Sofia university “St. Kliment Ohridski”

Faculty of Mathematics and Informatics

**Subject: Modern DevOps Practices**

**Winter semester, 2024/2025**

**Project: GitHub Workflow for Online Games Platform**

**Course project**

**Created by:**

**Sasho Vasilchev, Software engineering, year 3, fn: 0MI0600197**

**Silvia Zemedelska, Software engineering, year 3, fn: 6MI0600237**

Table of Contents

1. **Project Introduction3**
2. **Workflow Overview3**
3. **Detailed Job Descriptions4**
   1. PushDockerImgFrontEnd4
   2. Build4
   3. Test5
   4. Security5
   5. BuildDocker-image6
   6. ScanDocker-imageWithTrivy7
   7. PushDocker-image7
   8. DeployToMiniKube8
4. **Conclusion8**

This documentation provides a detailed description of each part of the GitHub Workflow, designed to automate processes related to development, testing, security, containerization, and deploying applications in Kubernetes. We will explore each job, the tools used, and how they align with DevOps and SDLC project requirements.

1. **Project Introduction**

The "OnlineGames" project is an interactive online gaming platform featuring two classic games: Tic-Tac-Toe and Connect4. Built using ASP.NET Core for the backend and Angular for the frontend, it supports both multiplayer and single-player modes against the computer. The platform includes features like user profiles, chat, notifications, and a friend management system. The games use AI algorithms like Minimax, with optimizations such as alpha-beta pruning, memoization, and bitmasking to enhance the gaming experience. The project is designed for real-time play with WebSocket communication.

1. **Workflow Overview**

* **Workflow Name:** main
* **Triggers:** The workflow is triggered by the following events:
  1. **Push to the main branch:** Activates the workflow whenever new code is pushed to the main branch.
  2. **Pull Request to the main branch:** Activates the workflow for every pull request targeting the main branch.
  3. **Manual Trigger:** The workflow can be started manually using workflow\_dispatch.
* **Workflow Phases:** The workflow is organized into several key jobs, each performing specific tasks:
  1. PushDockerImgFrontEnd
  2. Build
  3. Test
  4. Security
  5. BuildDocker-image
  6. ScanDocker-imageWithTrivy
  7. PushDocker-image
  8. DeployToMiniKube

1. **Detailed Job Descriptions**
   1. **PushDockerImgFrontEnd**

* **Purpose:** Build and publish a Docker image for the frontend application.
* **Operations:**
  1. **Repository Checkout:**
     + Uses actions/checkout@v4 to download source code from the repository.
  2. **Setup Docker Buildx:**
     + Uses docker/setup-buildx-action@v2 for multi-platform Docker image building.
  3. **Login to GitHub Container Registry (GHCR):**
     + Uses docker/login-action@v2 for authentication with GITHUB\_TOKEN.
  4. **Build and Push Docker Image:**
     + Executes docker/build-push-action@v4:
       - Context: ./OnlineGames.Client
       - Dockerfile: ./OnlineGames.Client/Dockerfile
       - Tags: ghcr.io/sashovas/online\_games\_frontend:latest
       - Push: Automatically uploads the image to GHCR.
* **Result:** The frontend Docker image is successfully published to GHCR.
  1. **Build**
* **Purpose:** Build the backend application using .NET.
* **Operations:**
  1. **Repository Checkout:**
     + Uses actions/checkout@v4.
  2. **Setup .NET SDK:**
     + Uses actions/setup-dotnet@v4 with version 6.0.x.
  3. **Dependency Restore:**
     + Runs dotnet restore in ./OnlineGames.Server.
  4. **Project Build:**
     + Runs dotnet build --no-restore for efficiency.
  5. **Upload Artifacts:**
     + Uses actions/upload-artifact@v4 to store build results.
* **Result:** Backend application is built, and artifacts are stored.
  1. **Test**
* **Purpose:** Run tests and verify code quality for the backend application.
* **Dependencies:** Requires the Build job to complete successfully.
* **Operations:**
  1. **Download Artifacts:**
     + Uses actions/download-artifact@v4.
  2. **Setup .NET SDK:**
     + Same configuration as in the Build job.
  3. **Check Code Formatting:**
     + Runs dotnet format --verify-no-changes.
  4. **Run Tests:**
     + Executes dotnet test --no-build --verbosity normal.
* **Result:** Tests are executed, and results are documented.
  1. **Security**
* **Purpose:** Identify potential vulnerabilities in the code using Snyk.
* **Dependencies:** Requires the Build job to complete successfully.
* **Operations:**
  1. **Install Snyk CLI:**
     + Runs npm install -g snyk.
  2. **Authentication:**
     + Uses the SNYK\_TOKEN secret for login.
  3. **Vulnerability Scan:**
     + Runs snyk test --all-projects --severity-threshold=high.
  4. **Monitoring:**
     + Runs snyk monitor --all-projects --severity-threshold=high.
* **Result:** Potential vulnerabilities are identified and monitored via Snyk.
  1. **BuildDocker-image**
* **Purpose:** Build a Docker image for the backend application and save it as an artifact.
* **Dependencies:** Requires Test and Security jobs to complete successfully.
* **Operations:**
  1. **Download Artifacts:**
     + Uses actions/download-artifact@v4.
  2. **Setup Docker Buildx:**
     + Uses docker/setup-buildx-action@v2.
  3. **Login to GHCR:**
     + Uses docker/login-action@v2.
  4. **Build Docker Image:**
     + Executes docker/build-push-action@v4:
       - Context: ./OnlineGames.Server
       - Dockerfile: ./OnlineGames.Server/Dockerfile
       - Tags: ghcr.io/sashovas/online\_games\_api:latest
       - The image is not directly pushed.
  5. **Export as Tar File:**
     + Saves the image using docker save.
  6. **Upload Tar File as Artifact:**
     + Uses actions/upload-artifact@v4.
* **Result:** Docker image is saved as a tar file and uploaded.
  1. **ScanDocker-imageWithTrivy**
* **Purpose:** Scan the Docker image for vulnerabilities using Trivy.
* **Dependencies:** Requires BuildDocker-image to complete successfully.
* **Operations:**
  1. **Download Tar File:**
     + Uses actions/download-artifact@v4.
  2. **Load Docker Image:**
     + Runs docker load -i image\_backend.tar.
  3. **Install Trivy:**
     + Adds Trivy's repository and installs it using apt.
  4. **Scan for Vulnerabilities:**
     + Runs trivy image --severity HIGH,CRITICAL.
* **Result:** Vulnerability report for the Docker image.
  1. **PushDocker-image**
* **Purpose:** Publish the Docker image to GHCR.
* **Dependencies:** Requires ScanDocker-imageWithTrivy to complete successfully.
* **Operations:**
  1. **Download Tar File:**
     + Uses actions/download-artifact@v4.
  2. **Login to GHCR:**
     + Uses docker/login-action@v2.
  3. **Load Docker Image:**
     + Runs docker load -i image\_backend.tar.
  4. **Push Docker Image:**
     + Runs docker push ghcr.io/sashovas/online\_games\_api:latest.
  5. **DeployToMiniKube**
* **Purpose:** Deploy the frontend and backend applications to a Kubernetes cluster using Minikube.
* **Dependencies:** Requires PushDocker-image and PushDockerImgFrontEnd to complete successfully.
* **Operations:**
  1. **Repository Checkout:**
     + Uses actions/checkout@v4.
  2. **Start Minikube:**
     + Uses medyagh/setup-minikube@latest.
  3. **Verify Cluster:**
     + Runs kubectl get pods -A.
  4. **Deploy Applications:**
     + Uses kubectl apply -f deployment.yaml.
  5. **Wait for Readiness:**
     + Runs kubectl wait --for=condition=ready pod -l app=frontend --timeout=120s.
* **Result:** Applications are successfully deployed and ready for testing in a local Minikube cluster.

1. **Conclusion**

This GitHub Workflow is a robust and comprehensive solution for automating key stages of application development, including building, testing, securing, containerizing, and deploying .NET-based applications. By leveraging GitHub Actions and integrating tools such as Docker, Snyk, Trivy, and Minikube, the workflow ensures streamlined and efficient DevOps practices aligned with SDLC requirements.

The process emphasizes automation, quality assurance, and security at every stage, from building and testing to scanning for vulnerabilities and deploying to a Kubernetes cluster. This structured approach not only enhances the reliability and security of the applications but also reduces manual effort and the risk of errors.

Through this workflow, development teams can achieve faster delivery cycles, better code quality, and a seamless transition from local development to production-like environments. It serves as an excellent foundation for teams looking to implement scalable and efficient CI/CD pipelines.