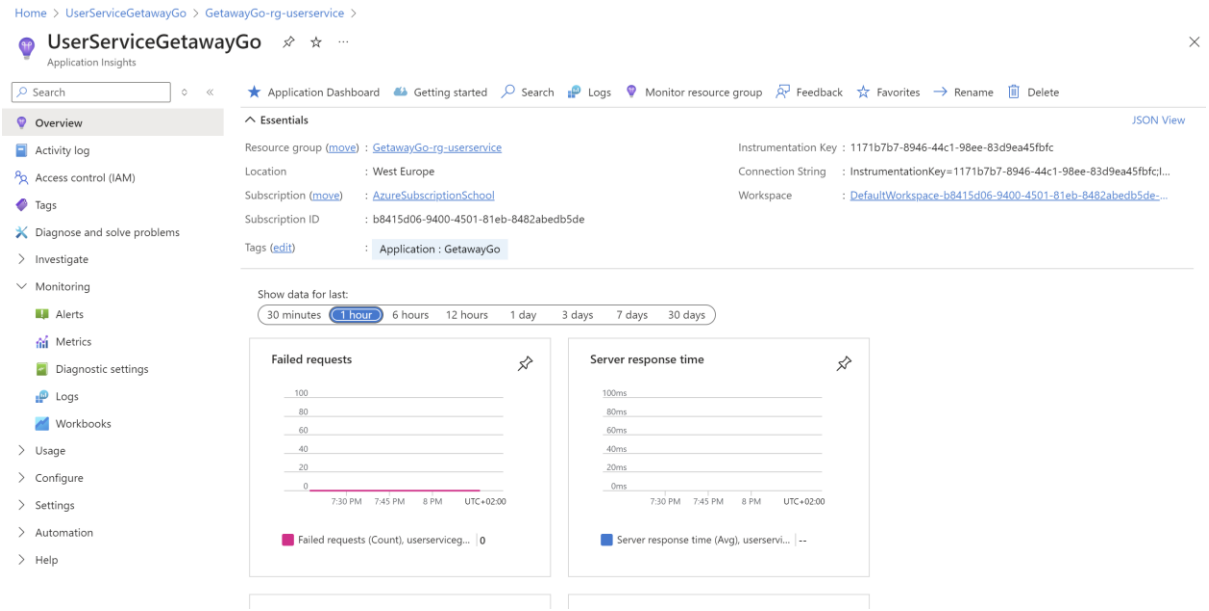


Figure 15 – Application Insights of UserManagementService App Service

Furthermore, I have connected my frontend to sentry.io to monitor errors and performance. If there is something wrong, and email is sent to me again.

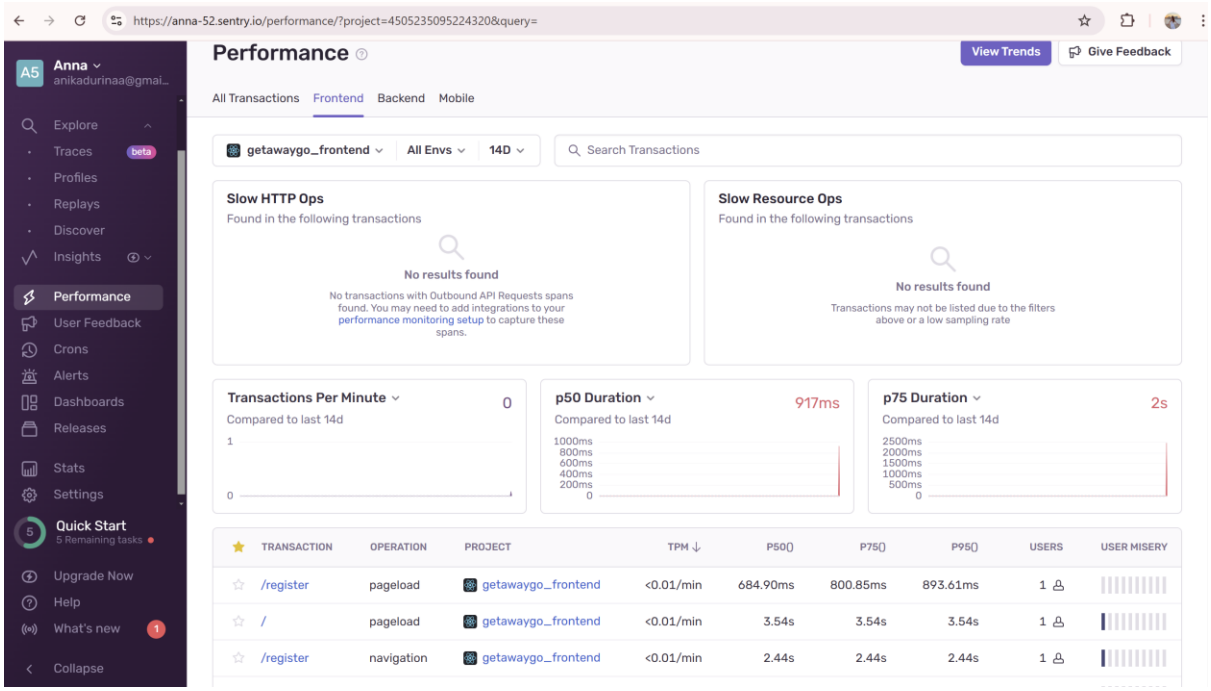


Figure 16 – Monitoring of frontend in sentry.io

Testing

Unit testing

For testing purposes, I have implemented Unit testing and I execute them in the pipelines. If they fail, the pipeline also fails. This way, I ensure that every time, the service is about to be deployed, it will be automatically tested. I am using xUnit and Moq to create fake object and test various scenarios.

Integration Testing

I have also implemented integration testing to ensure that the different services have successful communication. I am also testing the functionalities of external resources like Azure Service Bus, Stripe payment system, SendGrid, etc. These tests are also included in the pipeline. The unit and integration tests both run with the command ‘dotnet test’. If any test fails, the pipeline fails.

End-to-End Testing

I utilized Cypress to perform E2E testing in my application. I have integrated them into the pipeline for the frontend. If the tests fail, the pipeline stops and the web application is not deployed.

Snyk Security Testing

I have also integrated Snyk into my pipelines. This tool checks for vulnerabilities in the packages of the project. This is vital for the security of the platform.

OWASP ZAP Scan

I have also incorporated OWASP ZAP Scan to perform security testing and asses the risks in my application in the Build pipeline.

Azure Load Testing

I used Azure Load Testing to ensure that the application can handle various loads of requests. I used different types of testing such as linear and spike.

Lighthouse

Lastly, I generated reports using Lighthouse to assess the Performance, Accessibility, Best Practices, and SEO of the website.

Conclusion

GetawayGo utilizes DevSecOps as a way of working. I have already created CI/CD pipelines with automatic testing – unit, integration, end2end, snyk, and OWASP. Deployment is available only if build and all the tests pass, and is manually approved. Monitoring is done through Application Insights and Sentry.io. Further improvements will be made throughout the development of the project.