CI/CD Pipeline for MATLAB Algorithm Deployment

Tomasz Olejarczuk

Fontys

23-04-2024

Contents

CI/CD Pipeline for MATLAB Algorithm Deployment **2**

1. Summary **2**

2. Introduction **2**

3. Workflow Overview **2**

4. Pipeline Breakdown (Detailed) **4**

4.1 Actual code: 6

5. Deployment Considerations (Future) **7**

6. Conclusions **8**

CI/CD Pipeline for MATLAB Algorithm Deployment

1. Summary

This document outlines the CI/CD (Continuous Integration/Continuous Delivery) pipeline implemented for the MATLAB algorithm project. Utilizing GitHub Actions, the pipeline automates the process of building, versioning and deploying Docker images containing the MATLAB algorithm to the GitHub Container Registry (GHCR). The pipeline is triggered whenever changes are pushed to the 'main' branch, ensuring that the latest version of the algorithm is readily available for deployment, streamlining the development and release process.

2. Introduction

Purpose: The CI/CD pipeline aims to streamline the development and deployment workflow for the MATLAB algorithm project. Key objectives include:

- **Automation:** Automate the repetitive tasks of building, testing (if applicable), versioning, and publishing Docker images, reducing manual intervention and potential errors.
- **Consistency:** Enforce a consistent pipeline process, ensuring builds and deployments follow predefined steps, promoting reliability.
- **Agility:** Enable frequent updates to the MATLAB algorithm with the confidence that automated processes will build and deploy changes seamlessly.

Scope: The pipeline focuses on the following key components:

- **MATLAB Code Repository:** The source code for the MATLAB algorithm, hosted on GitHub.
- **Docker Images:** The containerized format used to package and deploy the MATLAB algorithm.
- **GitHub Container Registry (GHCR):** The repository for storing the built Docker images.

3. Workflow Overview

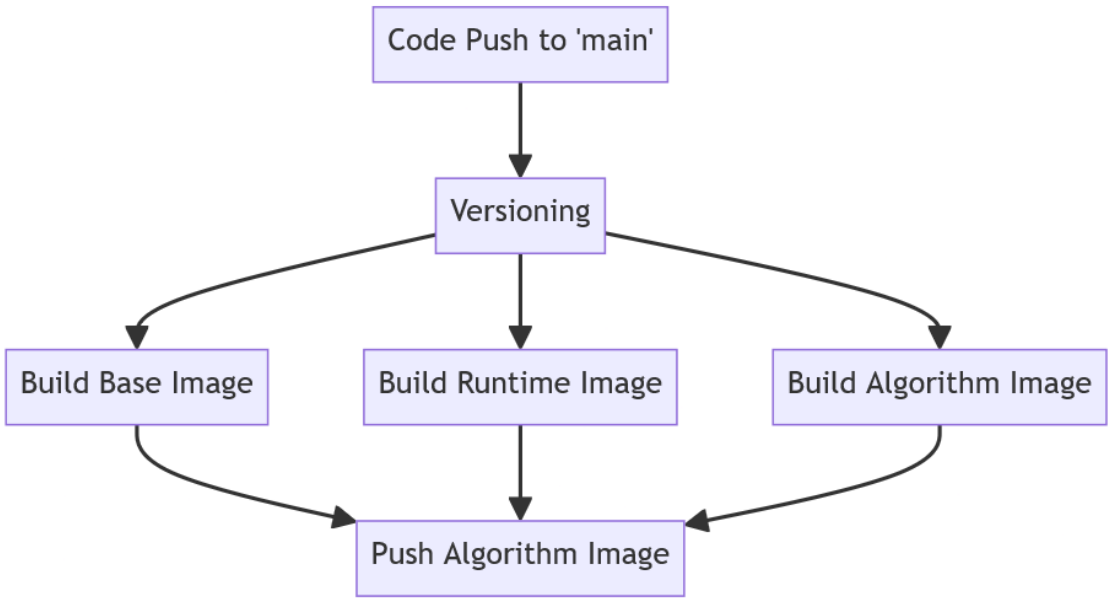
Trigger:

- The CI/CD pipeline is initiated whenever new code is pushed to the 'main' branch of the GitHub repository.

High-Level Steps:

1. **Semantic Versioning:**
    - The pipeline determines the appropriate version increment (major, minor, or patch) based on the nature of the committed changes.
    - A new version tag is generated and applied to the Docker image.
2. **Docker Image Build:**
    - **Base Image (if needed):** The pipeline checks if a base image with the required MATLAB runtime dependencies exists. If not, a new base image is built.
    - **Runtime Image (if needed):** The pipeline checks if a runtime image exists; if not, it's created. This image potentially layers on top of the base image.
    - **Algorithm Image:** The MATLAB algorithm code is packaged into a Docker image, leveraging the runtime (and potentially the base) image as its foundation. This final artifact is given the new version tag and a "latest" tag.
3. **Image Publishing:**
    - The newly built and tagged Docker image containing the MATLAB algorithm is pushed to the GitHub Container Registry (GHCR).

```
graph TD
    A[Code Push to 'main'] --> B[Versioning]
    B --> C[Build Base Image]
    B --> D[Build Runtime Image]
    B --> E[Build Algorithm Image]
    C --> F[Push Base Image]
    D --> F[Push Runtime Image]
    E --> F[Push Algorithm Image]
```



4. Pipeline Breakdown (Detailed)

Job: push-matlab-algorithm-image

- **runs-on:** ubuntu-latest
  - The pipeline executes within an Ubuntu-based environment.

Steps

1. **Checkout** (`actions/checkout@v4`)
   - Retrieves a copy of your project's code from the GitHub repository. This is essential for subsequent build steps.
2. **Semantic Versioning** (`anothrNick/github-tag-action@1.69.0`)
   - **Key Tasks:**

  - Examines the nature of changes in the code push to determine the appropriate version increment (e.g., major.minor.patch).
  - Generates a new unique version tag to be applied to the Docker image.
- **Note:** Explain how `anothrNick/github-tag-action` integrates with your code analysis or the commit message conventions you might use to drive versioning logic.

3. **GitHub Container Registry (GHCR) Login** (`docker/login-action@v3`)
   - Authenticates with GHCR using your GitHub credentials. This grants permission to push images in subsequent steps.

4. **Build Runtime Base Image**
   - **Conditional Logic:** If the `ghcr.io/teamiumdev/matlabruntimebase:r2024a` image doesn't already exist, it is built.
   - **Dockerfile:** This build is directed by `Dockerfile.deps`.
   - **Purpose:** This image installs MATLAB Runtime dependencies, providing a common foundation for later builds.

5. **Build Runtime Image**
   - **Conditional Logic:** Similarly, the `ghcr.io/teamiumdev/matlabruntime:r2024a` image is only built if it doesn't exist.
   - **Dockerfile:** This step uses `Dockerfile.runtime`.
   - **Purpose :** This layer might add additional dependencies or configuration before packaging the algorithm itself.

6. **Build Inventory Image**
   - **Dockerfile:** It uses a dedicated Dockerfile (e.g., `Dockerfile`).
   - **Image Building:** Bundles the MATLAB algorithm code, along with necessary runtime components, creating the final deployable image.
   - **Tagging:**
     - **Version Tag:** The new tag generated in the versioning step is applied.
     - **'latest' Tag:** This tag offers a convenient reference for deployment.

7. **Push Inventory Image**
   - **Destination:** Pushes the newly built image to GHCR, along with its tags, making it available for deployment.

4.1 Actual code:

```yaml
name: Deploy Images to GHCR and Auto Tag
on:
  push:
    branches:
      - main
jobs:
  push-matlab-algorithm-image:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
      with:
        fetch-depth: '0'
    - name: Minor version for each merge
      id: taggerDryRun
      uses: anothrNick/github-tag-action@1.69.0
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        WITH_V: false
        DRY_RUN: true
    - name: echo new tag
      run: |
        echo "The next tag version will be: ${{ steps.taggerDryRun.outputs.new_tag }}"
    - name: echo tag
      run: |
        echo "The current tag is: ${{ steps.taggerDryRun.outputs.tag }}"
    - name: echo part
      run: |
        echo "The version increment was: ${{ steps.taggerDryRun.outputs.part }}"
    - name: 'Checkout GitHub Action'
      uses: actions/checkout@v4
    - name: 'Login to GitHub Container Registry'
      uses: docker/login-action@v3
      with:
        registry: ghcr.io
        username: ${{github.actor}}
        password: ${{secrets.GITHUB_TOKEN}}
    - name: 'Build runtime base image'
      run: |
        if docker manifest inspect ghcr.io/teamiumdev/matlabruntimebase:r2024a
        then
          echo "It exists. Skip the build..."
        else
          echo "Building new image.."
          docker build . --file "Dockerfile.deps" --tag ghcr.io/teamiumdev/matlabruntimebase:r2024a
```

```
45            docker push ghcr.io/teamiumdev/matlabruntimebase:r2024a
46          fi
47      - name: 'Build runtime image'
48        run: |
49          if docker manifest inspect ghcr.io/teamiumdev/matlabruntime:
                r2024a
50          then
51            echo "It exists. Skip the build..."
52          else
53            echo "Building new image.."
54            docker build . --file "Dockerfile.runtime" --tag ghcr.io/
                 teamiumdev/matlabruntime:r2024a
55            docker push ghcr.io/teamiumdev/matlabruntime:r2024a
56          fi
57      - name: 'Build Inventory Image'
58        run: docker build . --file Dockerfile --tag ghcr.io/teamiumdev/
              matlab-algorithm:${{ steps.taggerDryRun.outputs.new_tag }} --
              tag ghcr.io/teamiumdev/matlab-algorithm:latest
59      - name: Minor version for each merge
60        id: taggerFinal
61        uses: anothrNick/github-tag-action@1.69.0
62        env:
63          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
64          WITH_V: false
65      - name: 'Push Inventory Image'
66        run: docker push --all-tags ghcr.io/teamiumdev/matlab-algorithm
```

5. Deployment Considerations (Future)

Currently, the CI/CD pipeline concludes with the production of Docker images stored in GitHub Container Registry (GHCR). To fully leverage the benefits of containerization and streamlined deployment, here are potential deployment pathways to consider as the project scales:

Kubernetes for Orchestration:

- **Benefits:** Kubernetes is a powerful container orchestration platform ideal for managing complex deployments. Key advantages include:
  - **Scalability:** Easily deploy multiple instances of your MATLAB image to handle increased load.
  - **Resilience:** Kubernetes offers self-healing, automatic restarts, and rollout/rollback features for high availability.
  - **Resource Management:** Optimize resource usage across your application components.

Deployment Tools:

- **ArgoCD:** A GitOps deployment tool designed for Kubernetes. Synchronizes the desired state defined in Git with the running Kubernetes cluster for continuous deployment.
  - **Advantages:** Declarative configuration, automated updates when code changes, enhanced observability.
- **Jenkins:** A versatile automation server that can be used to orchestrate deployments to Kubernetes.
  - **Advantages:** Extensible, integrates well with existing pipelines and tools.

6. Conclusions

The implementation of a CI/CD pipeline using GitHub Actions has significantly streamlined the development and deployment process for the MATLAB algorithm project. Key benefits realized include:

- **Automation:** The pipeline automates repetitive build, versioning, and image publishing tasks. This reduces manual effort and the potential for human error.
- **Standardization:** Enforces a structured workflow, ensuring consistency in the way your Docker images are built, tagged, and released.
- **Code-driven Deployment:** Changes to the MATLAB algorithm codebase directly trigger the creation of new Docker images, leading to a tighter integration between development and deployment.

Future Directions

As the project's complexity and scaling requirements grow, integrating a Kubernetes-based deployment strategy with tools like ArgoCD or Jenkins offers the following potential advantages:

- **Scalability on Demand:** Ability to deploy multiple instances of the MATLAB algorithm image to handle fluctuating workloads efficiently.
- **Enhanced Resilience:** Automatic self-healing, rollbacks, and versioning for greater stability and uptime.
- **Optimized Resource Management:** Ensures compute and memory resources are effectively allocated across the entire application within the Kubernetes cluster.

By incorporating a robust CI/CD pipeline and laying the foundation for future Kubernetes integration, the MATLAB algorithm project is well-positioned for continued growth and adaptable deployment strategies.