

## Lab: Set up a CI/CD pipeline in GitHub Actions to build and deploy the container, including a simple test stage.

This lab walks you through the steps to set up a CI/CD pipeline using GitHub Actions to build and deploy a container, including a simple test stage.

---

### Prerequisites

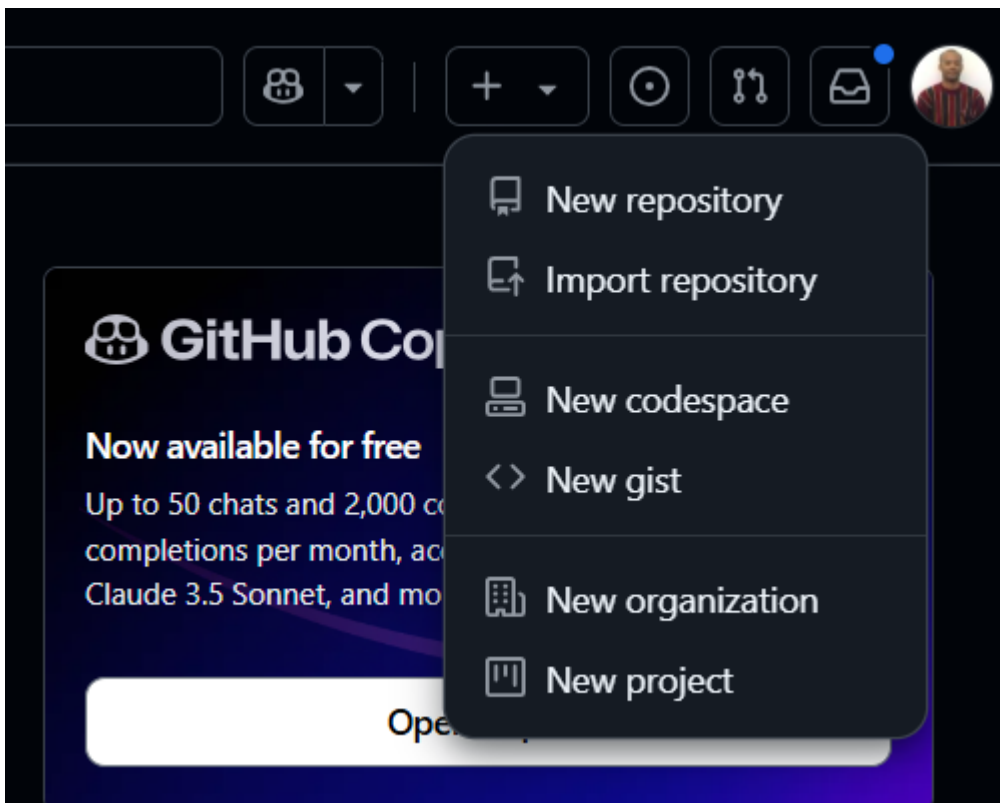
#### 1. Ubuntu Server:

- SSH access to your server (username and password).
- Docker already installed on the server.
- Pre-built Docker image named `my-dotnet-container` already present on the server. Connect with VM using SSH and verify image exists before proceeding to next step.

#### 2. GitHub Repository:

- A GitHub repository where your application code resides.
- Access to create and manage GitHub Actions workflows.

#### 3. Create a new repository



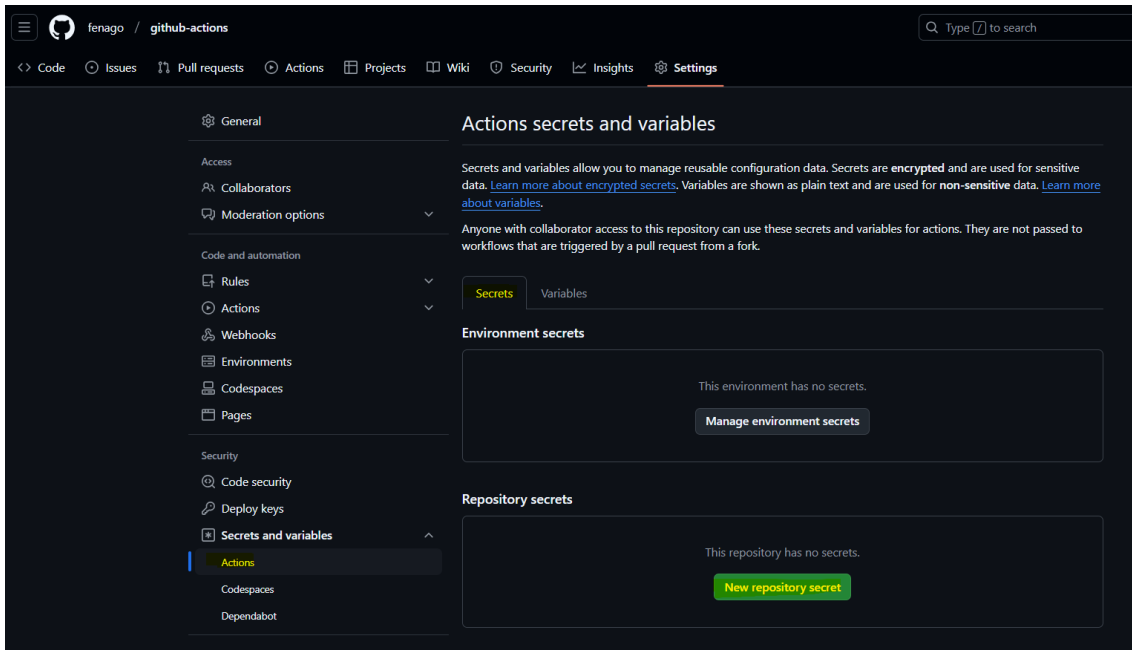
---

### Step 1: Add Secrets to GitHub Repository

1. Go to your GitHub repository.
2. Navigate to **Settings** > **Secrets and variables** > **Actions**.

3. Add the following secrets:

- `SERVER_USER` : SSH username for your server.
- `SERVER_PASSWORD` : SSH password for your server.
- `SERVER_IP` : IP address of your server.



## Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. [Learn more about encrypted secrets](#). Variables are shown as plain text and are used for **non-sensitive** data. [Learn more about variables](#).

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

Secrets

Variables

### Environment secrets

This environment has no secrets.

[Manage environment secrets](#)

### Repository secrets

[New repository secret](#)

Name 

Last updated



SERVER\_IP

now



SERVER\_PASSWORD

now



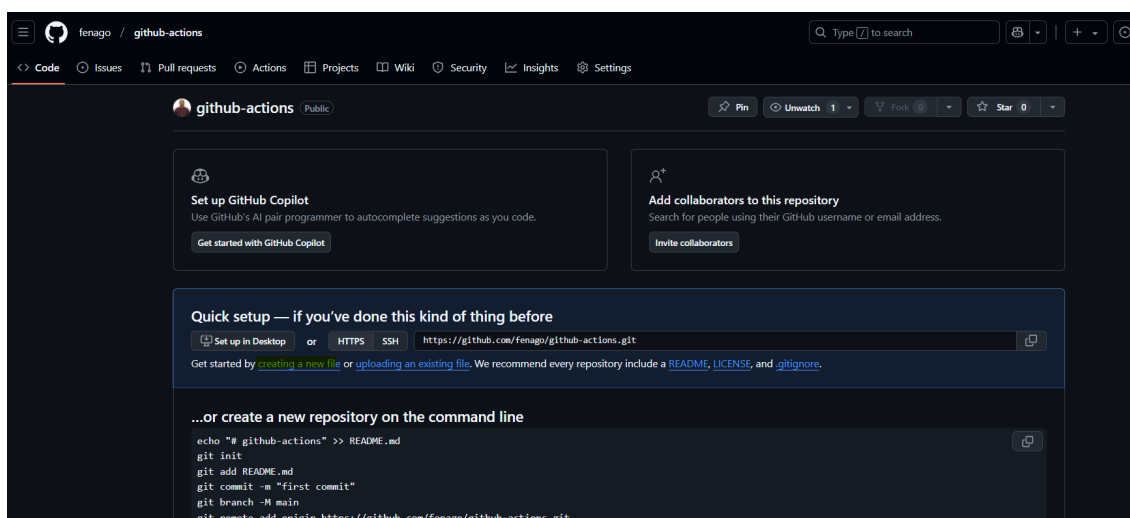
SERVER\_USER

1 minute ago



## Step 2: Add a GitHub Actions Workflow

1. Create a GitHub Actions workflow file in your repository by clicking **creating a new file**:



The screenshot shows the GitHub repository page for 'fenago / github-actions'. The repository is public and has 1 unwatch, 0 forks, and 0 stars. The 'Quick setup' section is highlighted, showing the repository URL: `https://github.com/fenago/github-actions.git`. Below the URL, it says: 'Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).' Below this, there is a section titled '...or create a new repository on the command line' with a code block containing the following commands:

```
echo "# github-actions" >> README.md
git init
git add README.md
git commit -m "First commit"
git branch -M main
git remote add origin https://github.com/fenago/github-actions.git
```

2. Create a new file: `.github/workflows/ci-cd.yml` .

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

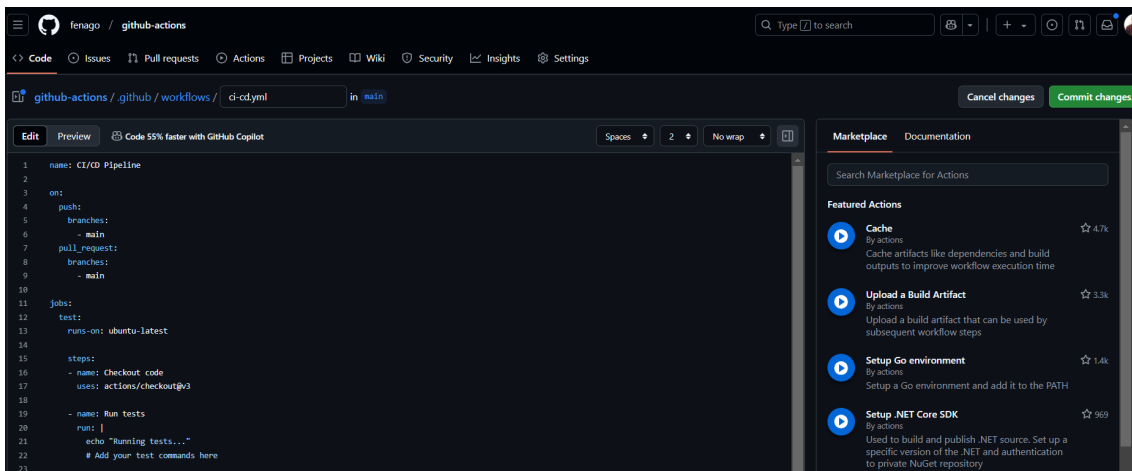
jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Run tests
        run: |
          echo "Running tests..."
          # Add your test commands here

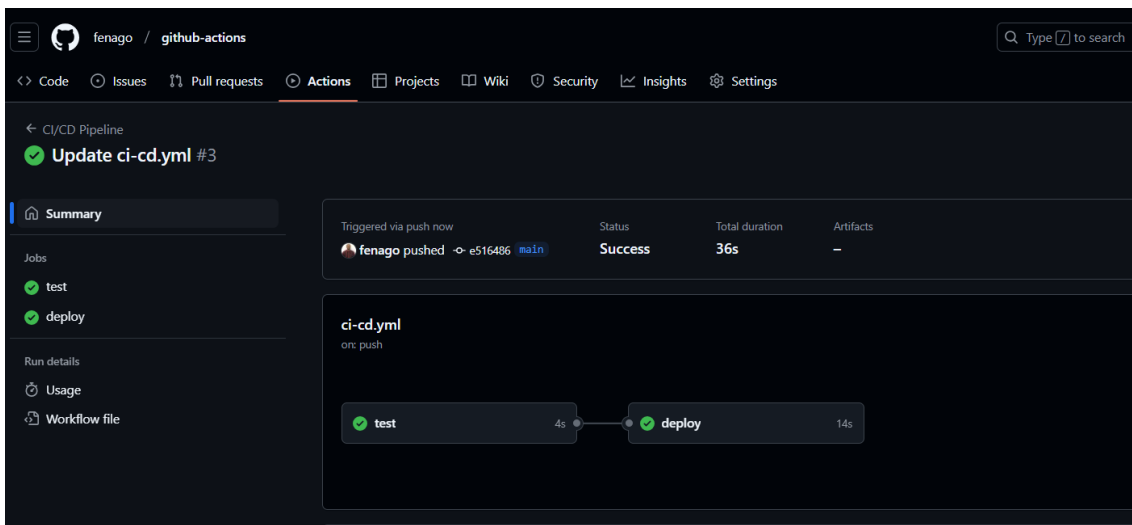
  deploy:
    runs-on: ubuntu-latest
    needs: test

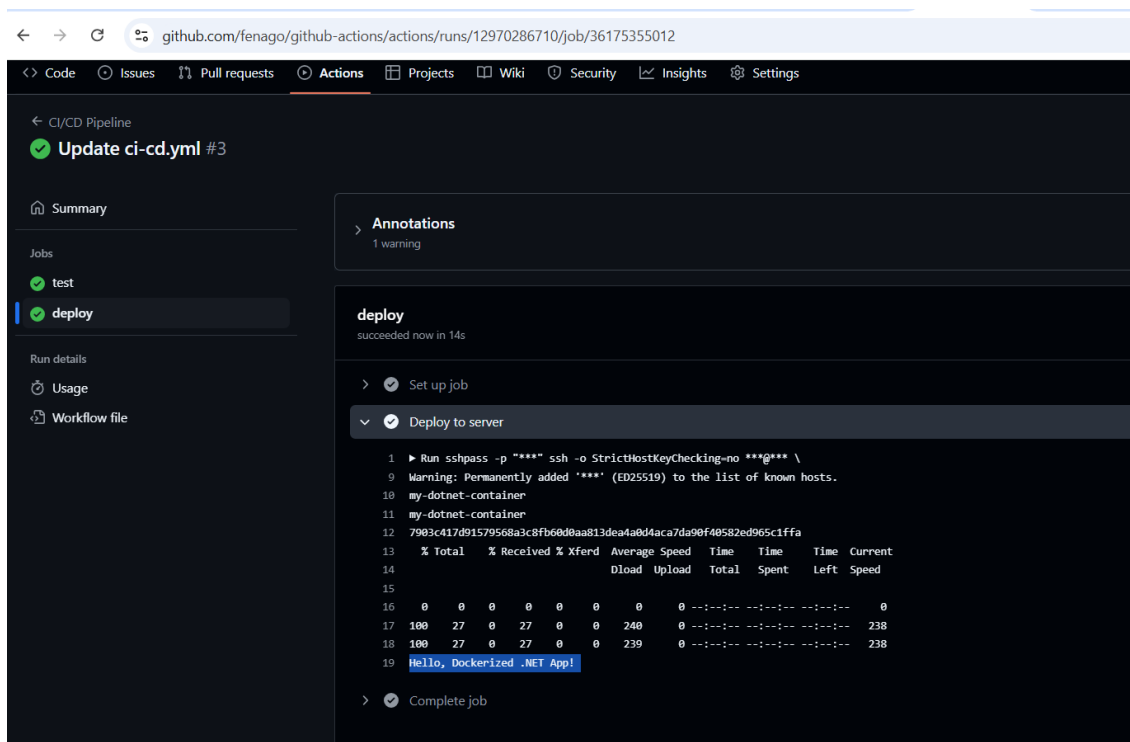
    steps:
      - name: Deploy to server
        run: |
          sshpass -p "${{ secrets.SERVER_PASSWORD }}" ssh -o StrictHostKeyChecking=no
          ${{ secrets.SERVER_USER }}@${{ secrets.SERVER_IP }} \
            "docker stop my-dotnet-container || true && \
            docker rm my-dotnet-container || true && \
            docker run -d --name my-dotnet-container -p 5000:5000 my-dotnet-app && \
            sleep 10 && \
            curl http://localhost:5000"
```



## Step 3: Test Your Workflow

1. Commit the changes to your repository:
2. Go to the **Actions** tab in your GitHub repository and check the progress of your workflow.





## Step 4: Verify Deployment

1. Log in to your server to confirm the container is running:

```
ssh username@server_ip
docker ps --filter "name=my-dotnet-container"
```

2. Test your application by accessing your server's IP address on port 5000 in a browser or using `curl` :

```
curl http://server_ip:5000
```

```

fenago@software-ecosystem-ubuntu22:~$ docker ps --filter "name=my-dotnet-container"
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
7903c417d915   my-dotnet-app  "dotnet MyDotNetApp..." 7 minutes ago  Up 7 minutes  0.0.0.0:5000->5000/tcp, :::5000->5000/tcp  my-dotnet-container
fenago@software-ecosystem-ubuntu22:~$

```

### Notes

- Ensure SSH and Docker ports are properly configured in your firewall.

You now have a fully functional CI/CD pipeline using GitHub Actions to deploy your pre-existing Docker containerized application `my-dotnet-container` to port 5000!