Steps to run CI and CD pipeline

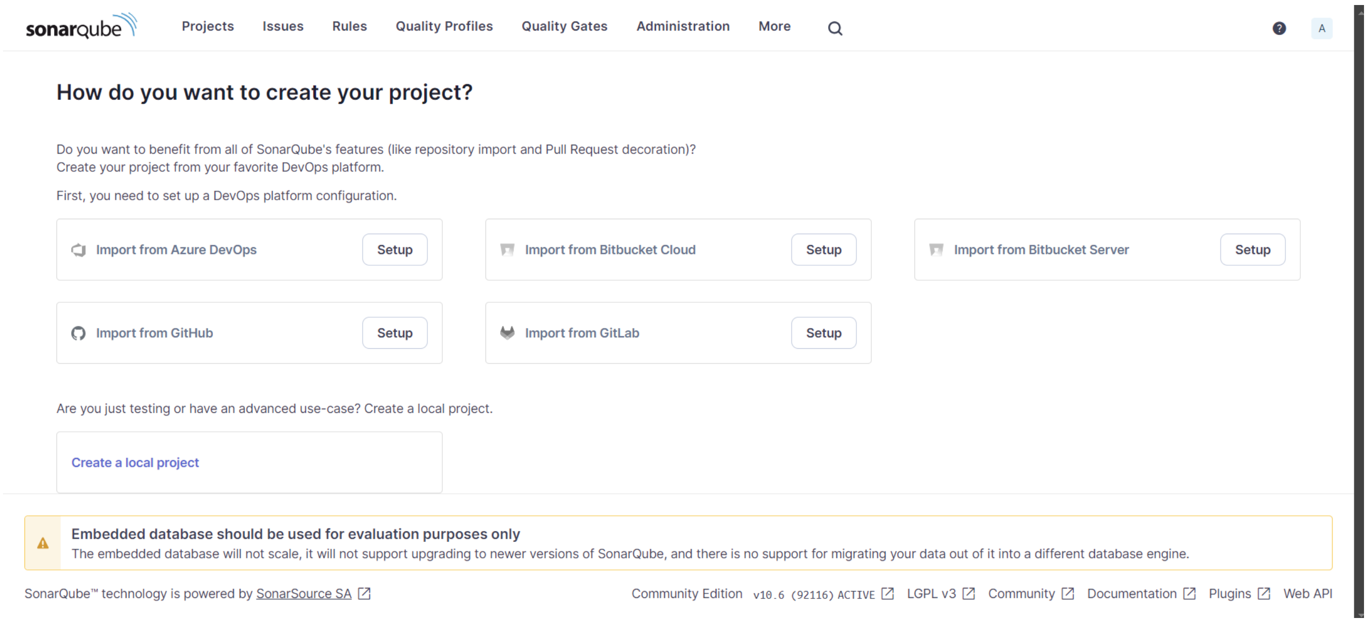
Steps to Setup CI

1. Run SonarQube

Start a SonarQube instance using Docker:

```
docker run --name sonarqube-custom -p 9000:9000 sonarqube:10.6-community
```

- Open a browser and navigate to `http://localhost:9000`
- Use the default credentials `admin/admin`.
- You'll be prompted to change the password.
- Select `Create a local project`



How do you want to create your project?

Do you want to benefit from all of SonarQube's features (like repository import and Pull Request decoration)?
Create your project from your favorite DevOps platform.

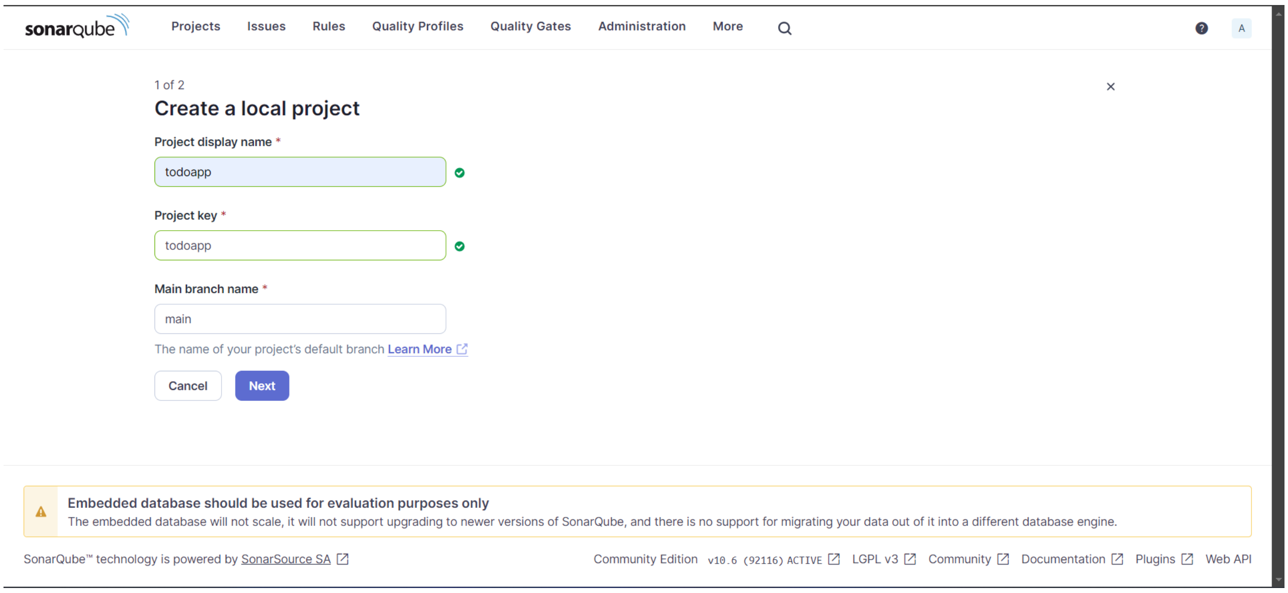
First, you need to set up a DevOps platform configuration.

Are you just testing or have an advanced use-case? Create a local project.

Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA   Community Edition v10.6 (92116) ACTIVE   LGPL v3   Community   Documentation   Plugins   Web API

- Set the following details:
  - **Project Display Name**: `todoapp`
  - **Project Key**: `todoapp`
  - **Branch Name**: `main`
  - Click **Next**.



1 of 2

Create a local project

Project display name *: todoapp

Project key *: todoapp

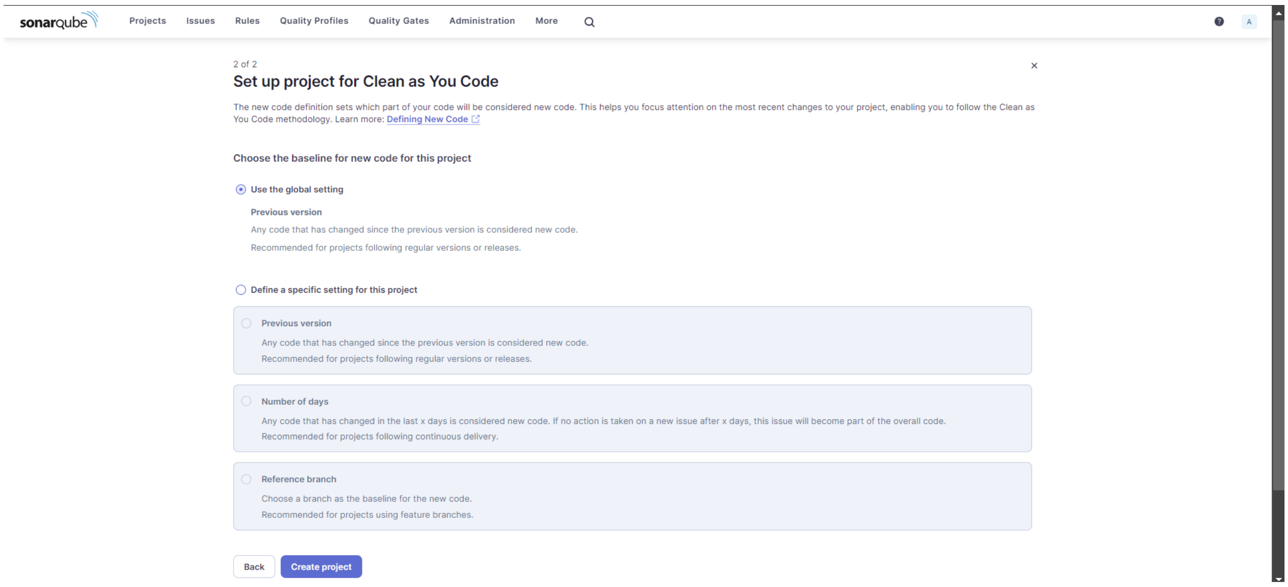
Main branch name *: main

The name of your project's default branch Learn More

Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA   Community Edition v10.6 (92116) ACTIVE LGPL v3 Community Documentation Plugins Web API

- Redirected to **Set up a project for Clean as You Code**.
  - Select **Use global settings**.
  - Click **Next**.



2 of 2

Set up project for Clean as You Code

The new code definition sets which part of your code will be considered new code. This helps you focus attention on the most recent changes to your project, enabling you to follow the Clean as You Code methodology. Learn more: Defining New Code

Choose the baseline for new code for this project

● Use the global setting

Previous version
Any code that has changed since the previous version is considered new code.
Recommended for projects following regular versions or releases.

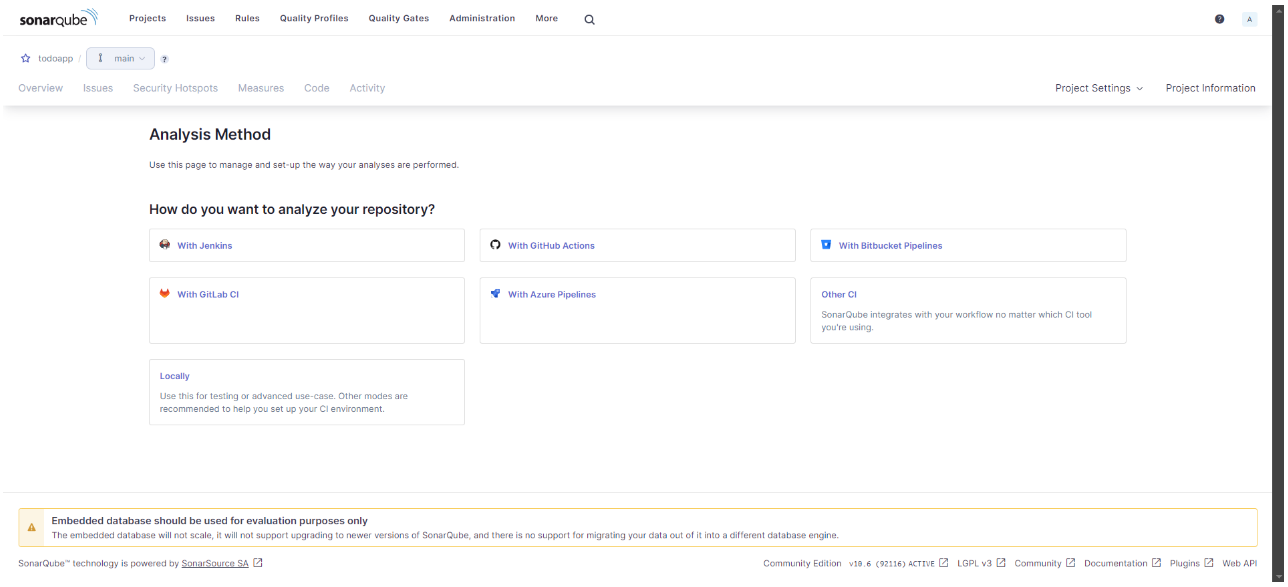
○ Define a specific setting for this project

Previous version
Any code that has changed since the previous version is considered new code.
Recommended for projects following regular versions or releases.

Number of days
Any code that has changed in the last x days is considered new code. If no action is taken on a new issue after x days, this issue will become part of the overall code.
Recommended for projects following continuous delivery.

Reference branch
Choose a branch as the baseline for the new code.
Recommended for projects using feature branches.

- Redirected to `Analysis Method`
  - Select with `Github Actions`



Analysis Method

Use this page to manage and set-up the way your analyses are performed.

How do you want to analyze your repository?

With Jenkins

With GitHub Actions

With Bitbucket Pipelines

With GitLab CI

With Azure Pipelines

Other CI

SonarQube integrates with your workflow no matter which CI tool you're using.

Locally

Use this for testing or advanced use-case. Other modes are recommended to help you set up your CI environment.

Embedded database should be used for evaluation purposes only

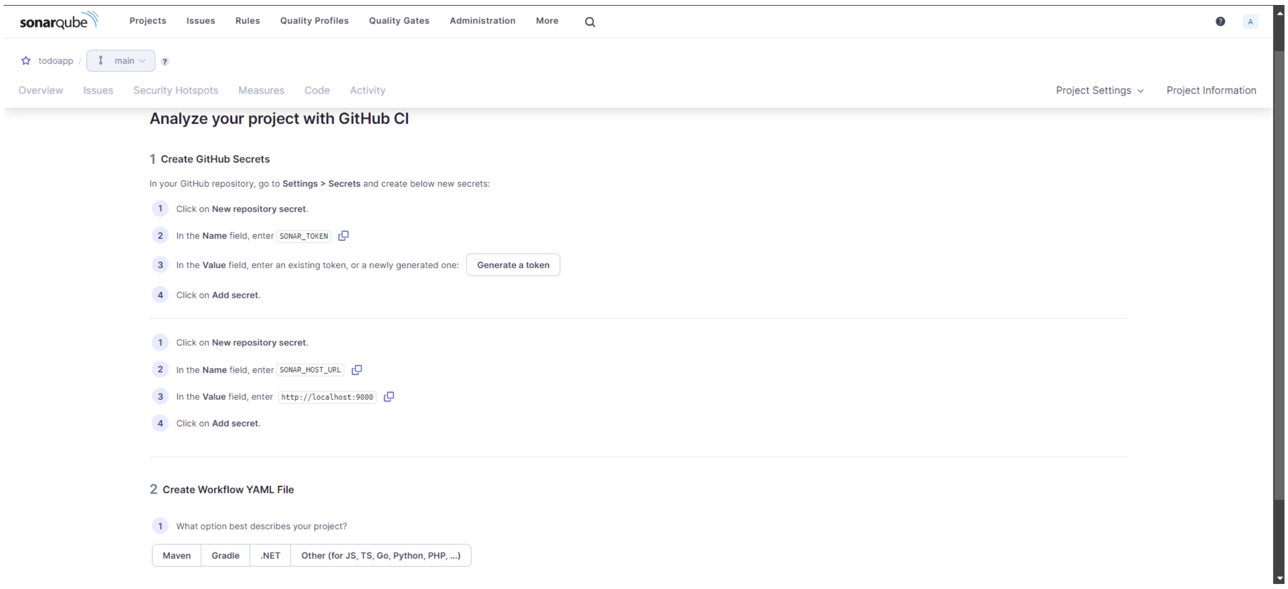
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA

Community Edition v10.6 (92116) ACTIVE LGPL v3 Community Documentation Plugins Web API

- Redirected to `Analyze you project with GitHub CI`
  - Generate a token and copy it.
  - Configure `pom.xml`: Update the properties section in your pom.xml file with the following

```xml
<properties>
  <sonar.projectKey>{SONAR_PROJECT_KEY}</sonar.projectKey>
  <sonar.host.url>{SONAR_HOST_URL}</sonar.host.url>
  <sonar.login>{SONAR_TOKEN}</sonar.login>
</properties>
```



Analyze your project with GitHub CI

1 Create GitHub Secrets

In your GitHub repository, go to **Settings > Secrets** and create below new secrets:

1. Click on **New repository secret**.
2. In the **Name** field, enter `SONAR_TOKEN`
3. In the **Value** field, enter an existing token, or a newly generated one:
4. Click on **Add secret**.

1. Click on **New repository secret**.
2. In the **Name** field, enter `SONAR_HOST_URL`
3. In the **Value** field, enter `http://localhost:9000`
4. Click on **Add secret**.

2 Create Workflow YAML File

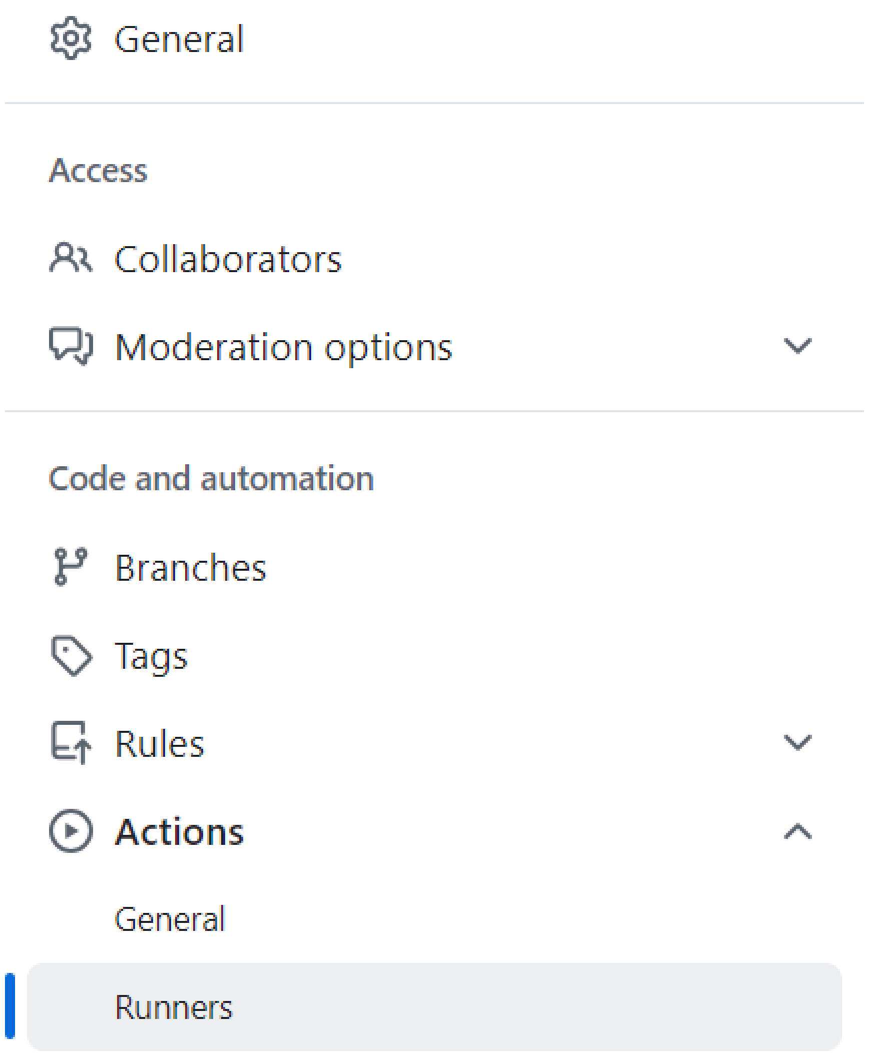
1. What option best describes your project?

To change the instructions for setting up a self-hosted runner for Windows instead of Linux, follow these steps:

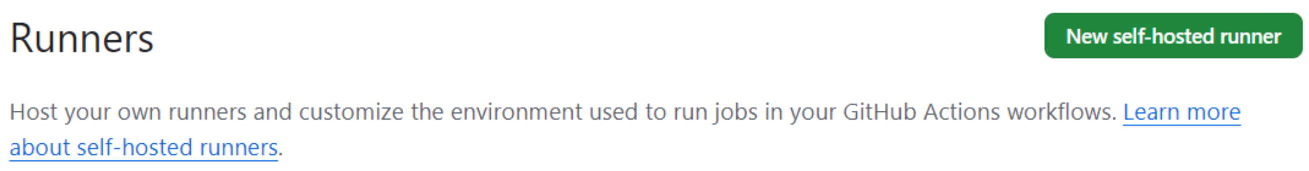
2. Run Self-Hosted on Windows

- Open the Command Prompt or PowerShell on your Windows machine.
- Go to the **Settings** tab of your repository on GitHub.

- Navigate to the **Actions** section and select **Runners**.



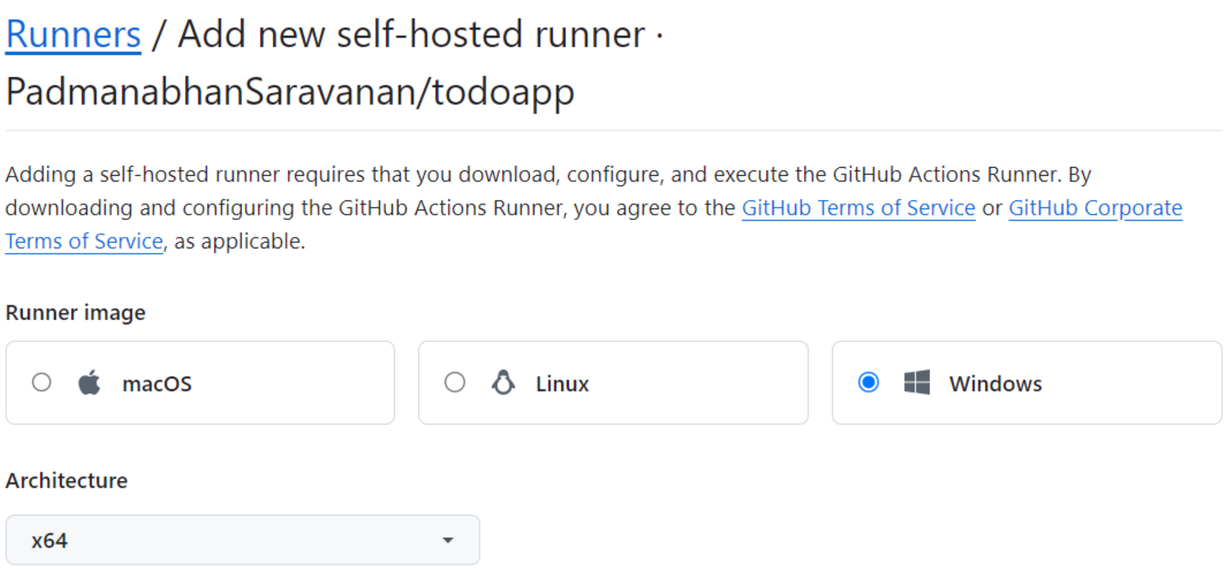
- Select **New self-hosted runner** to initiate the process of adding a self-hosted runner.



Runners

Host your own runners and customize the environment used to run jobs in your GitHub Actions workflows. Learn more about self-hosted runners.

- Choose the appropriate options:
  - **Operating System**: Select **Windows**.
  - **Architecture**: Select x64 (assuming your system architecture is 64-bit).



Runners / Add new self-hosted runner · PadmanabhanSaravanan/todoapp

Adding a self-hosted runner requires that you download, configure, and execute the GitHub Actions Runner. By downloading and configuring the GitHub Actions Runner, you agree to the GitHub Terms of Service or GitHub Corporate Terms of Service, as applicable.

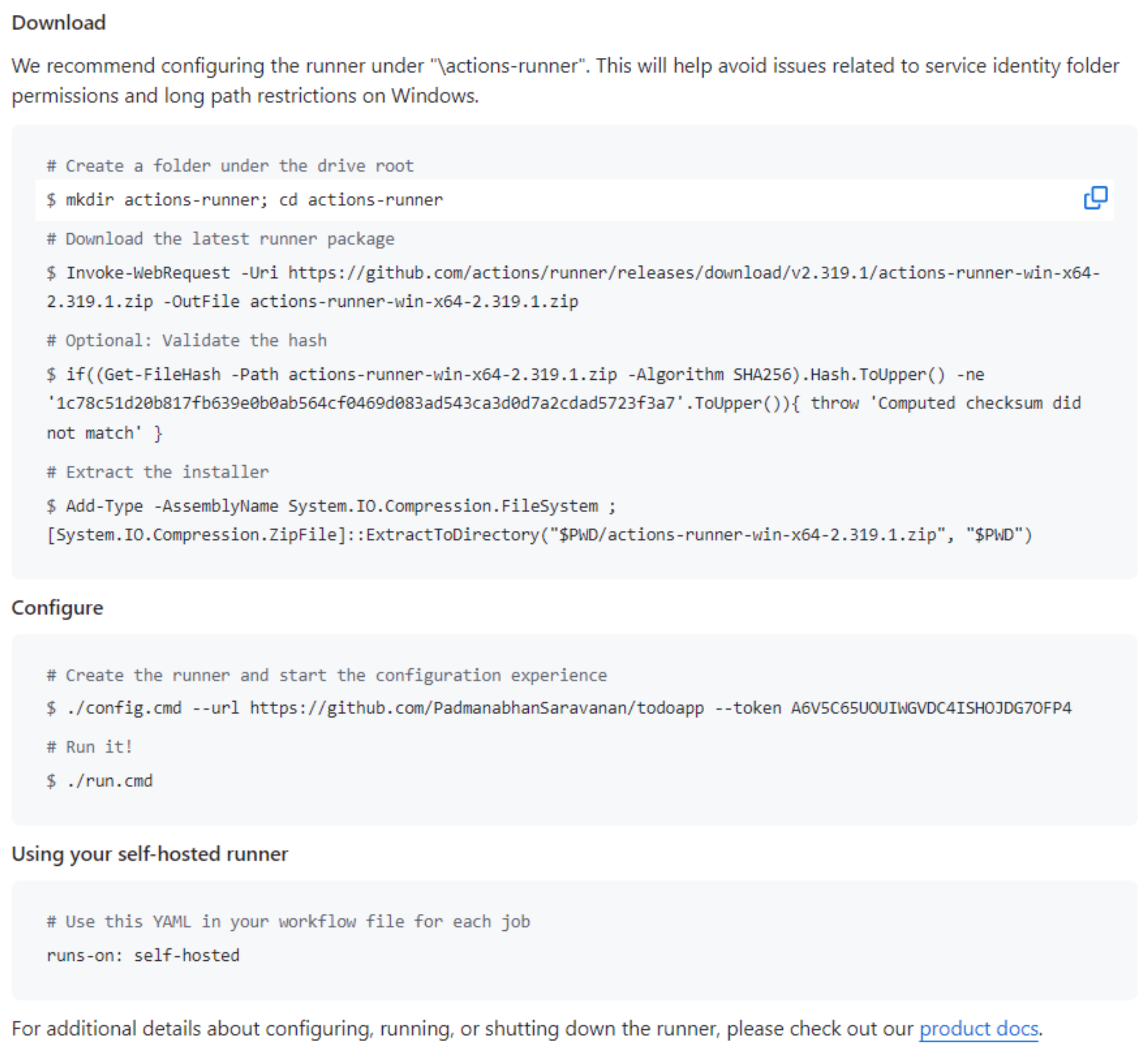
Runner image

macOS Linux [✓ Windows]

Architecture

x64

- After selecting **Windows**, follow the provided steps to set up the self-hosted runner. This typically involves:
    - Downloading the runner package for Windows.
    - Extracting the downloaded package.
    - Running the `config.cmd` command to configure the runner using the provided instructions.
    - Starting the runner using `run.cmd`.



Download

We recommend configuring the runner under "\actions-runner". This will help avoid issues related to service identity folder permissions and long path restrictions on Windows.

```
# Create a folder under the drive root
$ mkdir actions-runner; cd actions-runner
# Download the latest runner package
$ Invoke-WebRequest -Uri https://github.com/actions/runner/releases/download/v2.319.1/actions-runner-win-x64-
2.319.1.zip -OutFile actions-runner-win-x64-2.319.1.zip
# Optional: Validate the hash
$ if((Get-FileHash -Path actions-runner-win-x64-2.319.1.zip -Algorithm SHA256).Hash.ToUpper() -ne
'1c78c51d20b817fb639e0b0ab564cf0469d083ad543ca3d0d7a2cdad5723f3a7'.ToUpper()){ throw 'Computed checksum did
not match' }
# Extract the installer
$ Add-Type -AssemblyName System.IO.Compression.FileSystem ;
[System.IO.Compression.ZipFile]::ExtractToDirectory("$PWD/actions-runner-win-x64-2.319.1.zip", "$PWD")
```

Configure

```
# Create the runner and start the configuration experience
$ ./config.cmd --url https://github.com/PadmanabhanSaravanan/todoapp --token A6V5C65UOUIWGVDC4ISHOJDG7OFP4
# Run it!
$ ./run.cmd
```

Using your self-hosted runner

```
# Use this YAML in your workflow file for each job
runs-on: self-hosted
```

For additional details about configuring, running, or shutting down the runner, please check out our product docs.

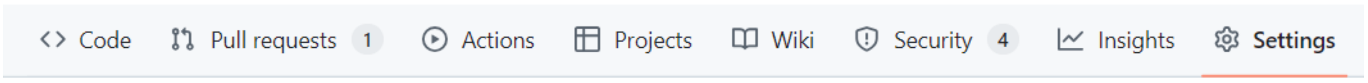
- After configuration, the self-hosted runner should be operational and ready to execute workflows for your repository. You can use it in your workflow YAML files by specifying `runs-on: self-hosted`.

3. Configure Docker

To set up Docker secrets (`DOCKER_USERNAME` and `DOCKER_PASSWORD`) in your GitHub repository, follow these detailed steps:

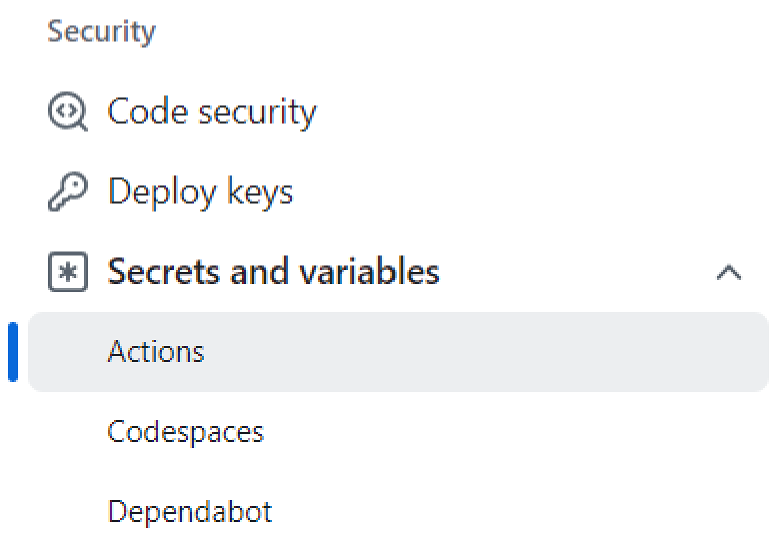
Step 1: Access Repository Settings

- Go to your GitHub repository.
- Click on the **Settings** tab located at the top of your repository page.



Step 2: Navigate to Secrets and Variables

- In the left sidebar, scroll down to the **Security** section.
- Click on **Secrets and variables**, then select **Actions**.



Step 3: Add Secrets

- Click on the **New repository secret** button.
- For `DOCKER_USERNAME`:
  - In the **Name** field, enter `DOCKER_USERNAME`.
  - In the **Secret** field, input your Docker Hub username.
  - Click the **Add secret** button.
- For `DOCKER_PASSWORD`:
  - Click **New repository secret** again.
  - In the **Name** field, enter `DOCKER_PASSWORD`.
  - In the **Secret** field, input your Docker Hub password or access token.
  - Click the **Add secret** button.

Your Docker credentials are now securely stored in your GitHub repository's secrets and can be accessed in your workflows using `${{ secrets.DOCKER_USERNAME }}` and `${{ secrets.DOCKER_PASSWORD }}`.

4. CI.yaml

```
name: Continuous Integration

on:
  # push:
  #   branches:
  #     - main
  workflow_dispatch:

jobs:
  checkout:
    runs-on: self-hosted
    steps:
      - uses: actions/checkout@v4
        with:
```

```yaml
          fetch-depth: 0  # Shallow clones should be disabled for a better
relevancy of analysis

  cache:
    runs-on: self-hosted
    needs: checkout
    steps:
      - name: Cache SonarQube packages
        uses: actions/cache@v4.0.2
        with:
          path: ~/.sonar/cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar

      - name: Cache Maven packages
        uses: actions/cache@v4.0.2
        with:
          path: ~/.m2
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
          restore-keys: ${{ runner.os }}-m2

  jacoco:
    runs-on: ubuntu-latest  # Change to Linux-based runner
    needs: cache
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Maven
        uses: stCarolas/setup-maven@v5
        with:
          maven-version: 3.8.2

      - name: Build with Maven
        run: mvn -B package -Pcoverage

      - name: Generate JaCoCo badge
        id: jacoco
        uses: cicirello/jacoco-badge-generator@v2
        with:
          badges-directory: badges
          generate-branches-badge: true
          generate-summary: true

      - name: Log coverage percentages to workflow output
        run: |
          echo "coverage = ${{ steps.jacoco.outputs.coverage }}"
          echo "branches = ${{ steps.jacoco.outputs.branches }}"

      - name: Upload JaCoCo coverage report as a workflow artifact
        uses: actions/upload-artifact@v4.4.0
        with:
          name: jacoco-report
          path: target/site/jacoco/
  sonarqube:
    runs-on: self-hosted
    needs: jacoco
    steps:
      - name: Set up JDK 17
        uses: actions/setup-java@v1
        with:
          java-version: 17

      - name: Set up Maven
        uses: stCarolas/setup-maven@v5
        with:
          maven-version: 3.8.2

      - name: Check SonarQube accessibility
        shell: powershell
        run: |
          $Response = Invoke-WebRequest -Uri http://localhost:9000 -Method Head -
ErrorAction Stop
          if ($Response.StatusCode -eq 200) {
              Write-Output "SonarQube is accessible."
          } else {
            Write-Output "SonarQube is not accessible."
            exit 1
          }

      - name: Download JaCoCo coverage report
        uses: actions/download-artifact@v4.1.8
        with:
          name: jacoco-report
          path: target/site/jacoco/  # Ensure this matches the upload path

      - name: Build and analyze
        run: mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar

  artifacts:
    runs-on: self-hosted
    needs: sonarqube
    steps:
      - name: Copy JAR file to staging
        run: |
          mkdir staging
          Copy-Item target\*.jar staging
        shell: powershell

      - uses: actions/upload-artifact@v4
        with:
          name: Package
          path: staging

  release:
    runs-on: ubuntu-latest
    needs: artifacts
    steps:
      - name: Download web-app content
        uses: actions/download-artifact@v4.1.8
        with:
          name: Package

      - name: View content
        run: ls -R

      - name: Archive site content
        uses: thedoctor0/zip-release@master
        with:
          filename: app.zip

      - name: Create GitHub release
        id: create-new-release
        uses: actions/create-release@v1
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        with:
          tag_name: ${{ github.ref_type }}
          release_name: Release ${{ github.ref_type }}

      - name: Upload release asset
        uses: actions/upload-release-asset@v1
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        with:
          upload_url: ${{ steps.create-new-release.outputs.upload_url }}
          asset_path: ./app.zip
          asset_name: app-v${{ github.ref_type }}.zip
          asset_content_type: application/zip

  docker:
    runs-on: ubuntu-latest  # Changed to ubuntu-latest
    needs: release
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Maven
        uses: stCarolas/setup-maven@v5
        with:
          maven-version: 3.8.2

      - name: Build with Maven
        run: mvn -B package

      - name: Build Docker image
        run: docker build -t vijaynvb/todoapp .

      - name: Log in to Docker Hub
        run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u "${{
secrets.DOCKER_USERNAME }}" --password-stdin
```

```
      - name: Push image to Docker Hub
        run: docker push vijaynvb/todoapp
```

Steps to Setup CD

- Log in to your Google Cloud Console.
- In the left-hand menu, go to `IAM & Admin` > `Service Accounts`.
- Create a service account named `githubactions`.
- Assign the following roles to the service account:
  - `Kubernetes Engine Admin`
  - `Kubernetes Cluster Admin`
  - `Editor`
- Once the service account is created, you'll see it listed on the Service Accounts page
- Click on the Actions (three vertical dots) next to your service account and select Manage keys.
- Click `Add Key` > `Create New Key`.
- Select `JSON` as the key type and click Create
- Download the `JSON` key for this service account.
- Add the following secrets to your GitHub repository:
  - `GCP_PROJECT_ID`: Your Google Cloud `project ID`(find project id in google cloud console)
  - `GCP_SA_KEY`: The `JSON` key you downloaded
  - `GCP_SERVICE_ACCOUNT`: The email address of the service account

main.tf

```
provider "google" {
  project = var.project_id
  region  = var.region
}

resource "google_container_cluster" "primary" {
  name     = var.cluster_name
  location = var.location

  initial_node_count = 3

  node_config {
    machine_type = "e2-medium"
    oauth_scopes = [
      "https://www.googleapis.com/auth/cloud-platform",
    ]
    service_account = var.service_account
  }

  deletion_protection = false
}

variable "project_id" {
  description = "The ID of the GCP project"
}

variable "region" {
  description = "The GCP region"
}

variable "location" {
  description = "The GCP zone or location"
}

variable "cluster_name" {
  description = "The name of the GKE cluster"
}

variable "service_account" {
  description = "The service account for the GKE cluster"
}
```

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todoapih2
spec:
  replicas: 3
  selector:
    matchLabels:
      app: todo
  template:
    metadata:
      labels:
        app: todo
    spec:
      containers:
      - name: todoapih2
        image: vijaynvb/todoapp
        ports:
        - containerPort: 8081


---

apiVersion: v1
```

```yaml
kind: Service
metadata:
  name: svctodoh2api
spec:
  selector:
    app: todo
  ports:
    - port: 80
      targetPort: 8081
  type: LoadBalancer
```

cd.yaml

```yaml

name: Continuous Deployment

on:
  workflow_run:
    workflows: ["Continuous Integration"]
    types:
      - completed
  workflow_dispatch:

env:
  CREDENTIALS: ${{ secrets.GCP_SA_KEY }}
  PROJECT_ID: ${{ secrets.GCP_PROJECT_ID }}
  GKE_CLUSTER: gke-todoapp-cluster   # cluster name
  GKE_REGION: us-east1
  GKE_LOCATION: us-east1-b     # cluster location
  SERVICE_ACCOUNT: ${{ secrets.GCP_SERVICE_ACCOUNT }}
  TFSTATE_CACHE_KEY: terraform-state-${{ github.sha }}

jobs:
  create-cluster:
    if: ${{ github.event.workflow_run.conclusion == 'success' }}
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v1

      - name: Terraform Init
        run: terraform init
        env:
          GOOGLE_CREDENTIALS: ${{ env.CREDENTIALS }}

      - name: Terraform Plan
        run: terraform plan -input=false
        env:
          GOOGLE_CREDENTIALS: ${{ env.CREDENTIALS }}
          TF_VAR_project_id: ${{ env.PROJECT_ID }}
          TF_VAR_region: ${{ env.GKE_REGION }}
          TF_VAR_location: ${{ env.GKE_LOCATION }}
          TF_VAR_cluster_name: ${{ env.GKE_CLUSTER }}
          TF_VAR_service_account: ${{ env.SERVICE_ACCOUNT }}

      - name: Terraform Apply
        id: terraform-apply
        run: terraform apply -auto-approve -input=false
        env:
          GOOGLE_CREDENTIALS: ${{ env.CREDENTIALS }}
          TF_VAR_project_id: ${{ env.PROJECT_ID }}
          TF_VAR_region: ${{ env.GKE_REGION }}
          TF_VAR_location: ${{ env.GKE_LOCATION }}
          TF_VAR_cluster_name: ${{ env.GKE_CLUSTER }}
          TF_VAR_service_account: ${{ env.SERVICE_ACCOUNT }}

      - name: Cache Terraform State
        uses: actions/cache@v3
        with:
          path: |
            .terraform/
            terraform.tfstate
            terraform.tfstate.backup
          key: ${{ env.TFSTATE_CACHE_KEY }}

  deploy-to-cluster:
    runs-on: ubuntu-latest
    needs: create-cluster
    outputs:
      baseurl: ${{ steps.extract-url.outputs.baseurl }}
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Install Google Cloud SDK
        uses: 'google-github-actions/auth@v2'
        with:
          credentials_json: ${{ env.CREDENTIALS }}

      - name: Set up Cloud SDK
        uses: 'google-github-actions/setup-gcloud@v2'

      - name: Configure kubectl to use gke-gcloud-auth-plugin
        run: gcloud components install kubectl

      - name: Get GKE credentials
        run: gcloud container clusters get-credentials ${{ env.GKE_CLUSTER }} --
zone ${{ env.GKE_LOCATION }} --project ${{ secrets.GCP_PROJECT_ID }}

      - name: Deploy application to GKE
        run: kubectl apply -f deployment.yaml

      - name: Wait for the service to be ready
        run: |
          echo "Waiting for 60 seconds..."
          sleep 60
      - name: Extract baseurl from GKE service
        id: extract-url
        run: |
          EXTERNAL_IP=$(kubectl get svc svctodoh2api -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')
          PORT=$(kubectl get svc svctodoh2api -o jsonpath='{.spec.ports[0].port}')
          echo "EXTERNAL_IP is: $EXTERNAL_IP"
          echo "PORT is: $PORT"
          echo "baseurl=http://$EXTERNAL_IP:$PORT" >> $GITHUB_OUTPUT

  run-postman-tests:
    runs-on: ubuntu-latest
    needs: deploy-to-cluster
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Install Newman
        run: npm install -g newman

      - name: Run Postman Collection
        env:
          BASEURL: ${{ needs.deploy-to-cluster.outputs.baseurl }}
        run: |
          newman run postman_collection.json --env-var "base_url=${{ env.BASEURL
}}"

  destroy-cluster:
    runs-on: ubuntu-latest
    needs: [deploy-to-cluster, run-postman-tests]
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Restore Terraform State Cache
        uses: actions/cache@v3
        with:
          path: |
            .terraform/
            terraform.tfstate
            terraform.tfstate.backup
          key: ${{ env.TFSTATE_CACHE_KEY }}

      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v1

      - name: Terraform Init
        run: terraform init
        env:
          GOOGLE_CREDENTIALS: ${{ env.CREDENTIALS }}

      - name: Terraform Destroy
```

```
        run: terraform destroy -auto-approve -input=false
        env:
          GOOGLE_CREDENTIALS: ${{ env.CREDENTIALS }}
          TF_VAR_project_id: ${{ env.PROJECT_ID }}
          TF_VAR_region: ${{ env.GKE_REGION }}
          TF_VAR_location: ${{ env.GKE_LOCATION }}
          TF_VAR_cluster_name: ${{ env.GKE_CLUSTER }}
          TF_VAR_service_account: ${{ env.SERVICE_ACCOUNT }}
```