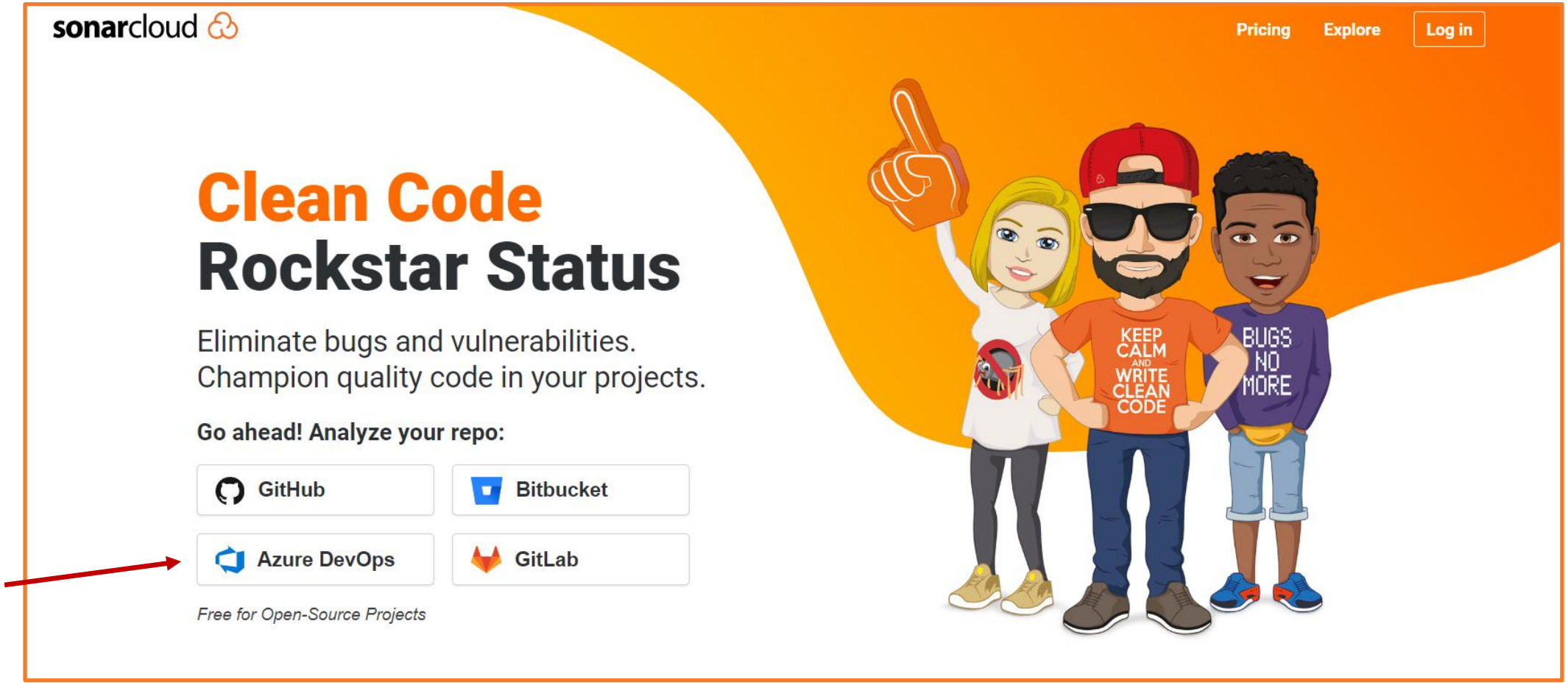
This is **John Kear's** ppt. Connecting Azure DevOps to Sonar Cloud with Code Coverage using Coverlet

Pre-requisite:
1) An Azure DevOps account
2) A GitHub account with repository

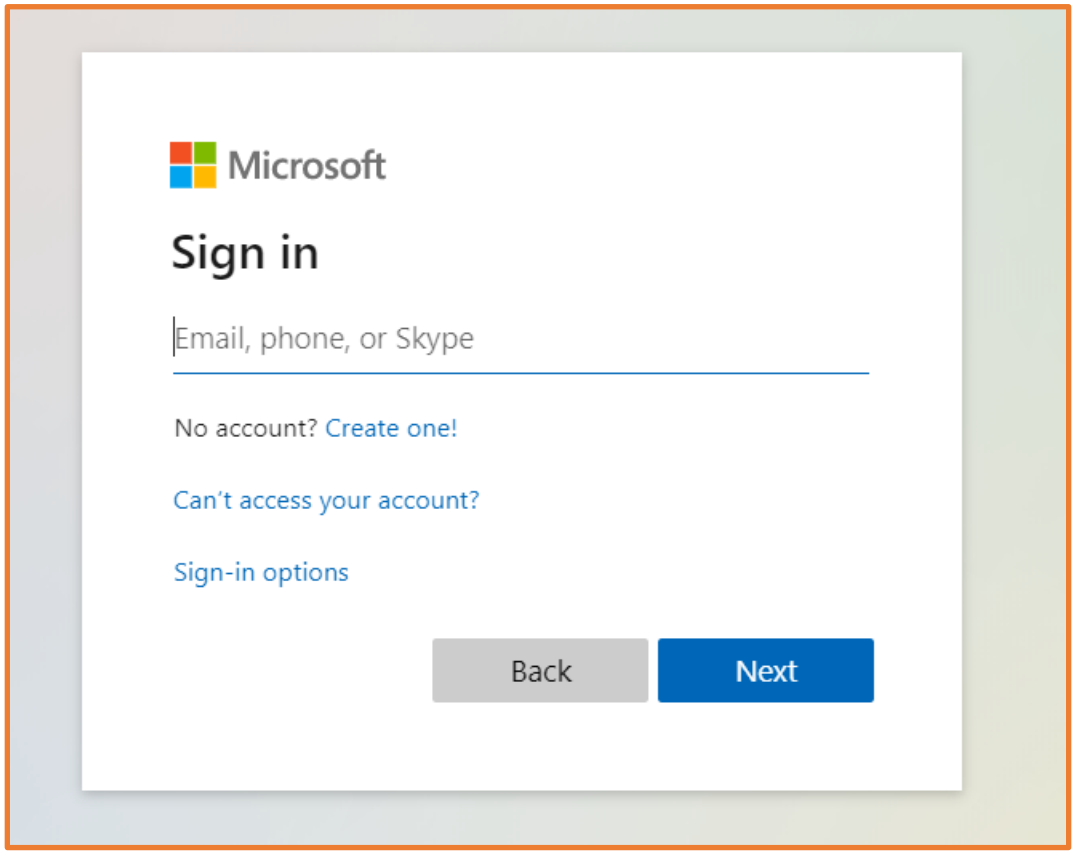
Darius Vallejo example

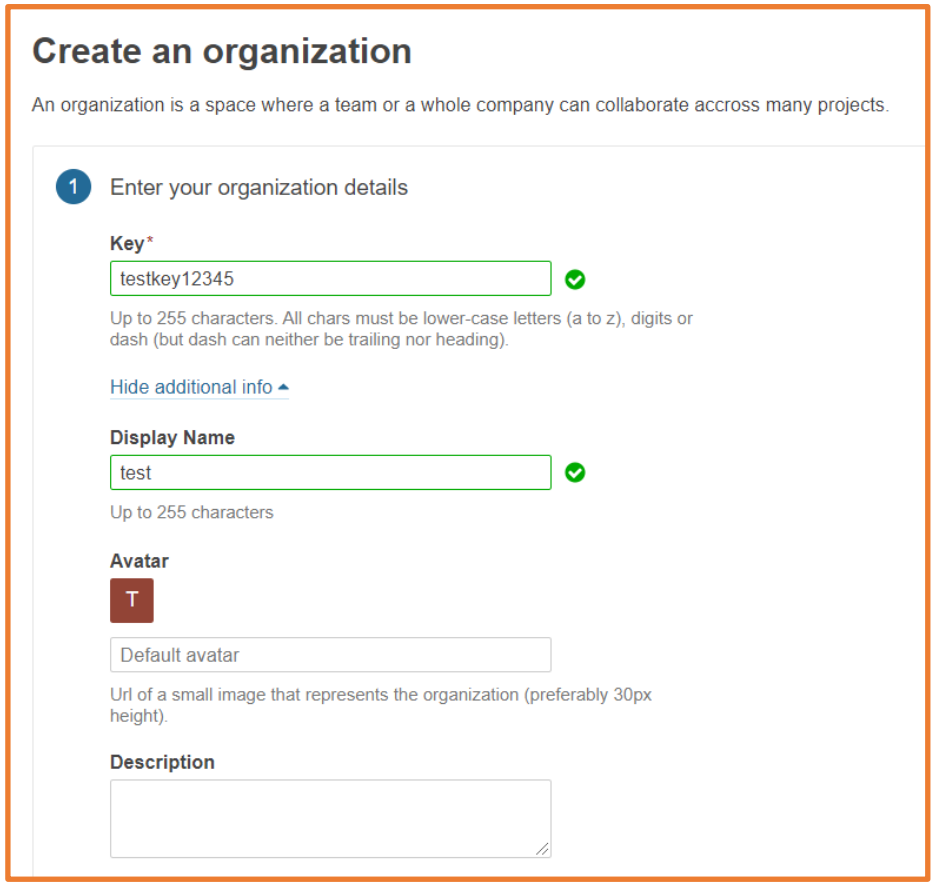
- Darius Vallejo based his integration on the below tutorial. https://azuredevopslabs.com/labs/vstsextend/sonarcloud//

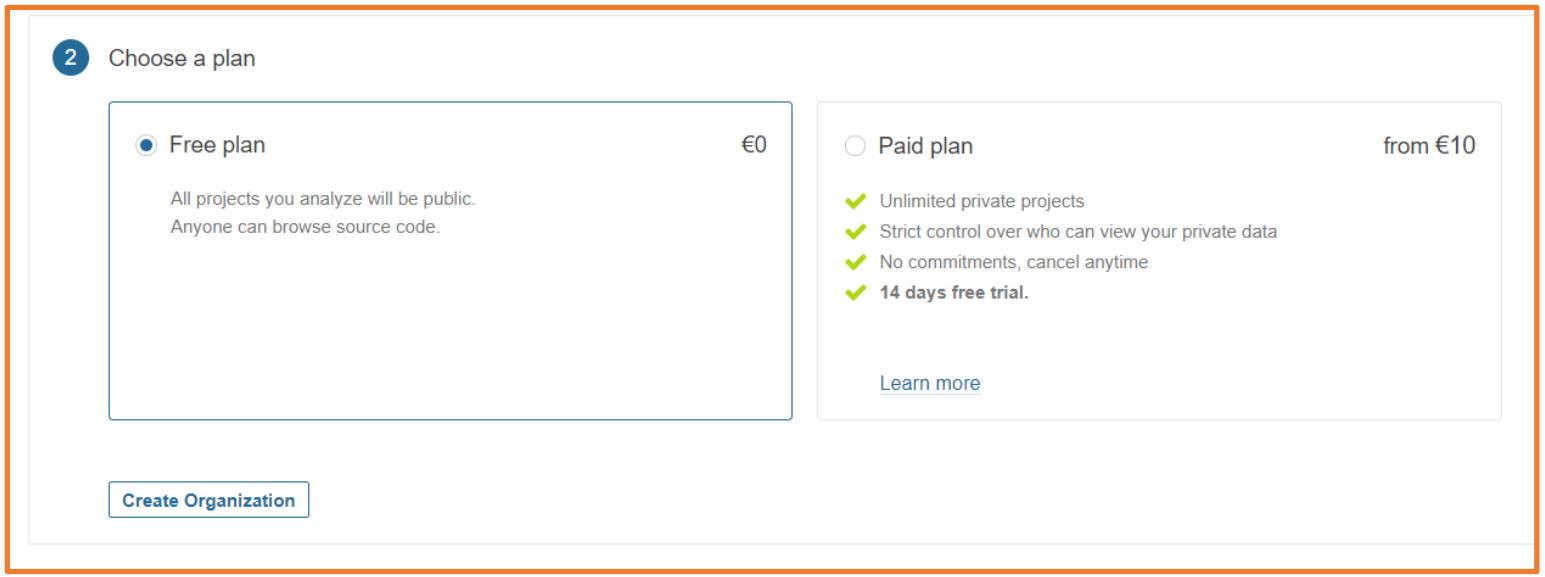
Sonarcloud.io
Get started using Azure DevOps



Sign in with your Microsoft account

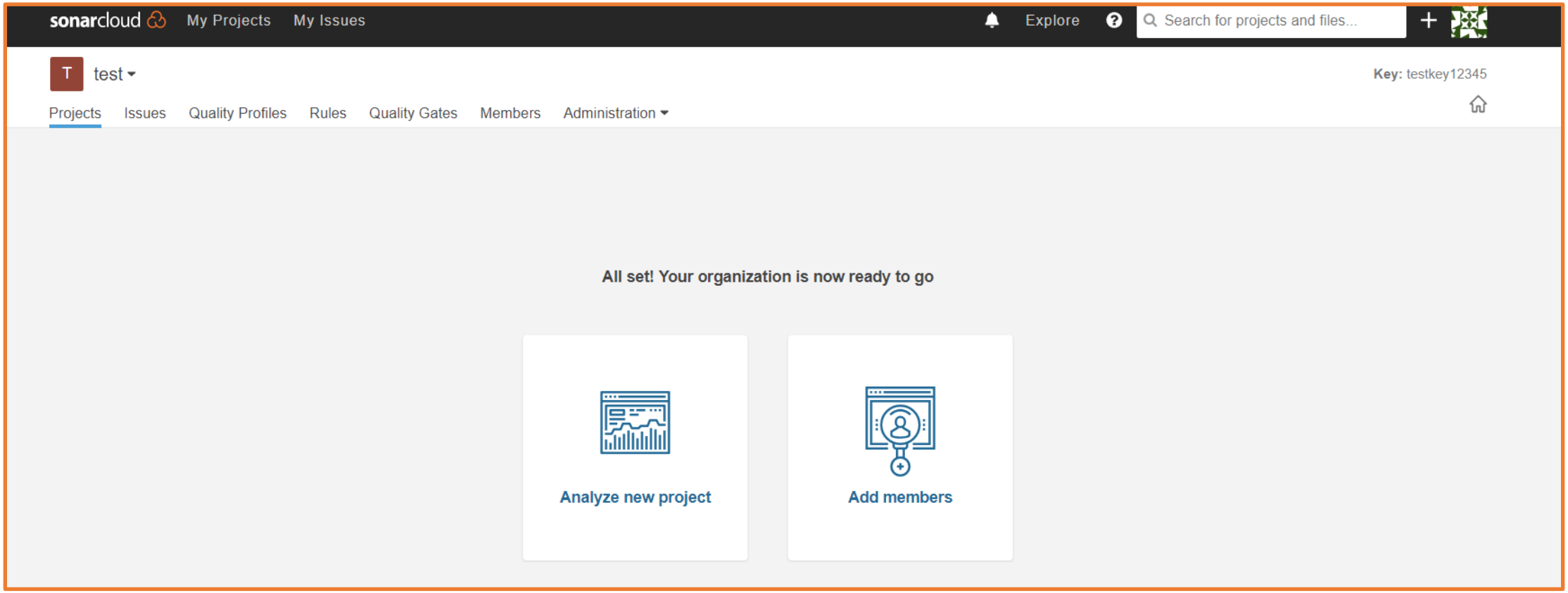






- Create an organization
- Enter a key for your organization
- Click “Continue”
- Select “Free plan”
- Click “Create Organization”
- Click on “Create manually” when you see it after making the organization.
- DO NOT fill in the next part they give you.
- Go straight to generating the token in the next slide.

Analyze a new project



Analyze project setup

Analyze projects - Set up manually

Organization*

test testkey12345

Create another organization

Project key*

thisIsMyProjKey123

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

Display name*

thisIsMyProjKey123

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

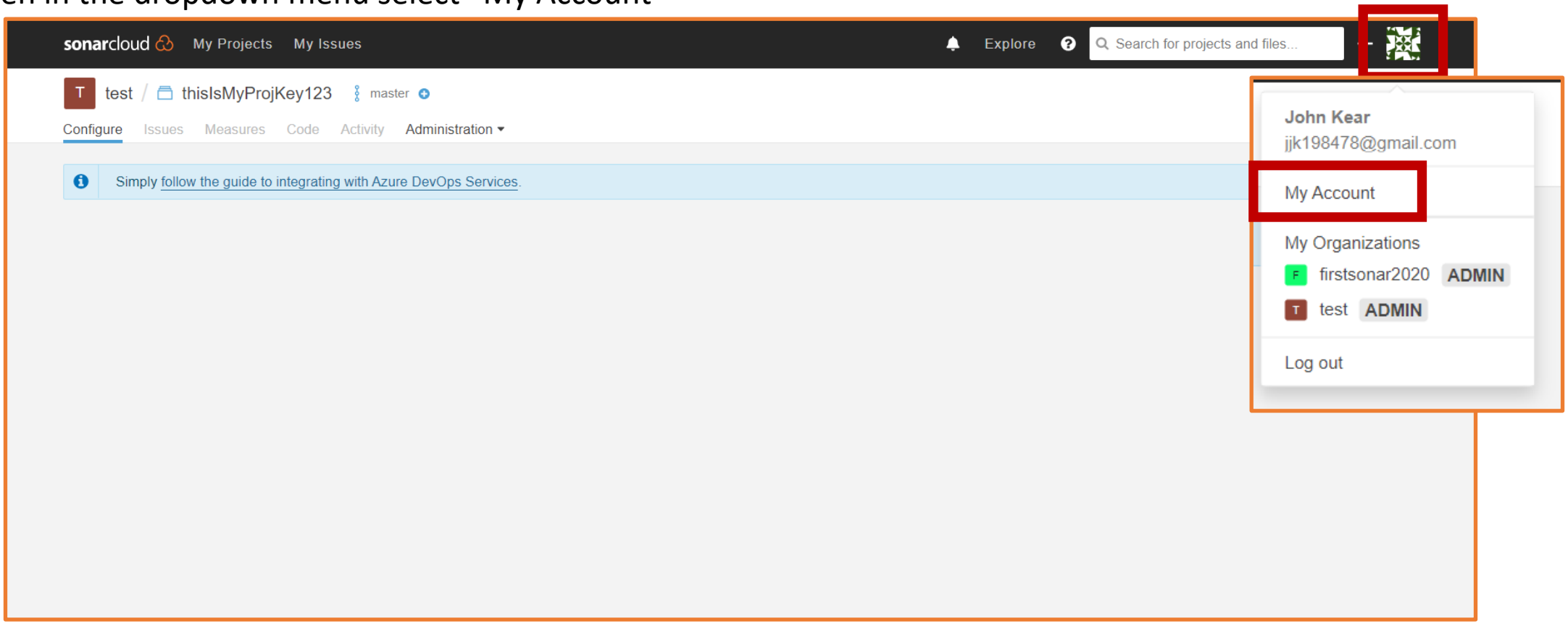
Analyze private projects with our Paid Plan from €10

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

Upgrade Learn more

Generate Token

When configuring Sonar Cloud in Azure DevOps, a token for your Sonar Cloud project will be needed.
-Click on your account image in the top right of the screen
-Then in the dropdown menu select “My Account”

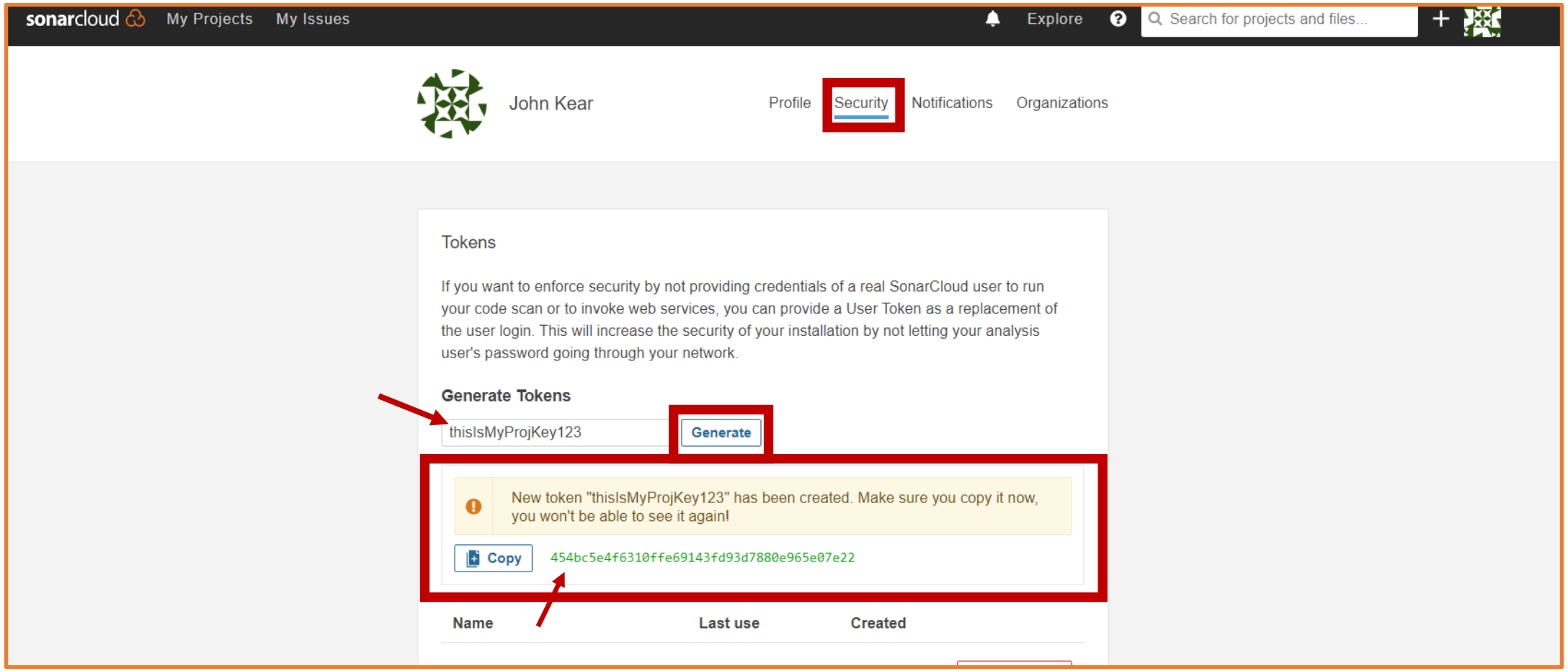


-Select “Security” in the header menu

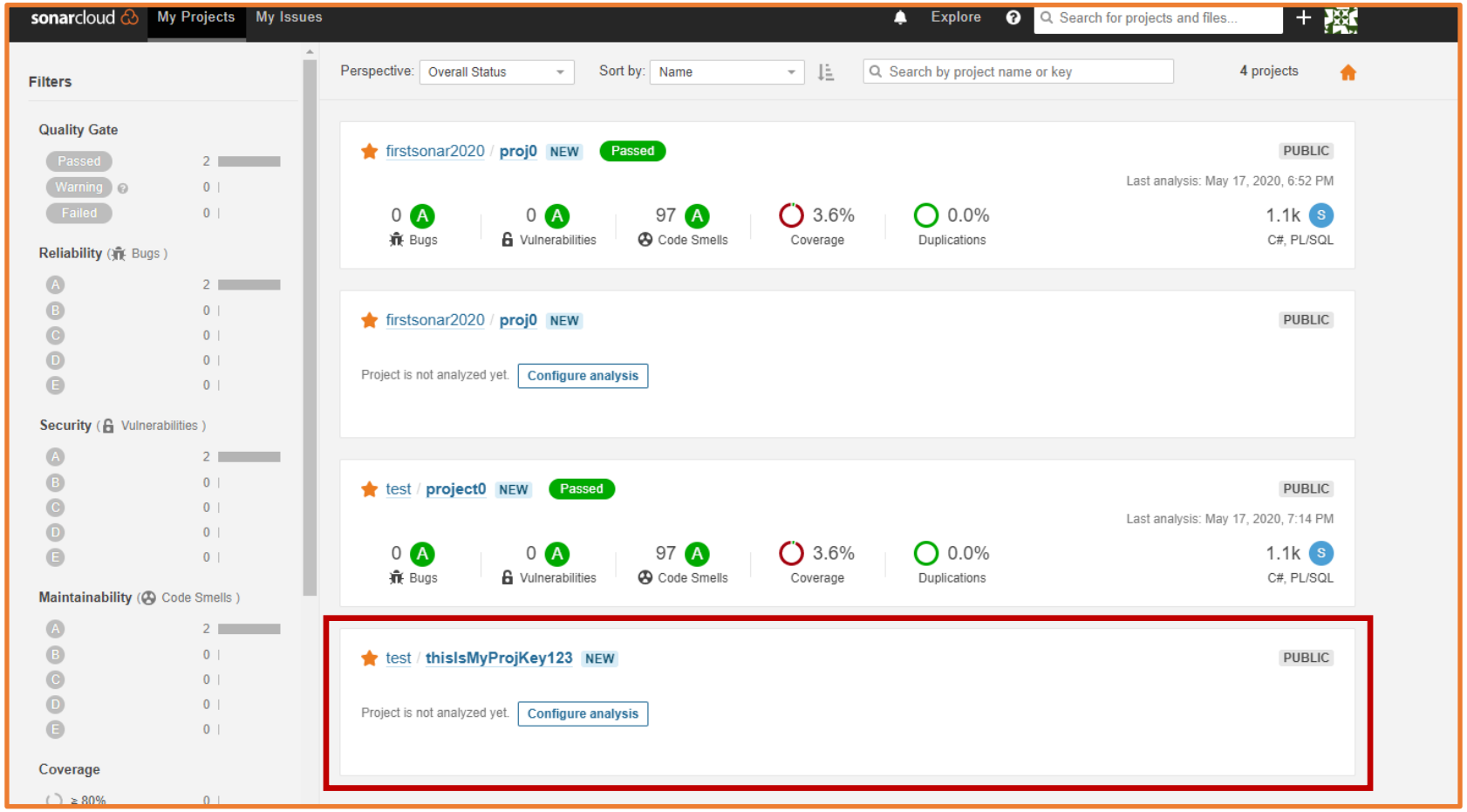
-type the name of the project key we just created in the “Generate Tokens” input box

-click “Generate” this will generate a token for the specified project key

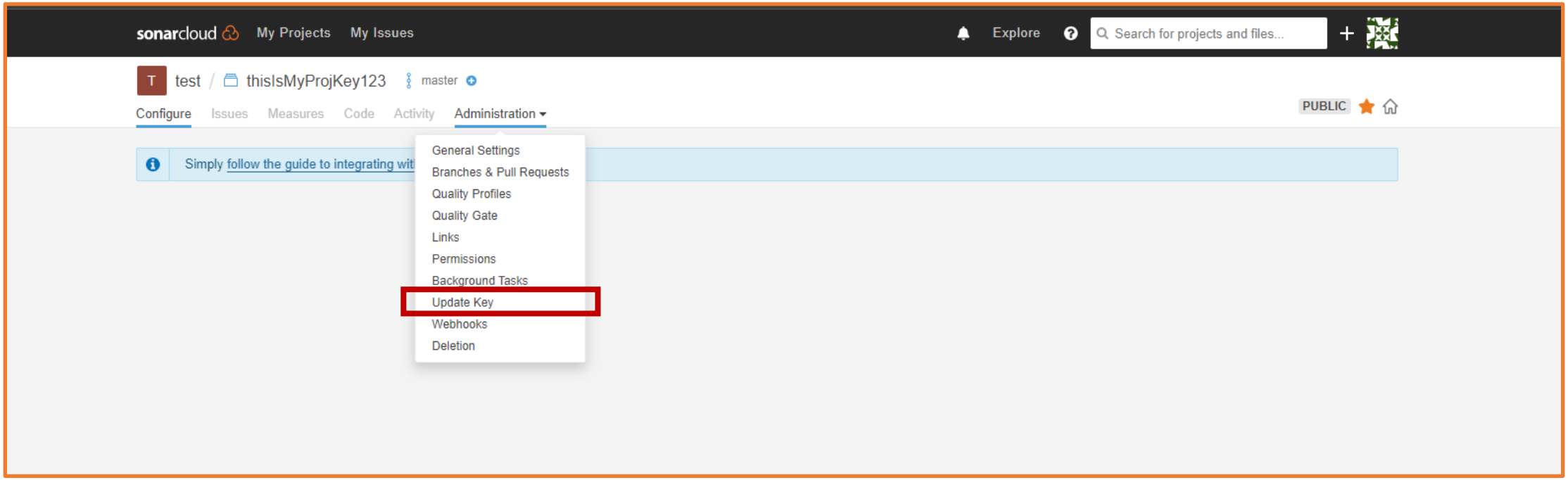
-be sure to copy and save this key somewhere, it will be used in future steps



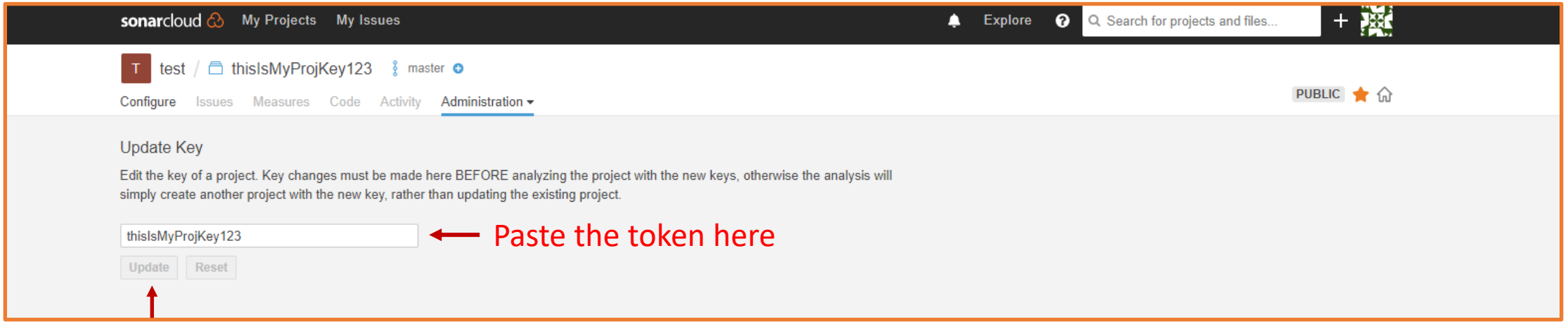
- Now that we have a new token, we
- From your SonarCloud home page, select the project



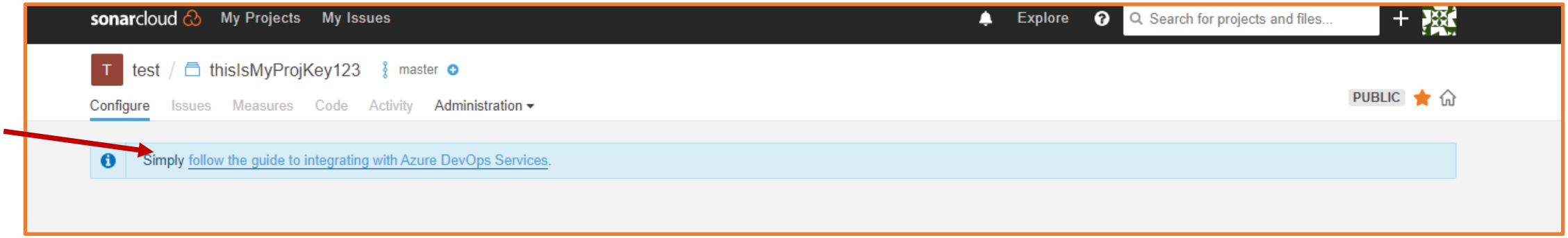
- Select the “Administration” option from the header menu
- In the drop down menu select “Update Key”



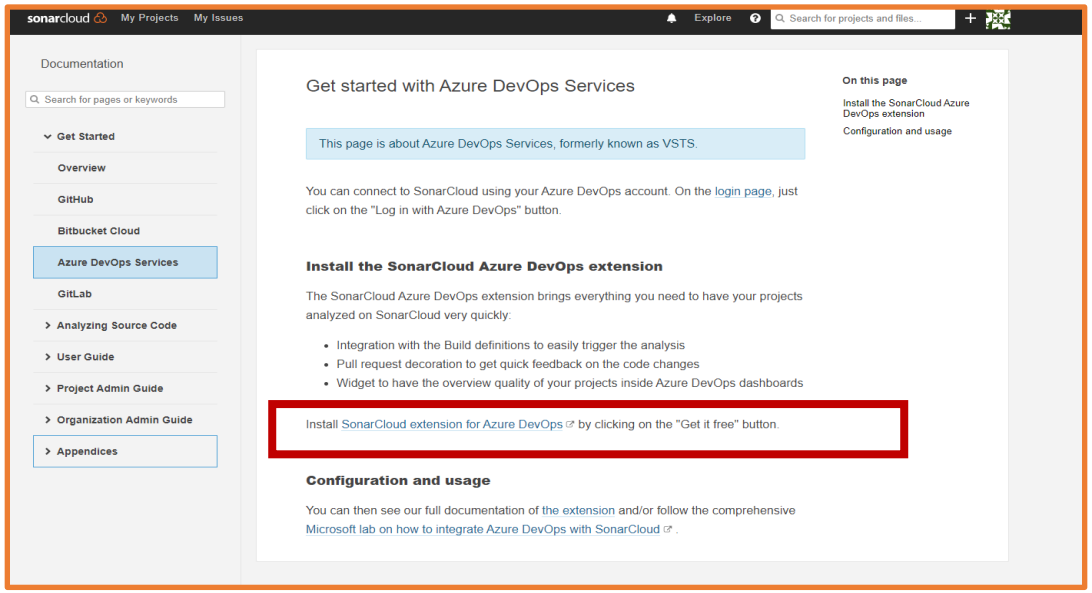
- On the “Update Key” page, paste the token that you generated into the text input box and select update.
- This step is necessary for the SonarCloudAnalyze step in Azure DevOps to properly execute



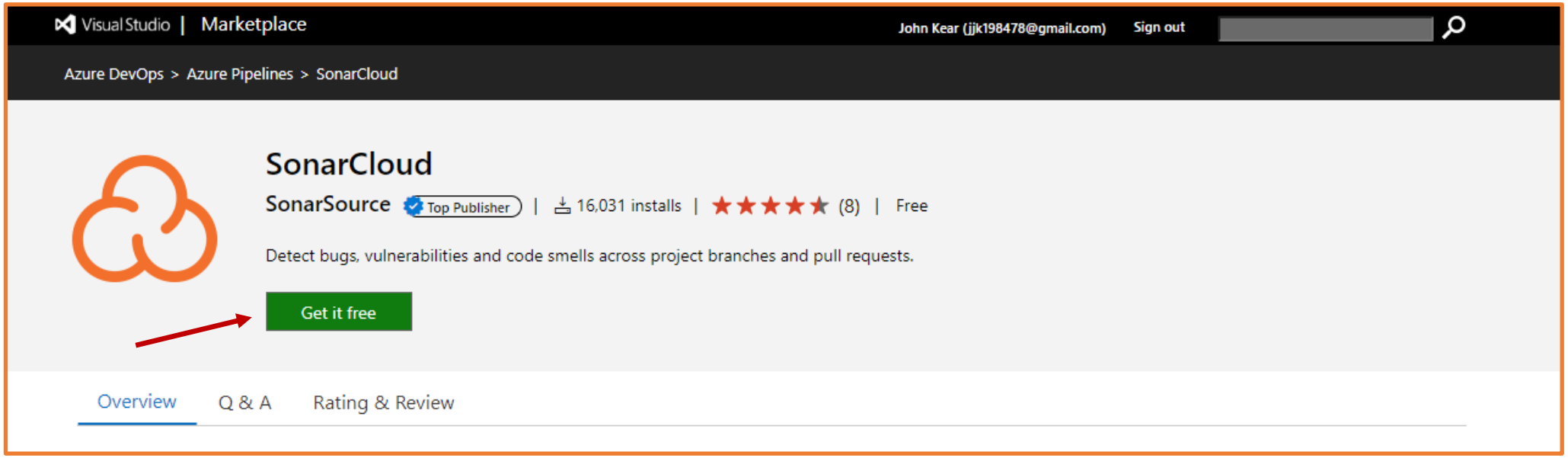
- Now we will need to install the SonarCloud extension to Azure DevOps
- From the Project page in SonarCloud click the “follow the guide to integrating with Azure DevOps Services” link (https://sonarcloud.io/documentation/integrations/vsts/)



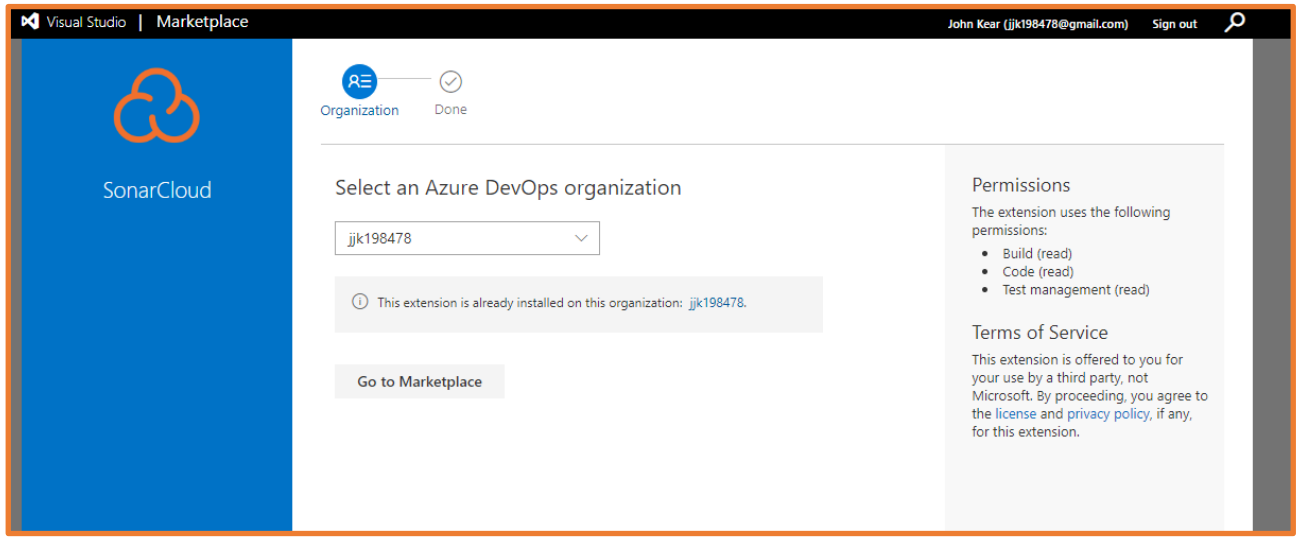
- Click the “Install SonarCloud extension for Azure DevOps” link (https://marketplace.visualstudio.com/items?itemName=SonarSource.sonarcloud)



- Click “Get it free”

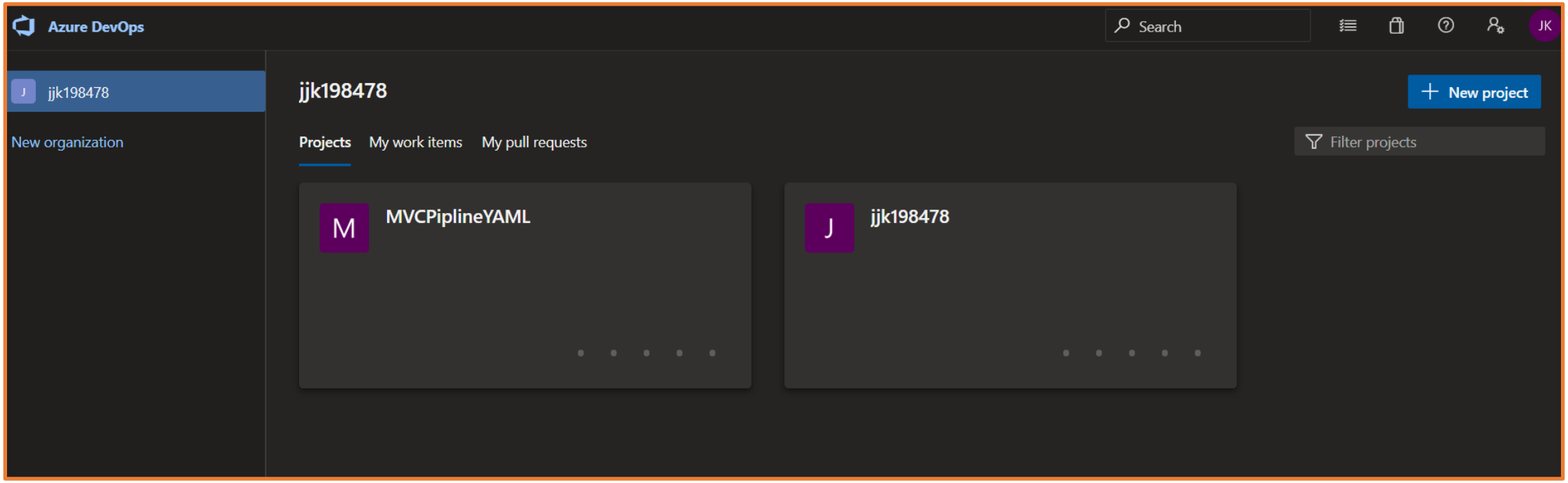


- Select your Azure DevOps organization from the drop down menu, then follow the prompts for installation.
- I already have the extension installed so I am unable to show a walkthrough of this process.

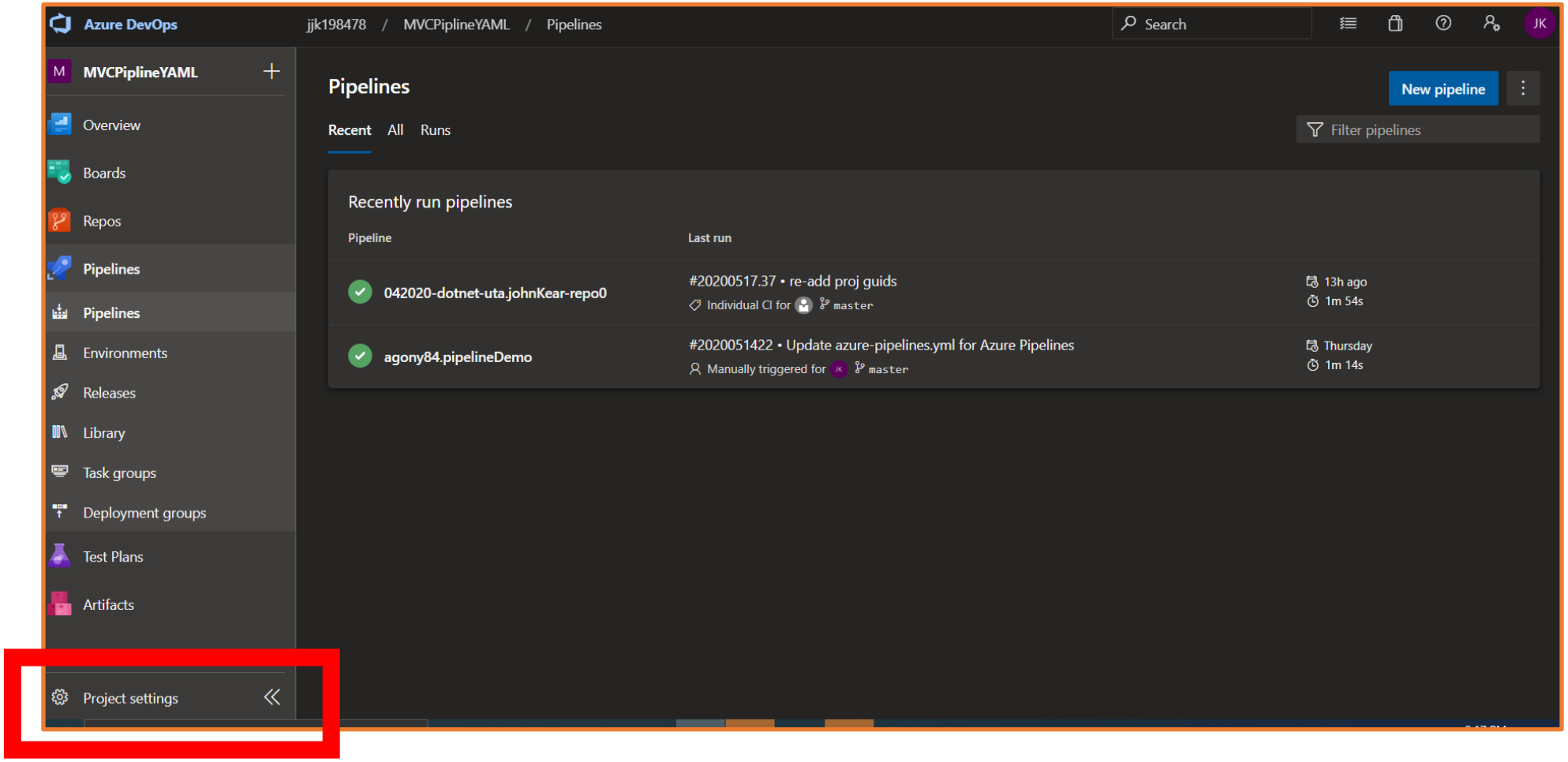


Configuring Sonar Cloud in Azure DevOps

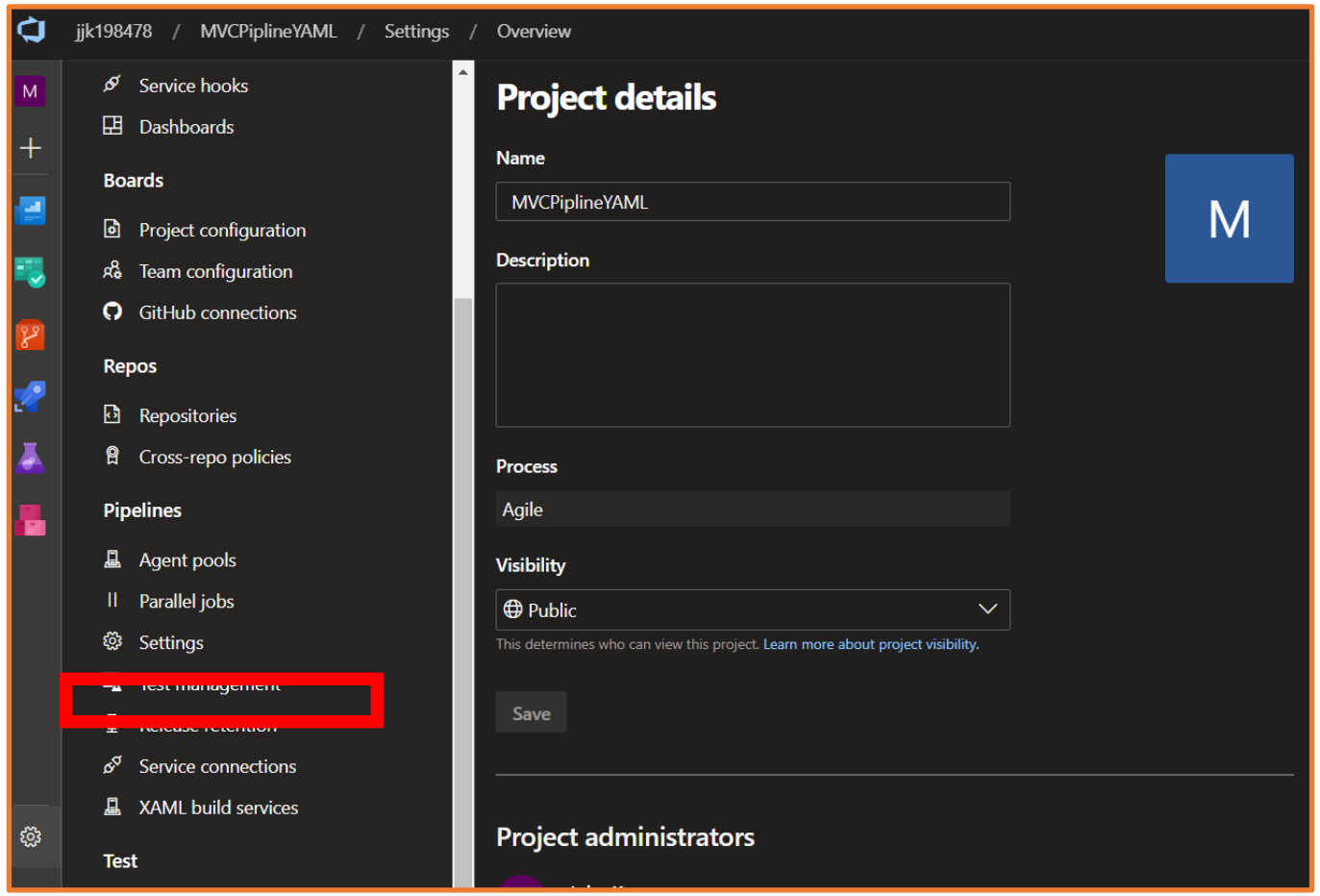
- With the SonarCloud extension installed to Azure DevOps we can now add the service to our Azure Project
- In Azure DevOps, select the project you wish to use with sonar cloud



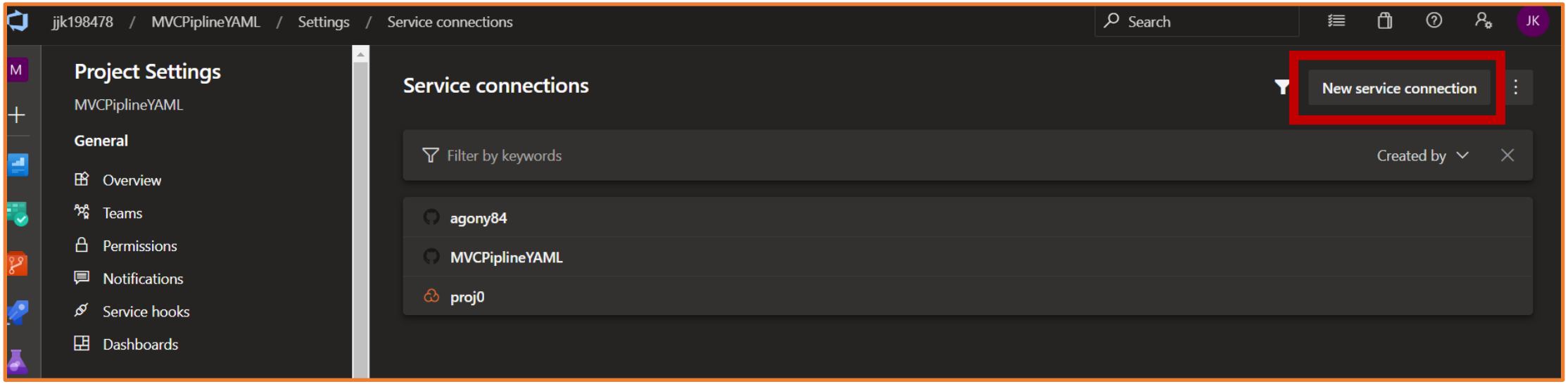
Select Project settings in the bottom left



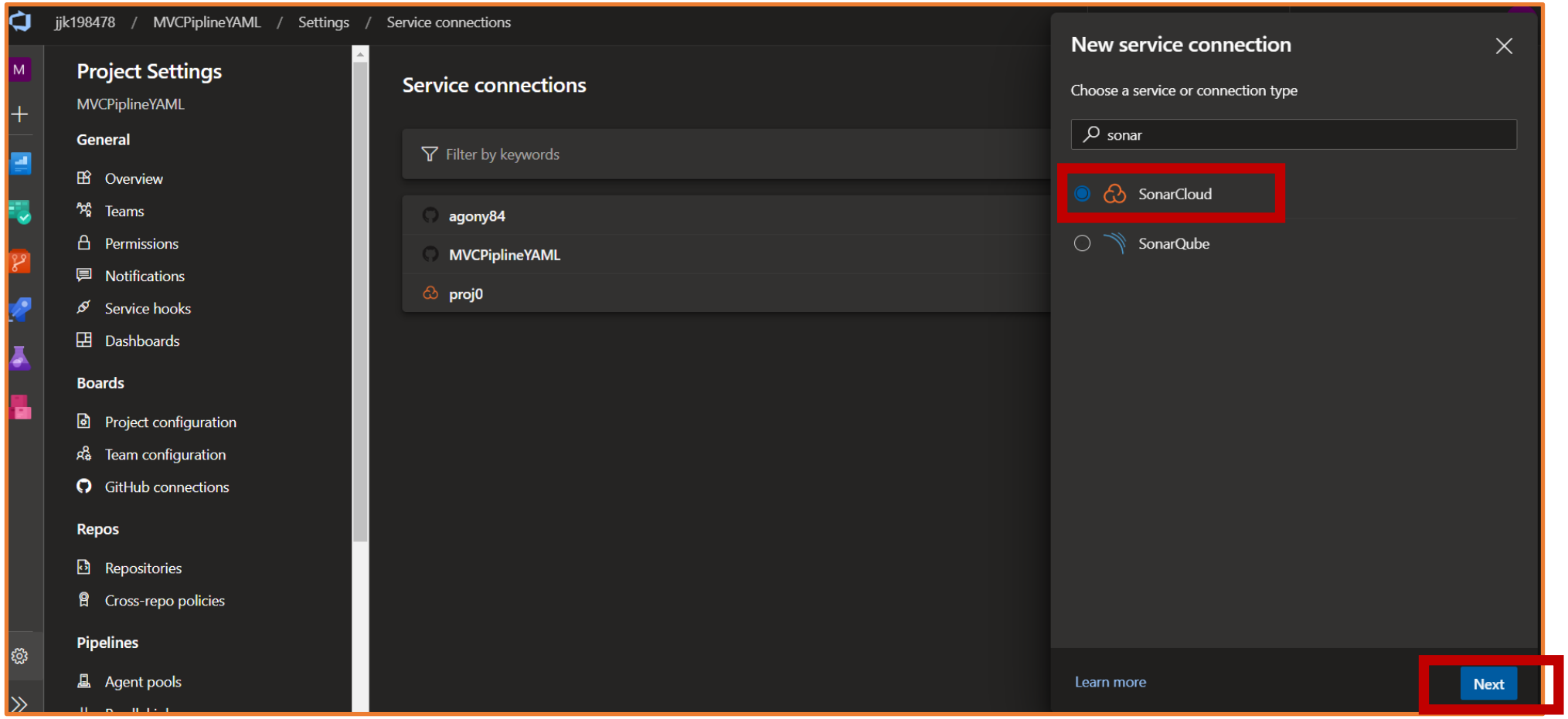
In project settings scroll down on the left and select “Service Connections” under the Pipelines heading



In service connections select “New service connection” at the top right



Search for “SonarCloud”. Select it and click “Next”



-Paste the token from sonar cloud into the “SonarCloud Token” input box

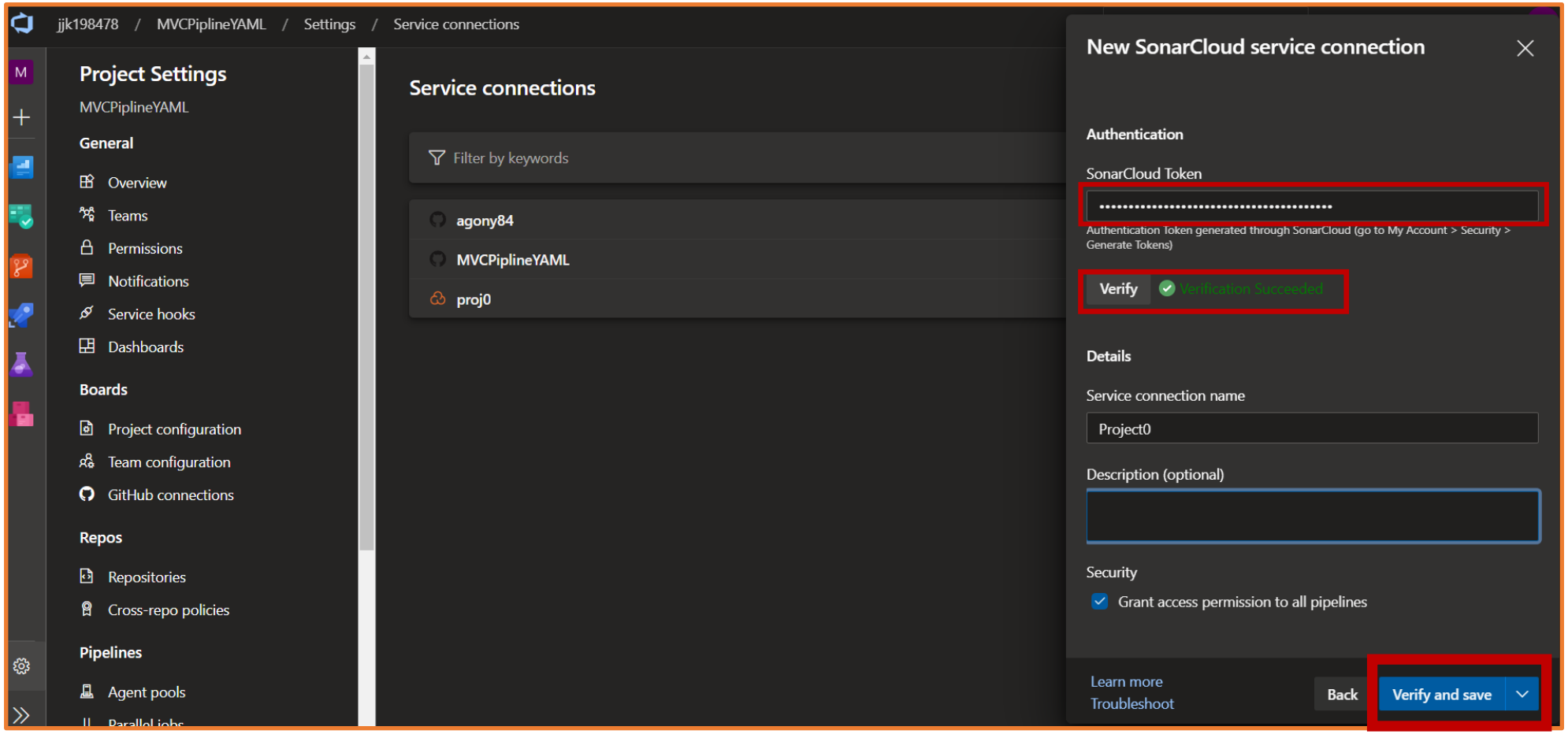
-Click “Verify” to make sure that it actually works. If not, you will need to go SonarCloud and generate a new token

-Enter a name for the “Service connection name”

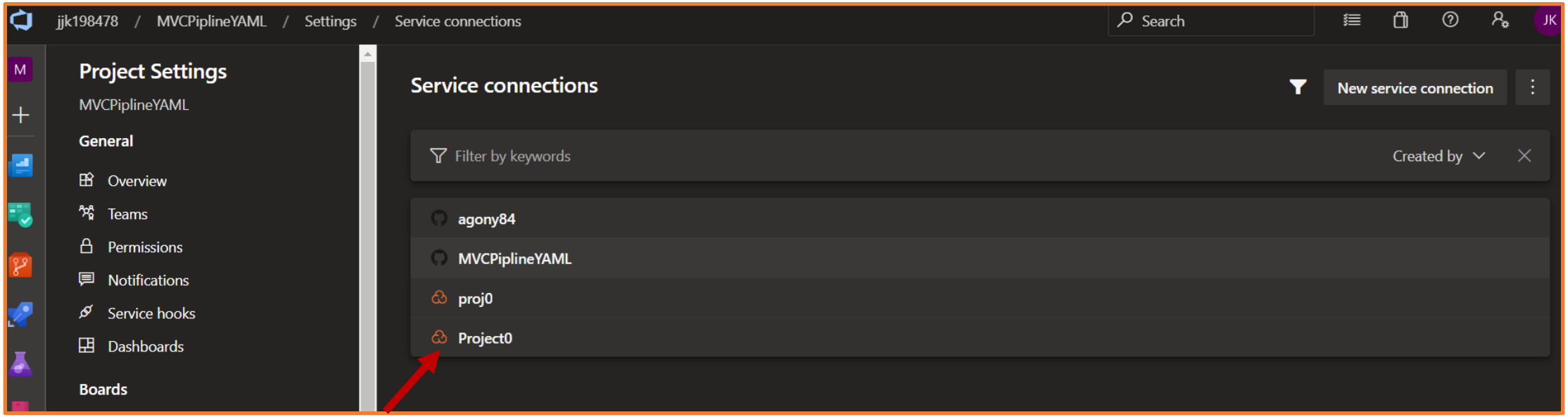
-Enter a description (optional)

-Leave the option “Grant access permission to all pipelines” selected

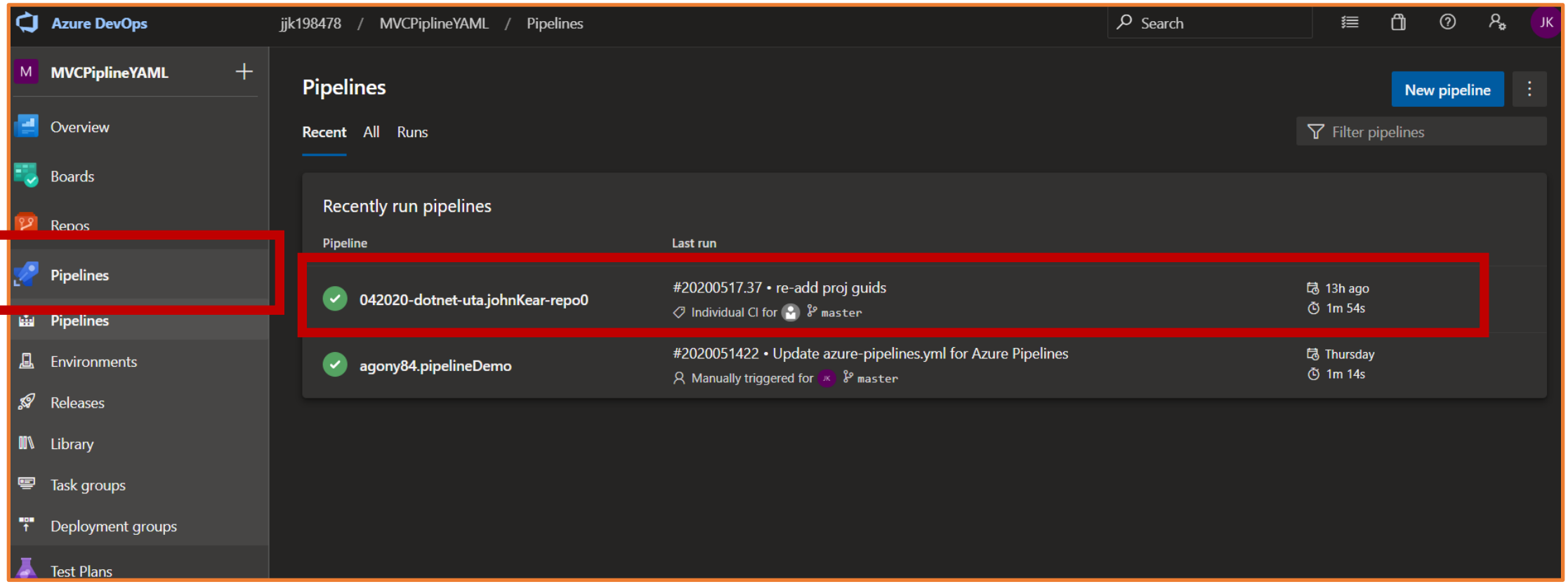
-Click “Verify and save”



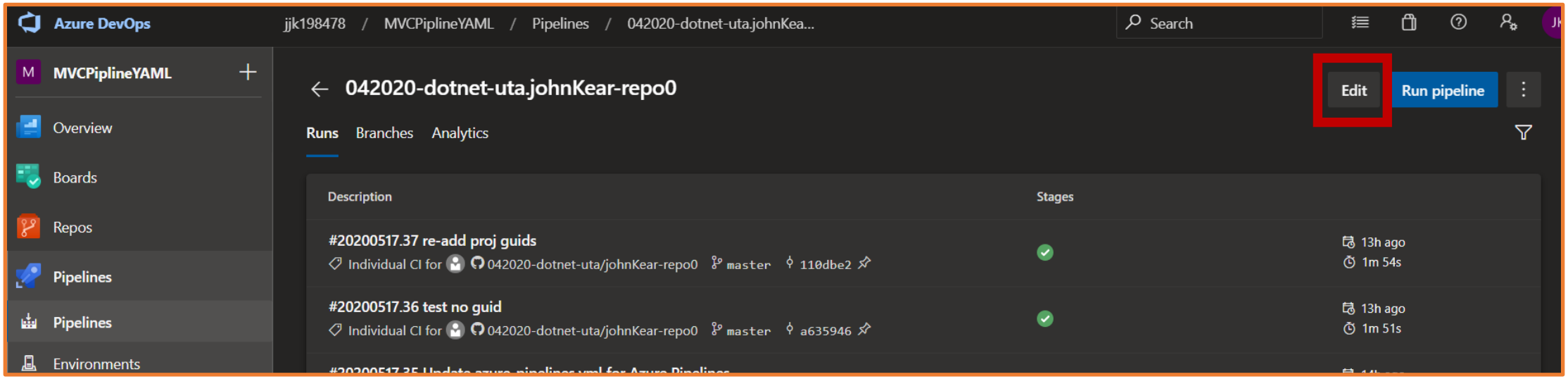
-Once verified you should see the new sonar cloud connection in your “Service connections”
-Now any pipeline in the project will have access to this sonar cloud service connection



-Go back to your Azure DevOps project page and select “Pipelines” in the left menu
-Select the pipeline you wish to add the sonar cloud service to



-Select “Edit” at the top right to edit the pipeline .yml file



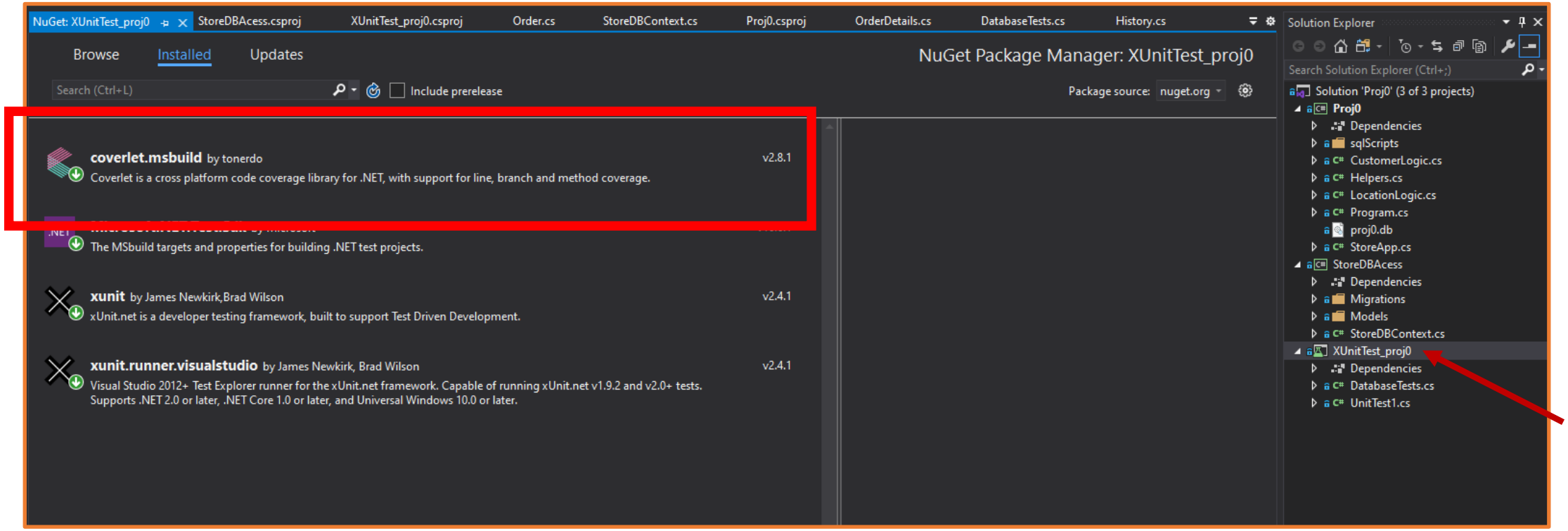
042020-dotnet-uta.johnKear-repo0

master | 042020-dotnet-uta/johnKear-repo0 / azure-pipelines.yml *

```yaml
# Starter pipeline
# Start with a minimal pipeline that you can customize to build and deploy your code.
# Add steps that build, run tests, deploy, and more:
# https://aka.ms/yaml
# Whenever a change is made to the master branch
trigger:
- master

# using ubuntu latest vmImage
pool:
  vmImage: 'ubuntu-latest'

pr: 'none'

stages:
  - stage:
    jobs:
      - job: 'build'
        steps:
          - script: dotnet build 'Proj0/Proj0/Proj0.sln'
          - script: echo 'Project built'
      - job: 'test'
        dependsOn: 'build'
        steps:
          - script: dotnet test 'Proj0/XUnitTest_proj0/XUnitTest_proj0.csproj'
          - script: echo 'tests run'
      - job: 'sonarcloud'
        dependsOn: "test"
      - job: 'publish'
        dependsOn: 'test'
        steps:
```

If you are using the pipeline created for your proj0 your .yml will look similar to this.

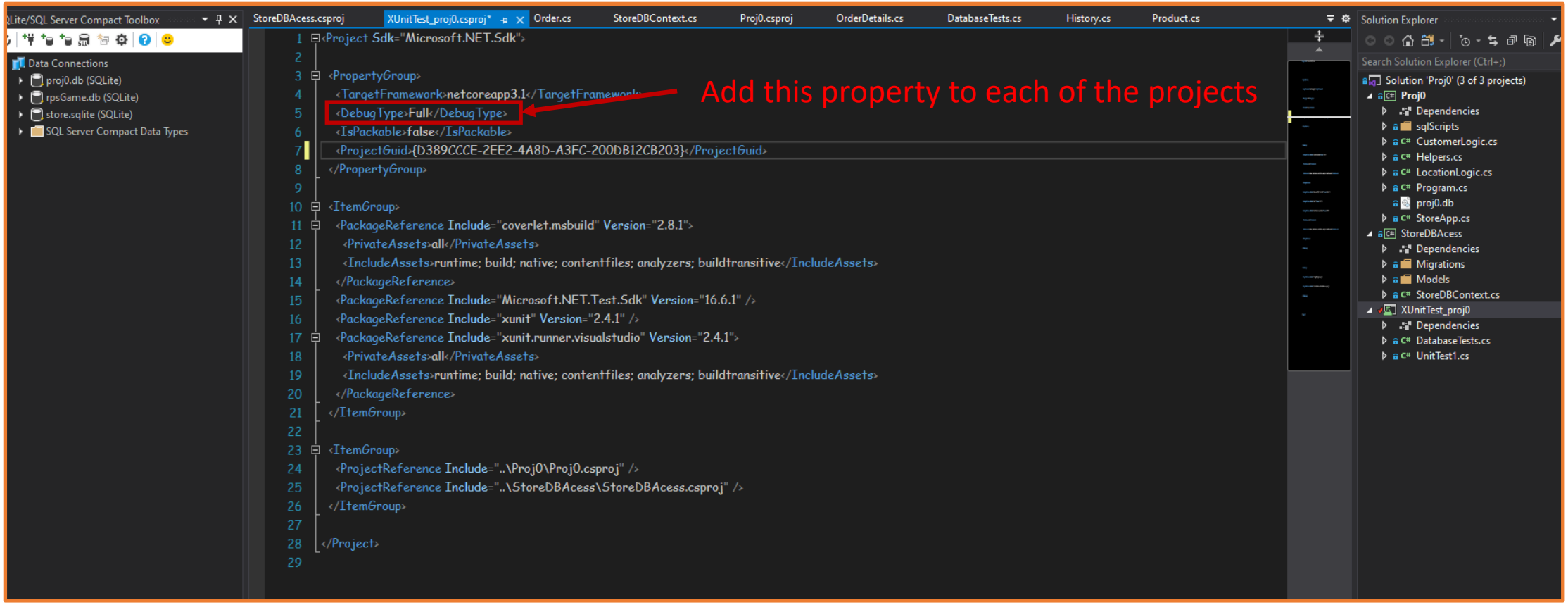
-At this point there are some things that need to be done to the .NET Core project we will be working with. Otherwise Sonar Cloud won't be happy.
-On your computer, open the project you will be analyzing. For me it will be project0
-For the code coverage analysis and report to work, your test project needs to use the "coverlet.msbuild" package. (it may be possible to use something else, but this is what I used)
NOTE: Be sure to ONLY install this package in your test project.
NOTE: The resource I found said to install through the PMC otherwise it would cause issues, however I installed through NuGet Package Manager and it works fine



-Now that coverlet.msbuild is installed we will need to add some properties to each of the projects in the application solution

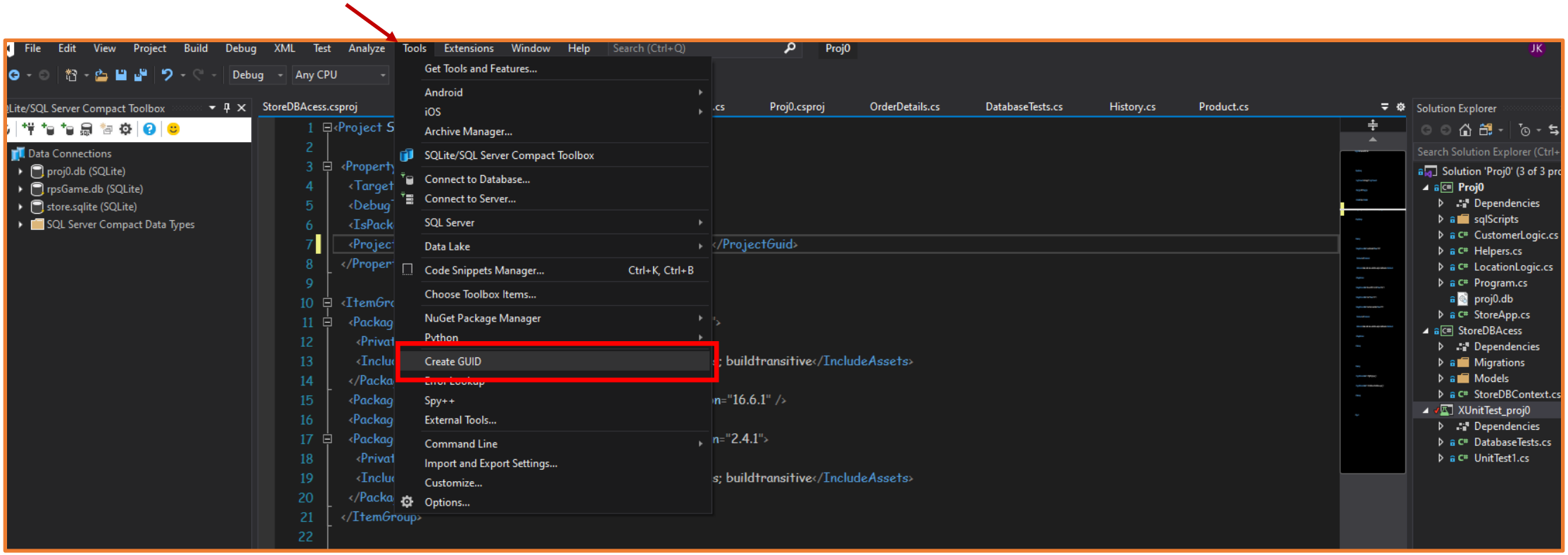
1)Each project will need a <Debug Type>Full</Debug Type> property otherwise the coverlet.msbuild tool may not function properly

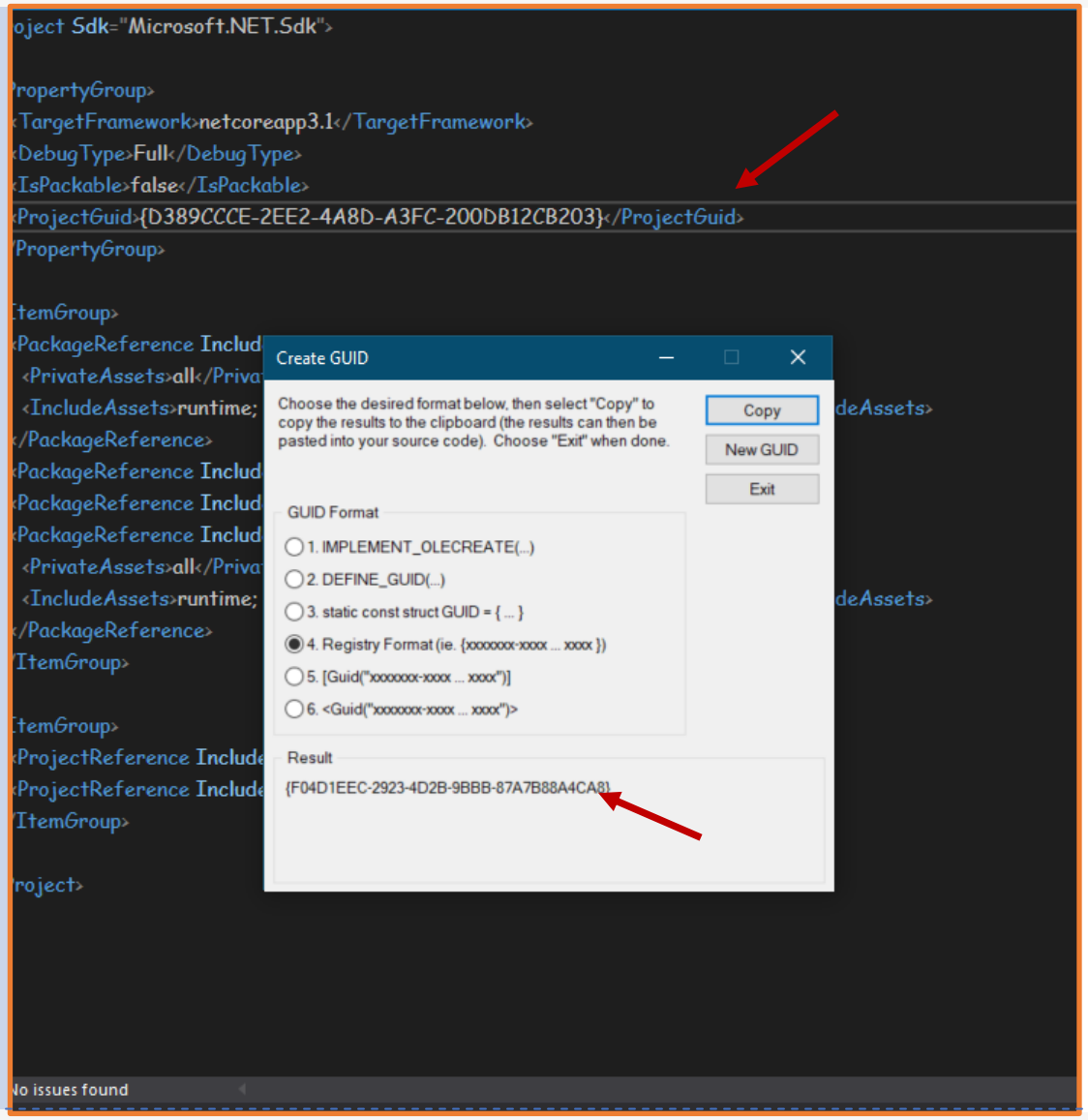
2)Each Project will need a <ProjectGuid>###</ProjectGuid> property so that sonar cloud is able to differentiate between them and analyze them properly. I will show how to generate guids in the following slides



-Each project will need its own unique Guid

-To generate a guid in Visual Studio 2019 Community edition click Tools->Create Guid





-A “Create GUID” window will pop up
-Select option 4 “Registry Format”
-The GUID will be shown in the “Results” section at the bottom of the window.
-Copy this GUID somewhere for safe keeping and select “New GUID” to generate a new GUID for each of your projects
-Close the “Create GUID” window and add the property <ProjectGuid>projectuniqueguid</ProjectGuid> to each of your projects under <PropertyGroup>
-Be sure to save your changes and push the project to your Github repo
-Now that these changes have been made, you are ready to edit your .yml in Azure DevOps

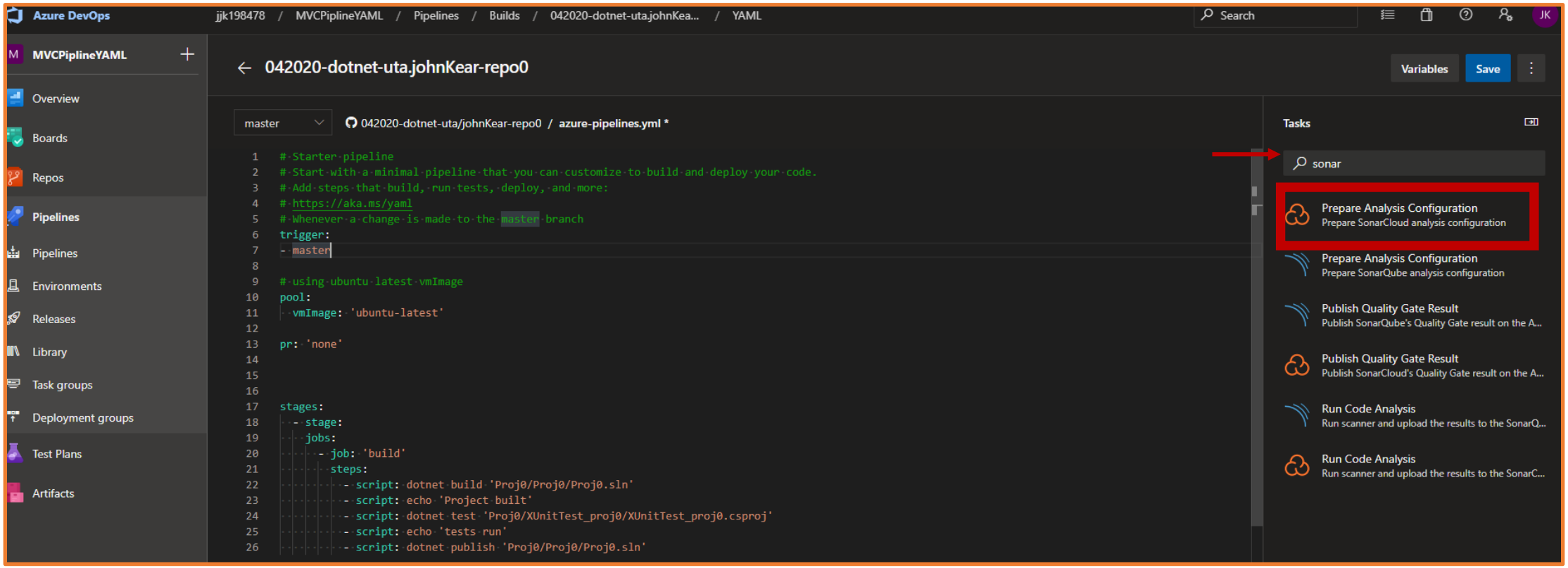
Although we were taught to setup our .yml file in this manner, sonar cloud uses three tools (Analysis Configuration, Analysis, and Publish) that must be run in the same step or they won't work.

```yaml
# Starter pipeline
# Start with a minimal pipeline that you can customize to build and deploy your code.
# Add steps that build, run tests, deploy, and more:
# https://aka.ms/yaml
# Whenever a change is made to the master branch
trigger:
- master

# using ubuntu latest vmImage
pool:
  vmImage: 'ubuntu-latest'

pr: 'none'


stages:
  - stage:
    jobs:
      - job: 'build'
        steps:
          - script: dotnet build 'Proj0/Proj0/Proj0.sln'
          - script: echo 'Project built'
      - job: 'test'
        dependsOn: 'build'
        steps:
          - script: dotnet test 'Proj0/XUnitTest_proj0/XUnitTest_proj0.csproj'
          - script: echo 'tests run'
      - job: 'sonarcloud'
        dependsOn: "test"
      - job: 'publish'
        dependsOn: 'test'
        steps:
          - script: dotnet publish 'Proj0/Proj0/Proj0.sln'
```

-The first thing to do is to reorganize our file and put the build, test and publish all in the same job and step

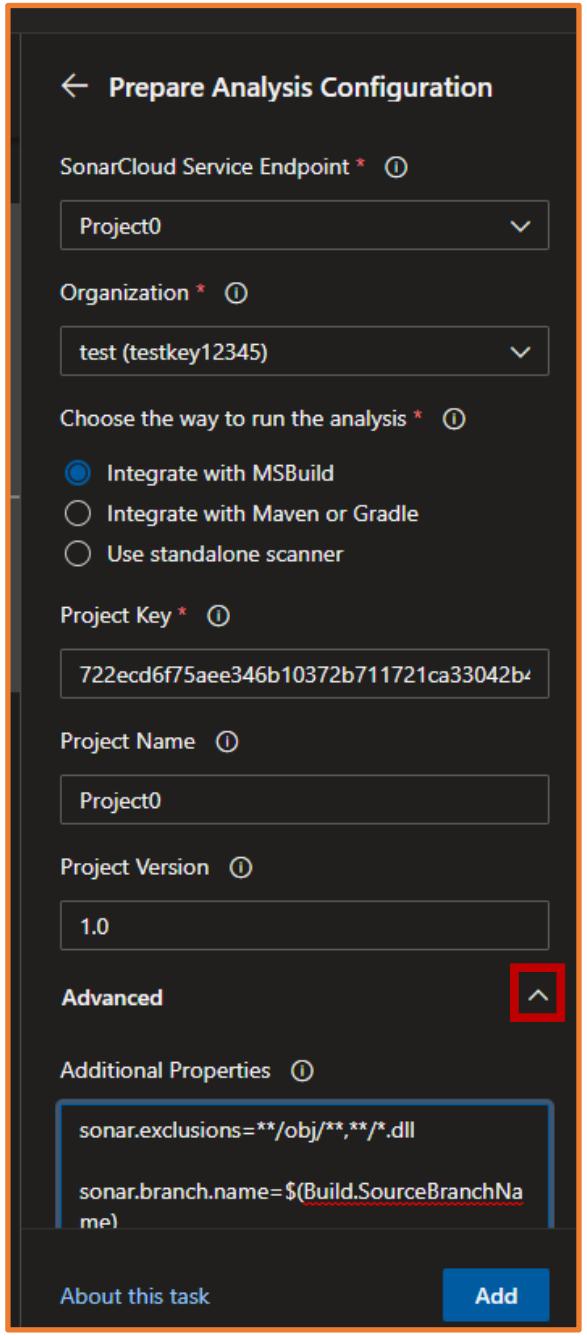
```yaml
# Starter pipeline
# Start with a minimal pipeline that you can customize to build and deploy your code.
# Add steps that build, run tests, deploy, and more:
# https://aka.ms/yaml
# Whenever a change is made to the master branch
trigger:
- master

# using ubuntu latest vmImage
pool:
  vmImage: 'ubuntu-latest'

pr: 'none'



stages:
  - stage:
    jobs:
      - job: 'build'
        steps:
          - script: dotnet build 'Proj0/Proj0/Proj0.sln'
          - script: echo 'Project built'
          - script: dotnet test 'Proj0/XUnitTest_proj0/XUnitTest_proj0.csproj'
          - script: echo 'tests run'
          - script: dotnet publish 'Proj0/Proj0/Proj0.sln'
```

-In the “Tasks” search type “sonar”
-Select “Prepare Analysis Configuration”



- From the dropdown select the SonarCloud Service Endpoint we created for the project
- From the dropdown, select the SonarCloud organization
- We are using the coverlet.msbuild tool for reports so select the “Integrate with MSBuild” option for analysis selected
- For the “Project Key” paste the generated token from sonar cloud
- Enter a project name
- Expand the “Advanced” section
- In the additional properties section add the following:
  sonar.exclusions=**/obj/**,**/*.dll
  sonar.branch.name=$(Build.SourceBranchName)
  sonar.cs.vstest.reportsPaths=$(Agent.TempDirectory)/*.trx
  sonar.cs.opencover.reportsPaths=$(Build.SourcesDirectory)/**/*.xml

- Click “Add”

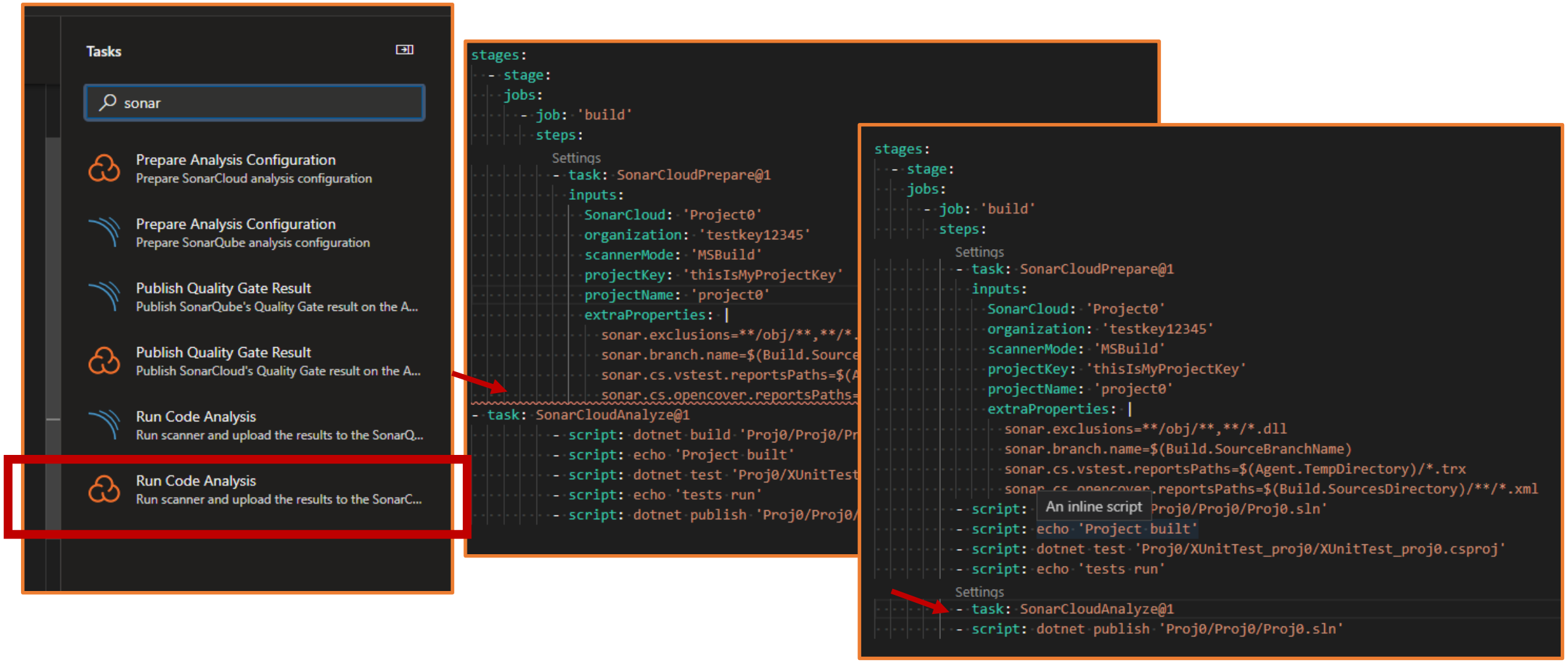


-A resulting task should have been added and look similar to this
-This task needs to be placed before the “Build” command as shown in the screenshot on the right.
Pay attention to indentation!

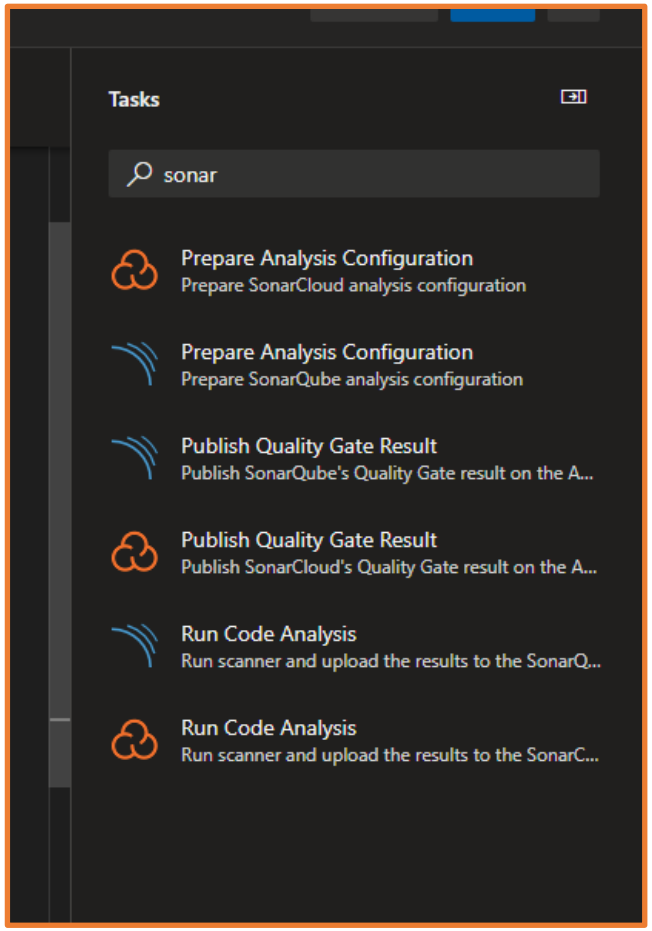
```yaml
        steps:
            - script: dotnet build 'Proj0/Proj0/Proj0.sln'
- task: SonarCloudPrepare@1
  inputs:
    SonarCloud: 'Project0'
    organization: 'testkey12345'
    scannerMode: 'MSBuild'
    projectKey: 'thisIsMyProjectKey'
    projectName: 'project0'
    extraProperties: |
      sonar.exclusions=**/obj/**,**/*.dll
      sonar.branch.name=$(Build.SourceBranchName)
      sonar.cs.vstest.reportsPaths=$(Agent.TempDirectory)/*.trx
      sonar.cs.opencover.reportsPaths=$(Build.SourcesDirectory)/**/*.xml
            - script: echo 'Project built'
            - script: dotnet test 'Proj0/XUnitTest_proj0/XUnitTest_proj0.csproj'
            - script: echo 'tests run'
            - script: dotnet publish 'Proj0/Proj0/Proj0.sln'
```

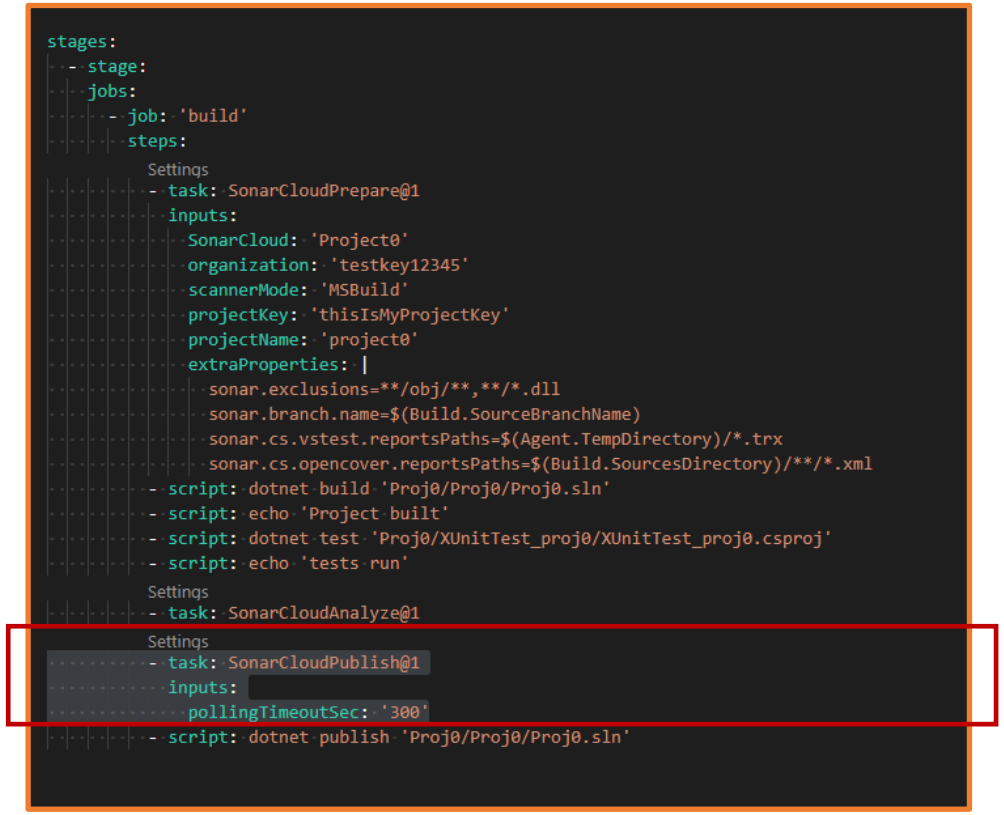
```yaml
stages:
  - stage:
    jobs:
      - job: 'build'
        steps:
          Settings
          - task: SonarCloudPrepare@1
            inputs:
              SonarCloud: 'Project0'
              organization: 'testkey12345'
              scannerMode: 'MSBuild'
              projectKey: 'thisIsMyProjectKey'
              projectName: 'project0'
              extraProperties: |
                sonar.exclusions=**/obj/**,**/*.dll
                sonar.branch.name=$(Build.SourceBranchName)
                sonar.cs.vstest.reportsPaths=$(Agent.TempDirectory)/*.trx
                sonar.cs.opencover.reportsPaths=$(Build.SourcesDirectory)/**/*.xml
          - script: dotnet build 'Proj0/Proj0/Proj0.sln'
          - script: echo 'Project built'
          - script: dotnet test 'Proj0/XUnitTest_proj0/XUnitTest_proj0.csproj'
          - script: echo 'tests run'
          - script: dotnet publish 'Proj0/Proj0/Proj0.sln'
```

-In the “Tasks” search enter sonar again
-Select “Run Code Analysis” make sure to select the option with the sonar cloud logo
-A tasks will be auto generated as shown
-Move the task to AFTER the test script



-In the “Tasks” search enter sonar again
-Select “Publish Quality Gate Result” and click “Add”. make sure to select the option with the sonar cloud logo
-A tasks will be auto generated as shown
-Move the task somewhere after the SonarCloudAnalyze task



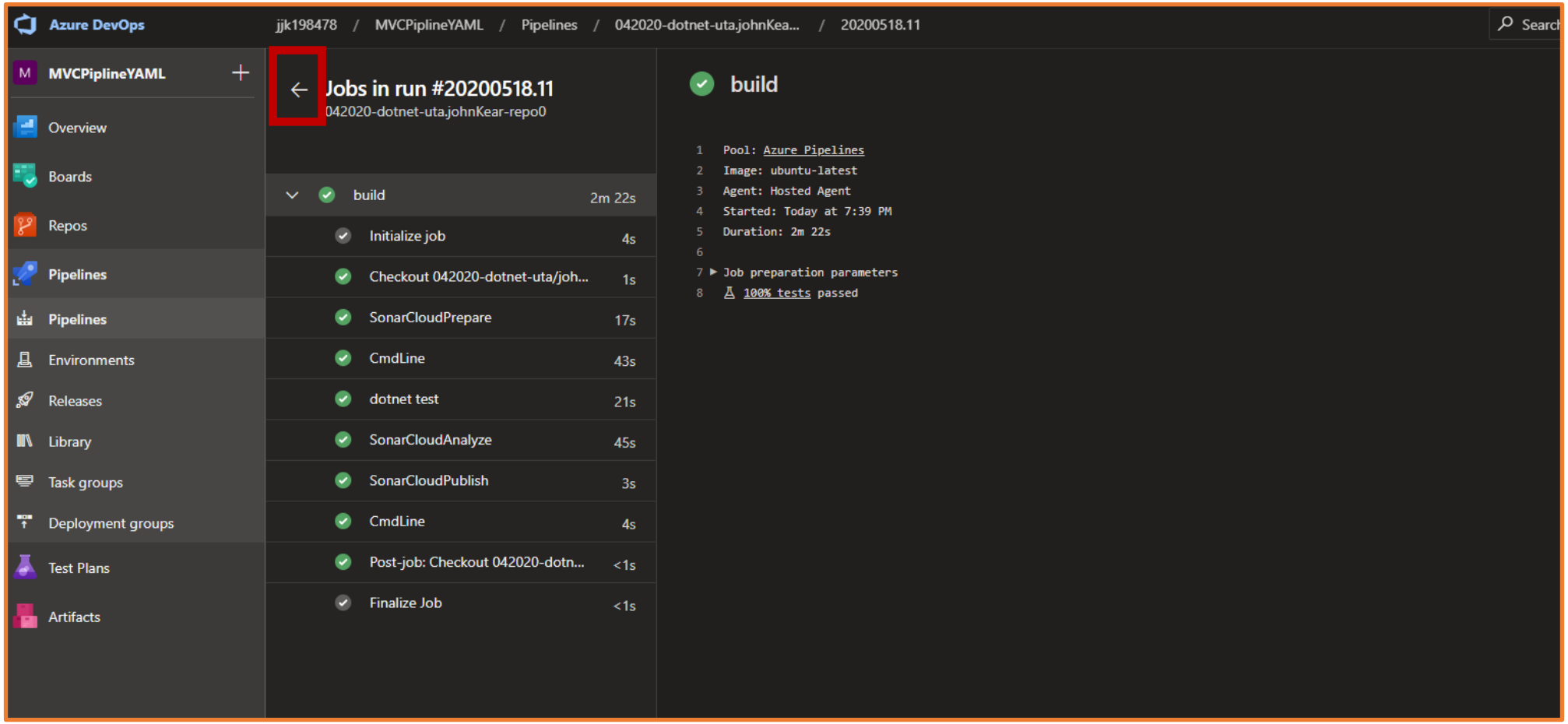


-In order for SonarCloudAnalyze and SonarCloudPublish to analyze and publish the Code Coverage, it must first be generated. Generating the Code Coverage analysis file is done during testing.
-To generate the Code Coverage analysis file, we must alter the test process.
-Instead of using the script dotnet test, we will replace it with a task as shown
-The task is a DotNetCoreCLI@2
- The display name is merely for Azure DevOps when running the task. We will see this later.
-The command is ‘test’
-In arguments we set the configuration parameters. These include setting Collect Coverage to true, setting the CoverletOutputFormat to opencover (very important!!!), the Coverletoutput path, and turn on the logger for test results.
-In “projects” we specify the test proj
-”nobuild” can be set to true

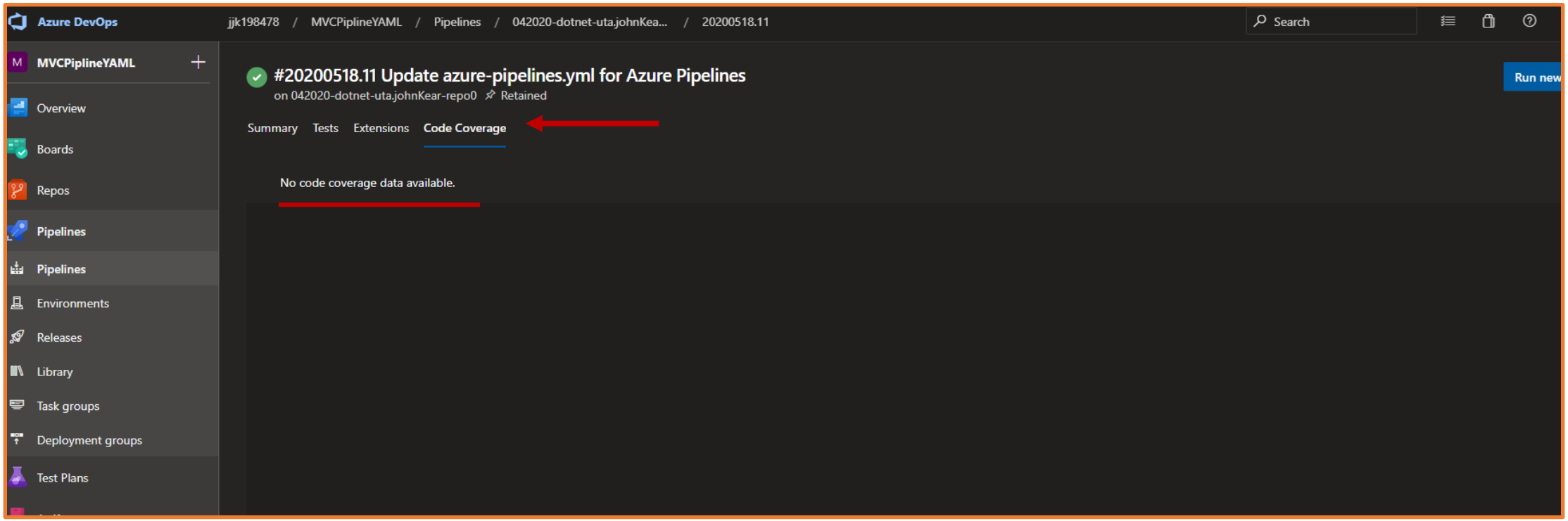
master 042020-dotnet-uta/johnKear-repo0 / azure-pipelines.yml *

```yaml
- job: 'build'
  steps:
    Settings
    - task: SonarCloudPrepare@1
      inputs:
        SonarCloud: 'Project0'
        organization: 'testkey12345'
        Project Key : 'MSBuild'
        projectKey: 'thisIsMyProjectKey'
        projectName: 'project0'
        extraProperties: |
          sonar.exclusions=**/obj/**,**/*.dll
          sonar.branch.name=$(Build.SourceBranchName)
          sonar.cs.vstest.reportsPaths=$(Agent.TempDirectory)/*.trx
          sonar.cs.opencover.reportsPaths=$(Build.SourcesDirectory)/**/*.xml
    - script: dotnet build 'Proj0/Proj0/Proj0.sln'
    - script: echo 'Project built'
    Settings
    - task: DotNetCoreCLI@2
      displayName: dotnet test
      inputs:
        command: 'test'
        arguments: '--configuration $(BuildConfiguration)
          /p:CollectCoverage=true
          /p:CoverletOutputFormat=opencover
          /p:Coverletoutput=$(Build.SourcesDirector)/TestResults/Coverage
          --logger trx'
        projects: '**/*XUnitTest_proj0.csproj'
        nobuild: true
    Settings
    - task: SonarCloudAnalyze@1
    Settings
    - task: SonarCloudPublish@1
      inputs:
        pollingTimeoutSec: '300'
    - script: dotnet publish 'Proj0/Proj0/Proj0.sln'
```

- Now that the test has been configured and code coverage configured we can save our .yml file and run it.
- The run should complete as shown
- By clicking the back arrow, we can go back to see the information generated



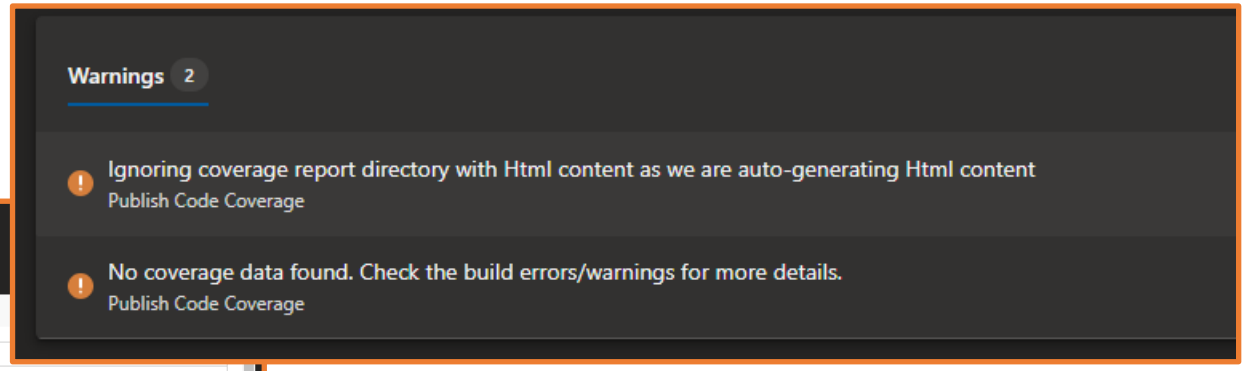
- If you click on the “Code Coverage” option you will notice there is no Code Coverage published to Azure DevOps
- This is because we did not add a task to publish code coverage to Azure. We will do this in the next slide
- Even though we have no code coverage report in Azure, we should still be able to see them in SonarCloud

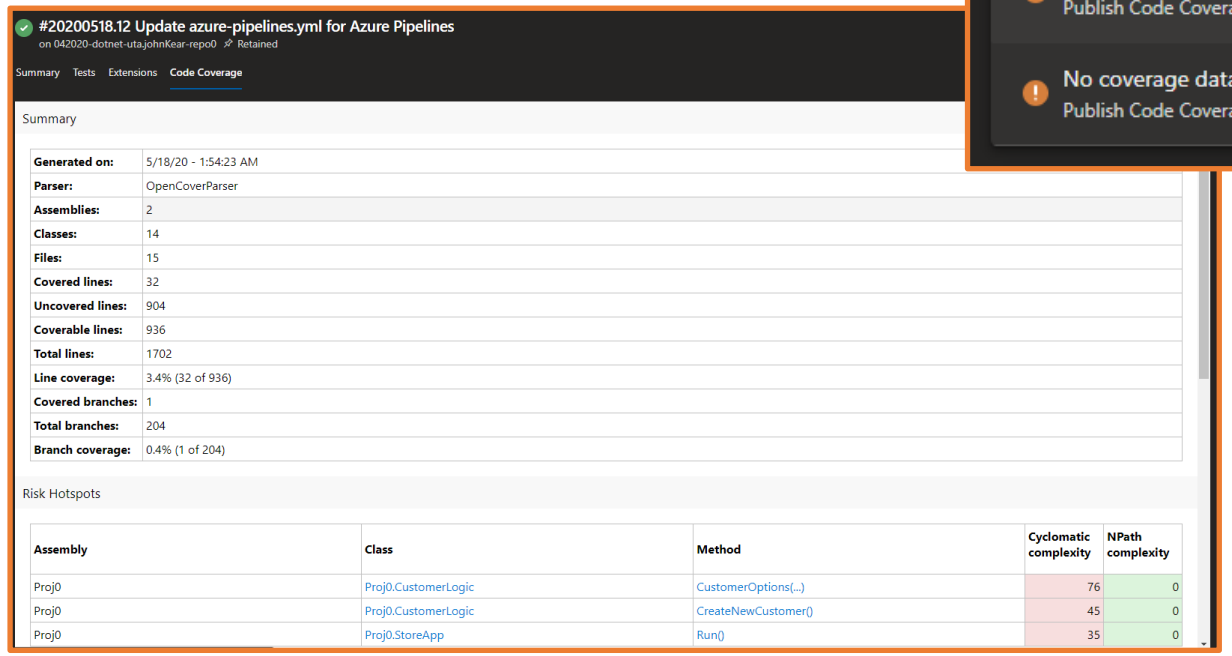


```yaml
- task: DotNetCoreCLI@2
  displayName: dotnet test
  inputs:
    command: 'test'
    arguments: '--configuration $(BuildConfiguration)
    /p:CollectCoverage=true /p:CoverletOutputFormat=opencover
    /p:Coverletoutput=$(Build.SourcesDirector)/TestResults/Coverage
    --logger trx'
    projects: '**/*XUnitTest_proj0.csproj'
    nobuild: true
Settings
- task: PublishCodeCoverageResults@1
  displayName: 'Publish Code Coverage'
  inputs:
    codeCoverageTool: Cobertura
    summaryFileLocation: '$(Build.SourcesDirectory)/**/*.xml'
    reportDirectory: '$(Build.SourcesDirectory)/CodeCoverage'
Settings
- task: SonarCloudAnalyze@1
Settings
- task: SonarCloudPublish@1
  inputs:
    pollingTimeoutSec: '300'
- script: dotnet publish 'Proj0/Proj0/Proj0.sln'
```

- Optionally, we can add a CodeCoveragePublish task so that we will also have the code coverage report in our Azure DevOps.
- This is done with a PublishCodeCoverageResults@1 task
- The task must come somewhere AFTER the test task (the code coverage report is generate in the test task)
- The input codeCoverageTool will be Cobertura
- The summaryFileLocation we can set to $(Build.SourcesDirectory)**/*.xml
- the report Directory we can set to $(Build.SourcesDirectory)/CodeCoverage

- After you save the changes and the pipeline runs, you will see that we now have results in the “Code Coverage” section.
- These results should match what is shown in sonar cloud.
- If you have followed this tutorial, you will also notice two warnings generated by the PublishCodeCoverageResults task. I have yet to investigate these and don’t know what is causing them.





- Back in sonar cloud you should see your project with the result including Coverage