

SONARCLOUD VIA GITHUB ACTIONS

PETRO KOLOSOV

ABSTRACT. Explains how to integrate SonarCloud with GitHub actions in a simple and quick approach.

CONTENTS

1. Introduction	1
1.1. Login to the SonarCloud	1
1.2. Create an organization at SonarCloud	2
1.3. Create a project on behalf of organization	3

1. INTRODUCTION

First choose which repository to analyze, it is rather obvious initial step, however. I choose to test it on my old repository github.com/kolosovpetro/IoC-Container.

1.1. **Login to the SonarCloud.** Next we login to the sonarcloud.io using your GitHub account

Date: October 31, 2022.

Key words and phrases. SonarCloud, Github Actions, DevOps, CICD .

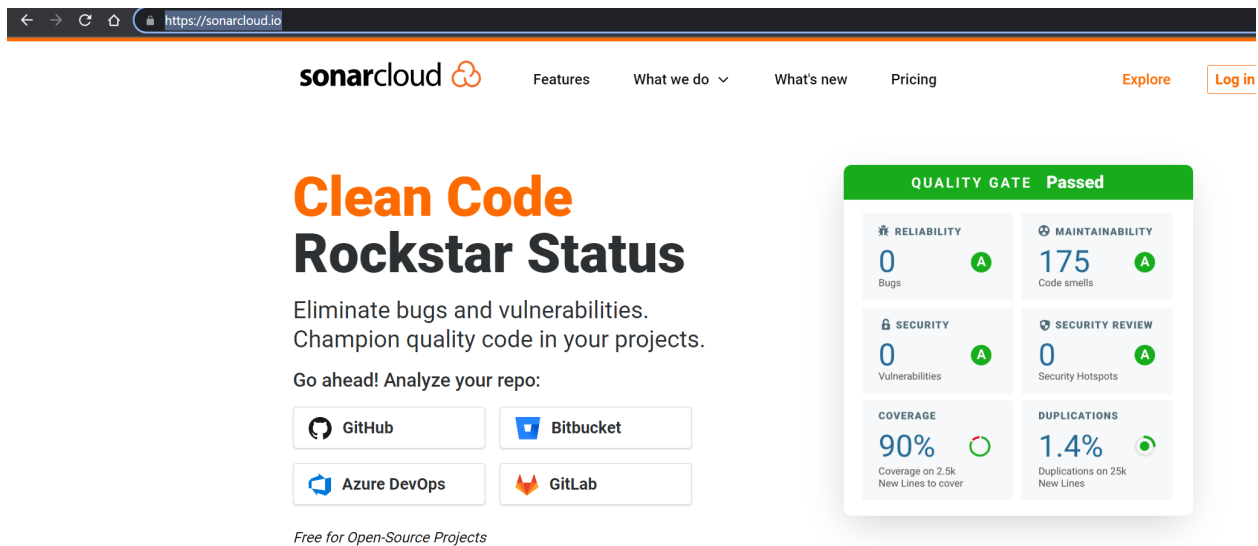


Figure 1. SonarCloud login page. Use login via GutHub.

1.2. **Create an organization at SonarCloud.** Now we create an organization we use to analyze your project, I created it as

petro-kolosov-own-repos

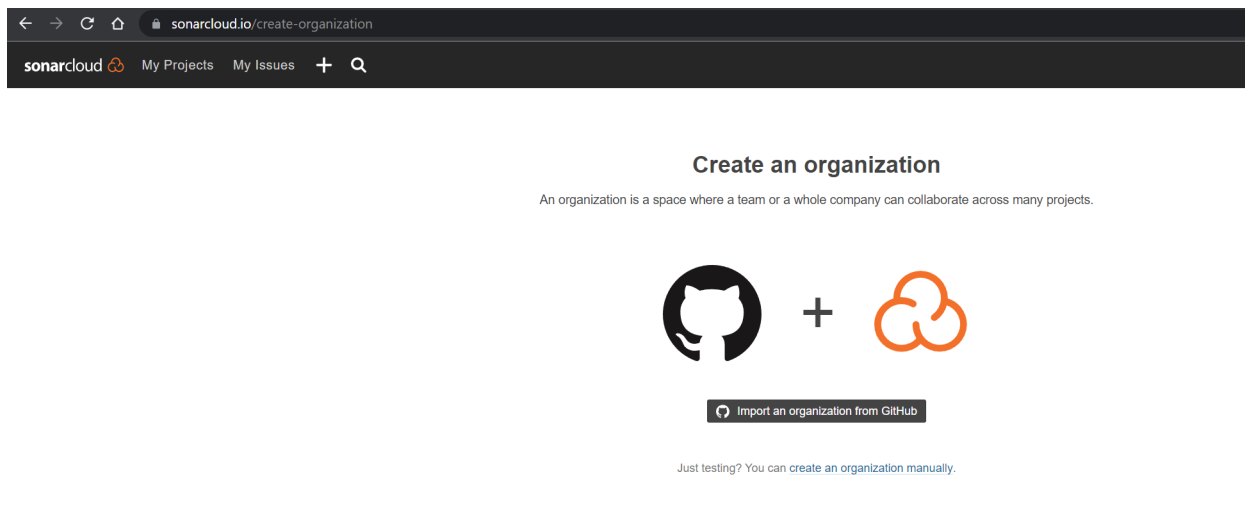


Figure 2. Create organization form, use create an organization manually at the bottom.

Then enter the key of the organization

Create an organization

An organization is a space where a team or a whole company can collaborate across many projects.

1 Enter your organization details

Manual setup is not recommended, and leads to missing features like appropriate setup of your project or analysis feedback in the Pull Request.

Key*

petro-kolosov-own-repo

Organization key must start with a lowercase letter or number, followed by lowercase letters, numbers or hyphens, and must end with a letter or number. Maximum length: 255 characters.

[Add additional info](#)

[Continue](#)

Figure 3. Enter the key of the organization.

Next, choose a plan for your organization, I use free one

Create an organization

An organization is a space where a team or a whole company can collaborate across many projects.

1 Enter your organization details petro-kolosov-own-repo

2 Choose a plan

Free plan €0

All projects you analyze will be public.
Anyone can browse source code.

Paid plan from €10

Unlimited private projects
 Strict control over who can view your private data
 No commitments, cancel anytime
 14 days free trial.

[Learn more](#)

[Create Organization](#)

We recommend to import your organization.

Figure 4. Choose a plan for your organization.

Click create then, so we have finished second step, creating of the organization.

1.3. **Create a project on behalf of organization.** At your recently created organization, click the **Analyze new project** button

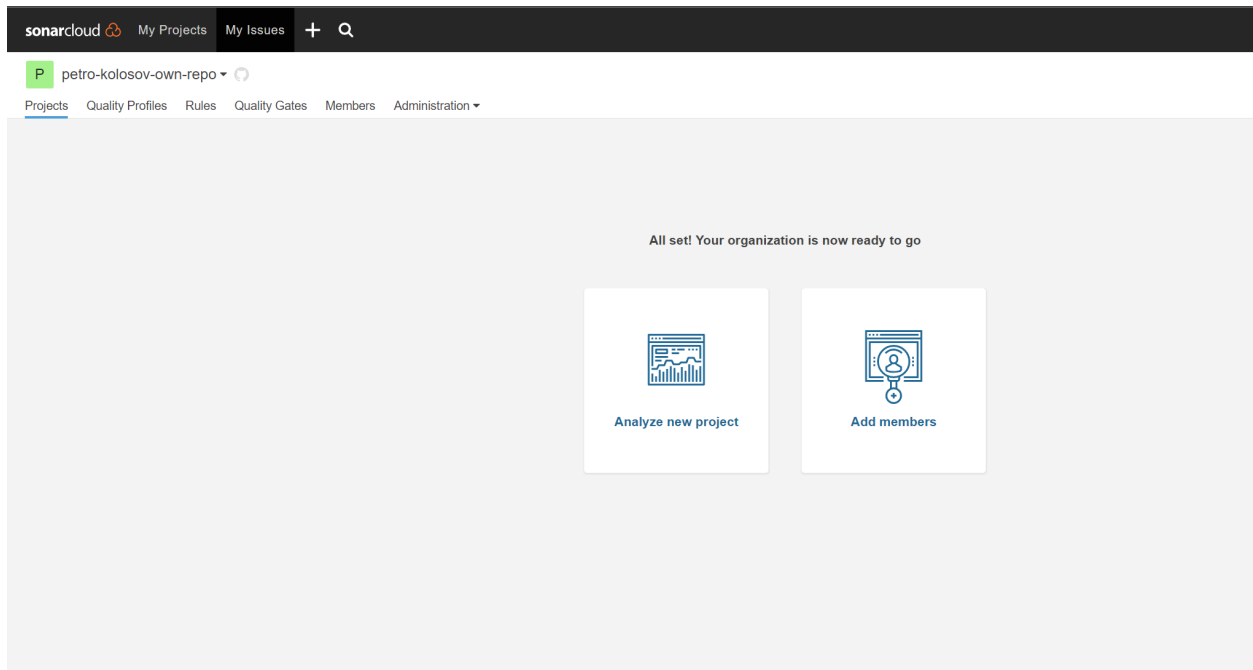


Figure 5. Click the Analyze new project button

Configure the name of your project

Analyze projects - Set up manually

i Manual setup is not recommended, and leads to missing features like appropriate setup of your project or analysis feedback in the Pull Request.

Organization*

petro-kolosov-own-repo petro-kolosov-own-repo [Create another organization](#)

Project key* **?**

loC-Container-New **LS** **✓**

Up to 400 characters. All letters, digits, dash, underscore, period or colon.

Display name* **?**

loC-Container-New **✓**

Up to 255 characters

Public

Anyone will be able to browse your source code and see the result of your analysis.

Private

Only members of the organization will be able to browse your source code and see the result of your analysis.

[Set Up](#)

Figure 6. Configure the name of your project

Click Set Up and choose Configure With Github Actions, so that it looks like

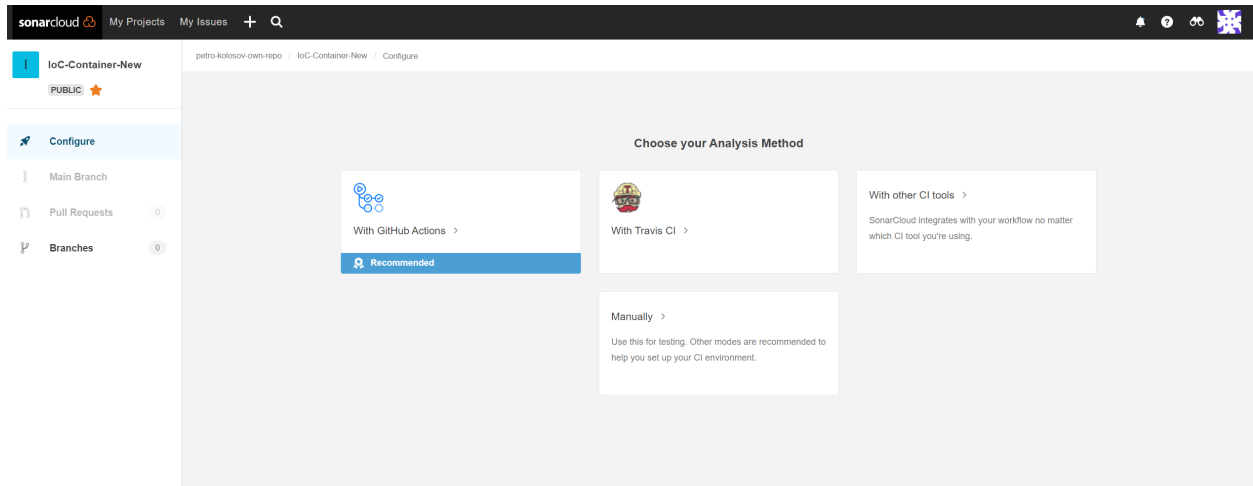


Figure 7. Configure With Github Actions.

Create a specified GitHub actions secret at your repository on GitHub

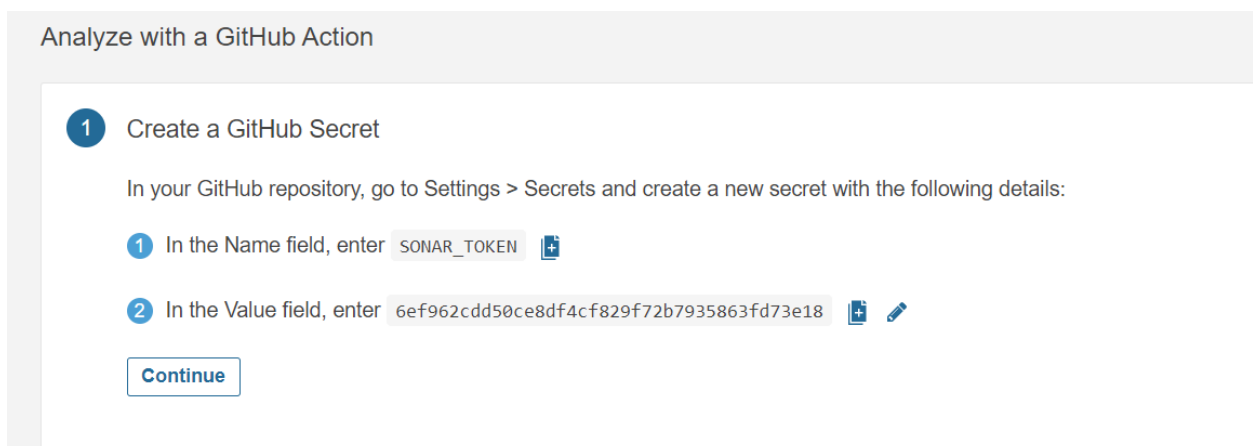


Figure 8. Create a specified GitHub actions secret.

Do not worry, this key won't work and here just for example. Clicking **Continue** and then .NET explores an example of the pipeline we are able to use on the GitHub as an action of our repository.

2 Create or update a `.github/workflows/build.yml` file

What option best describes your build?

Maven
 Gradle
 C, C++ or ObjC
 .NET
 Other (for JS, TS, Go, Python, PHP, ...)

Create or update your `.github/workflows/build.yml` [+](#)

Here is a base configuration to run a SonarCloud analysis on your master branch and Pull Requests. If you already have, you can just add some of these new steps to an existing one.

```

name: Build
on:
  push:
    branches:
      - master
  pull_request:
    types: [opened, synchronize, reopened]
jobs:
  build:
    name: Build
    runs-on: windows-latest
    steps:
      - name: Set up JDK 11
        uses: actions/setup-java@v1
        with:
          java-version: 1.11
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0 # Shallow clones should be disabled for a better relevancy of analysis
      - name: Cache SonarCloud packages
        uses: actions/cache@v1
        with:
          path: ~\sonar\cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
      - name: Cache SonarCloud scanner
        id: cache-sonar-scanner
        uses: actions/cache@v1
  
```

Figure 9. SonarCloud analyzer pipeline example.

Above pipeline runs on the **Windows** runner, but I prefer ubuntu runner for the reasons of performance so that I change it following way

```

name: Run SonarCloud analysis
on:
  push:
    branches: [ develop ]
  pull_request:
    types: [ opened, synchronize, reopened ]
  
```

```
workflow_dispatch:
jobs:
  run-sonarcloud-analysis:
    name: Run SonarCloud Analysis
    runs-on: ubuntu-latest
    steps:
      - name: Set up JDK 11
        uses: actions/setup-java@v1
        with:
          java-version: 1.11
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0

      - name: Cache SonarCloud packages
        uses: actions/cache@v1
        with:
          path: ~/sonar/cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar

      - name: Cache SonarCloud scanner
        id: cache-sonar-scanner
        uses: actions/cache@v1
        with:
          path: ~/.sonar/scanner
          key: ${{ runner.os }}-sonar-scanner
          restore-keys: ${{ runner.os }}-sonar-scanner

      - name: Setup .NET 6 SDK
        uses: actions/setup-dotnet@v1
        with:
          dotnet-version: 6.0.x

      - name: Install SonarCloud scanner
        if: steps.cache-sonar-scanner.outputs.cache-hit != 'true'
        run: |
```

```

mkdir -p ../sonar/scanner
chmod a+rwX ../sonar/scanner
dotnet tool update dotnet-sonarscanner --tool-path ../sonar/scanner

- name: Analyze project
  run: |
    ../sonar/scanner/dotnet-sonarscanner begin /k:"IoC-Container" /o:"petro-kolosov"
    /d:sonar.login="${{ secrets.SONAR_TOKEN }}" /d:sonar.host.url="https://sonarcloud.io"
    dotnet build -c Release
    ../sonar/scanner/dotnet-sonarscanner end /d:sonar.login="${{ secrets.SONAR_TOKEN }}"
  }}"

env:
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
  SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}

```

After the push to the develop branch the action is triggered and succeeded

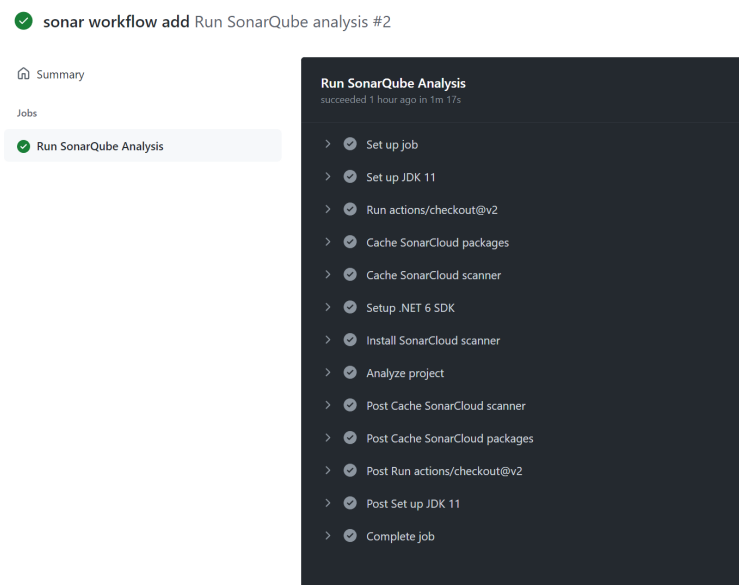


Figure 10. Action is triggered and succeeded.

So that pull request is analyzed and reported to the SonarCloud web application

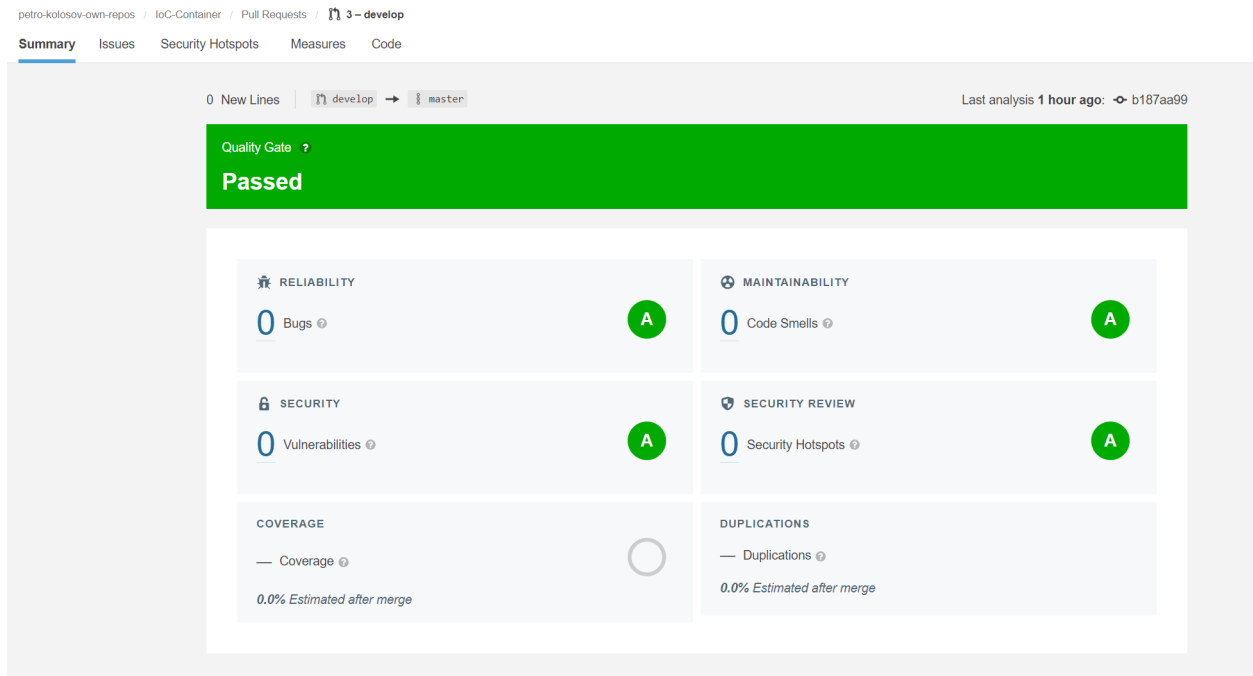


Figure 11. Pull request analysis.

After merge workflow runs again in order to analyze the main branch `master` so that we got a report as follows

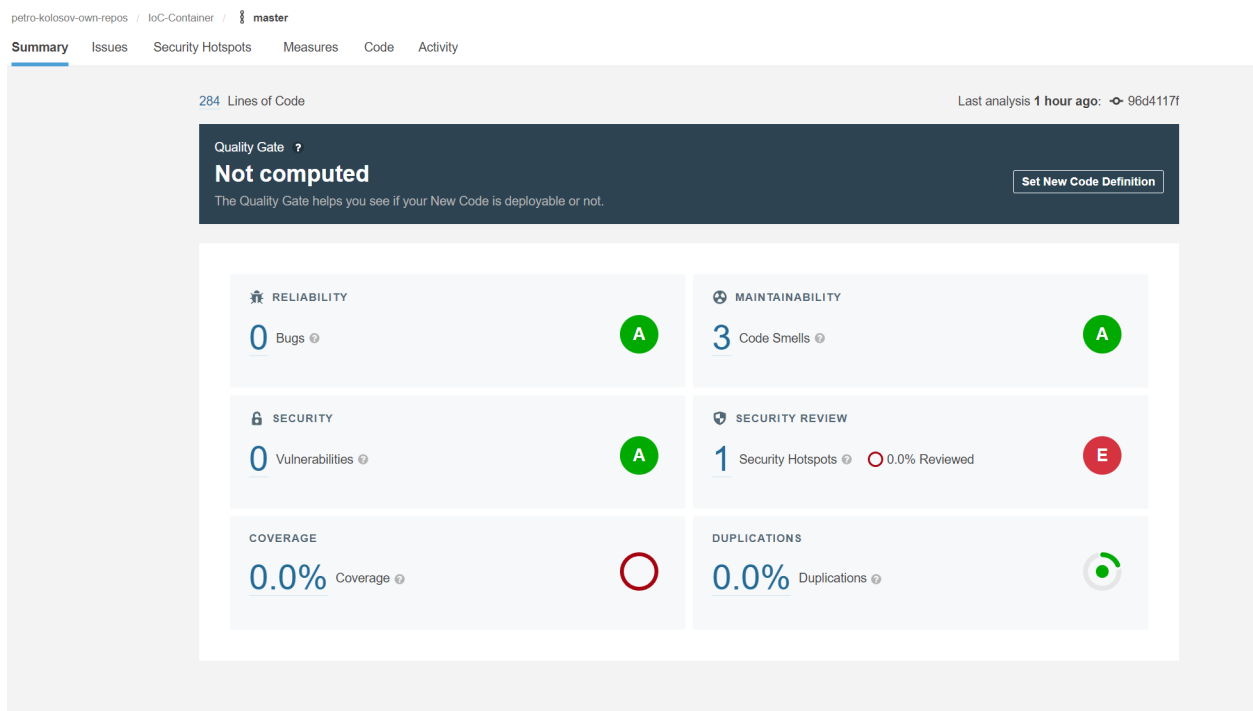


Figure 12. Main branch analysis.

Therefore, we have successfully established an integration between SonarCloud static code analyzer and GitHub repository.

- [Workflow source](#)
- [SonarCloud dashboard](#)

Email address: kolosovp94@gmail.com

URL: <https://kolosovpetro.github.io>