



Build CI/CD Pipeline using Azure DevOps

Table of Contents

| | |
|------------------------------------|-----------|
| 1. Introduction..... | 3 |
| 2. Solution Components..... | 3 |
| 3. Solution Diagram | 3 |
| 4. Detailed Steps | 4 |
| 5. References | 17 |

1. Introduction

This document outlines how to use Azure DevOps to implement Continuous Integration and Continuous Deployment (CI/CD) to constantly and consistently test and build your code and ship it to any target. For our case, we will consider CI/CD on top of a sample Java (it can be extended to Scala with few changes) standalone application ensuring standard practices like automated test case execution and code quality are enforced.

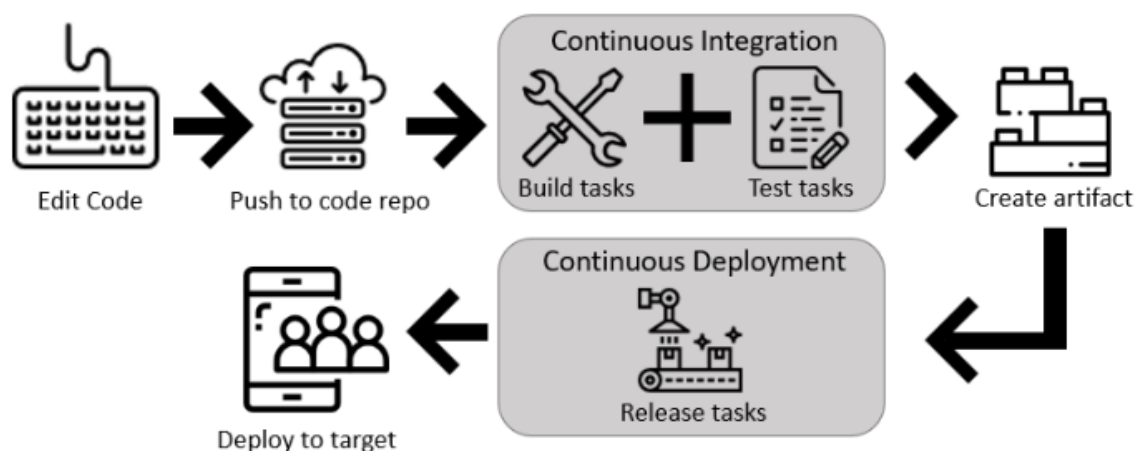
2. Solution Components

Continuous integration automates tests and builds for your project. CI helps to catch bugs or issues early in the development cycle, when they're easier and faster to fix. Items known as artifacts are produced from CI systems. They're used by the continuous delivery release pipelines to drive automatic deployments. The components we will use here are [GitHub](#) (Code Repository, Alt. [Azure Repo](#), [BitBucket](#), [SubVersion](#) etc.), [Maven](#) (Build Tool, Alt. [Ant](#) and [SBT](#)), [JUnit](#) (Unit Tests), [JaCoCo](#) (Code Coverage Tool, Alt. [Cobertura](#)) and [SonarQube](#) (Code Quality Tool, Alt. [PMD](#), [Checkstyle](#), [Findbugs](#) etc.).

Continuous delivery automatically deploys and tests code in multiple stages to help drive quality. Continuous integration systems produce deployable artifacts, which includes infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fixes to the target of your choice. For simplicity we will just move the artifact (jar) generated from CI pipeline to target [ADLS/Blob](#) storage location as our release event.

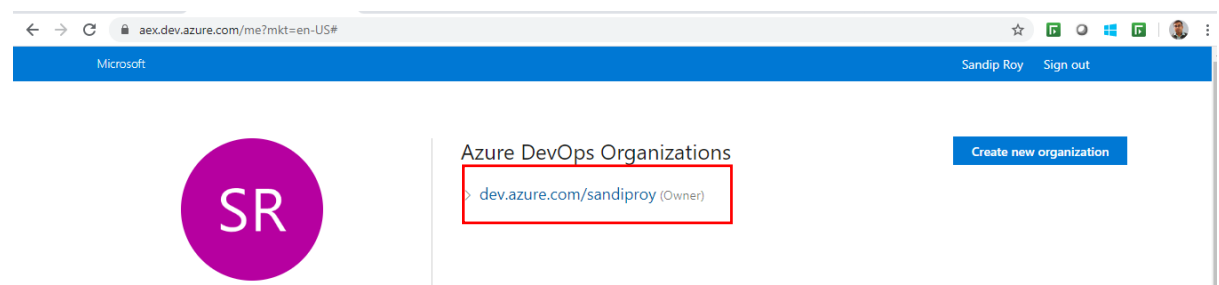
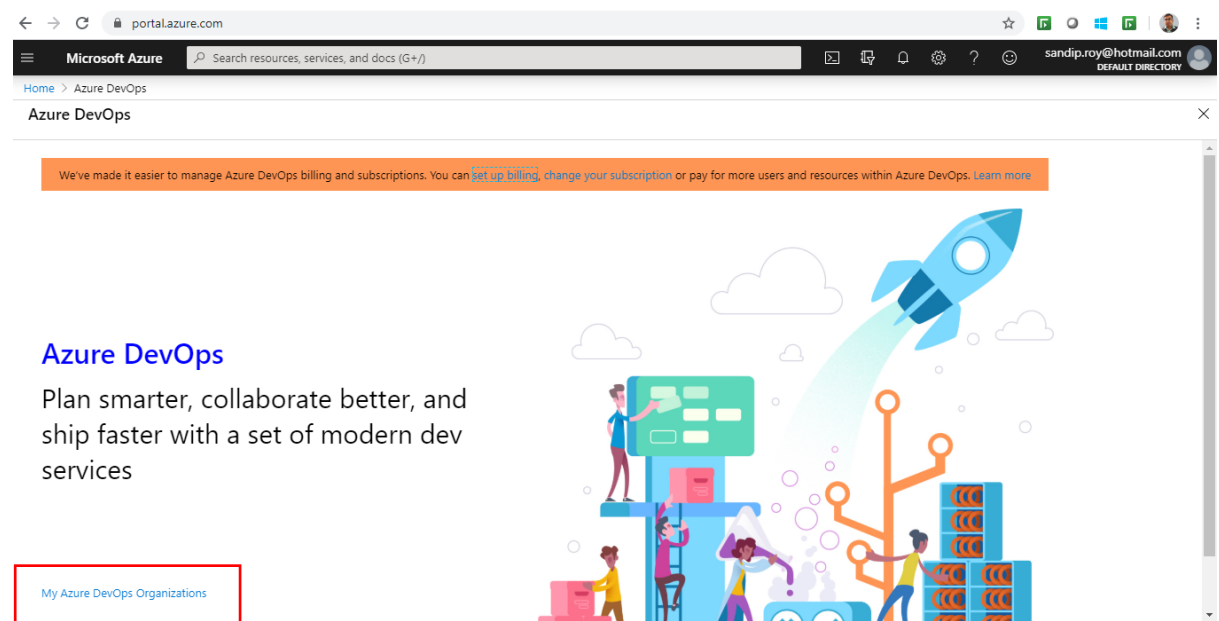
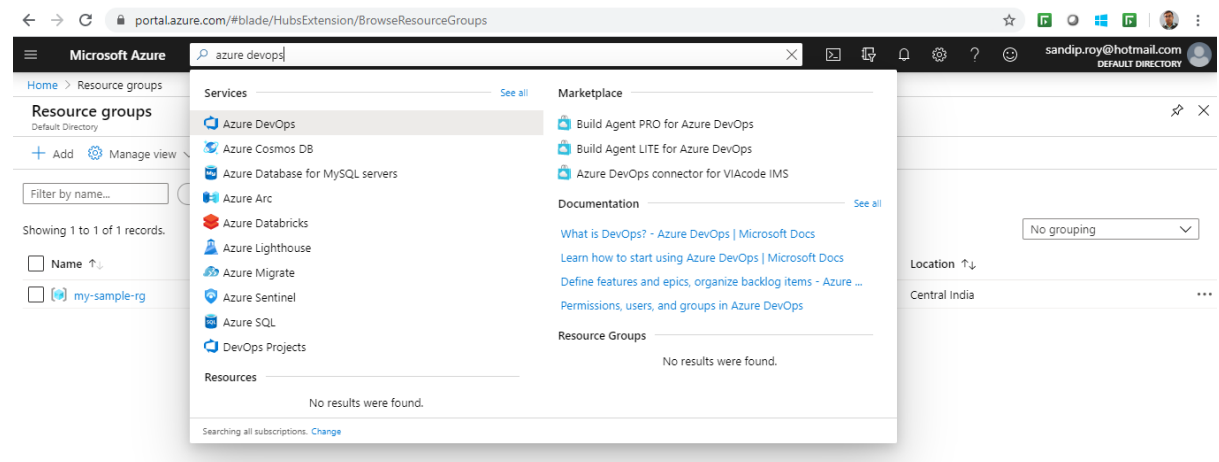
All above components are integrated with [Azure Pipelines](#) (cloud-hosted pipelines for Linux, macOS and Windows to build web, desktop and mobile applications and deploy the same to any cloud or on-premises environment).

3. Solution Diagram

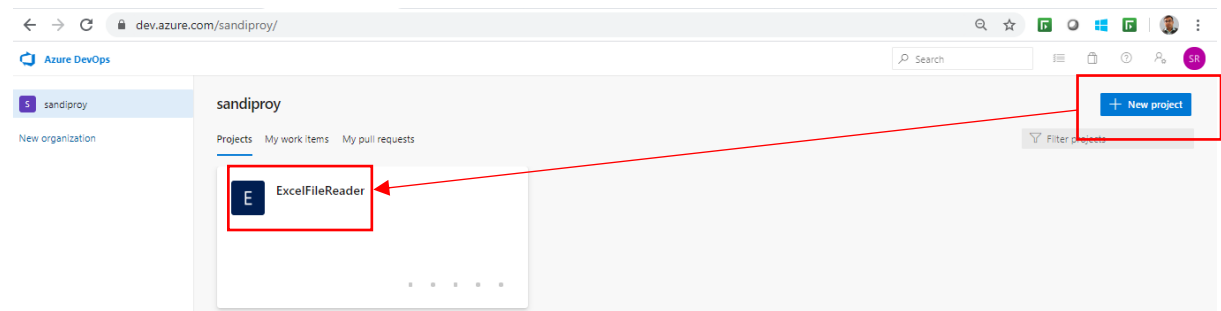


4. Detailed Steps

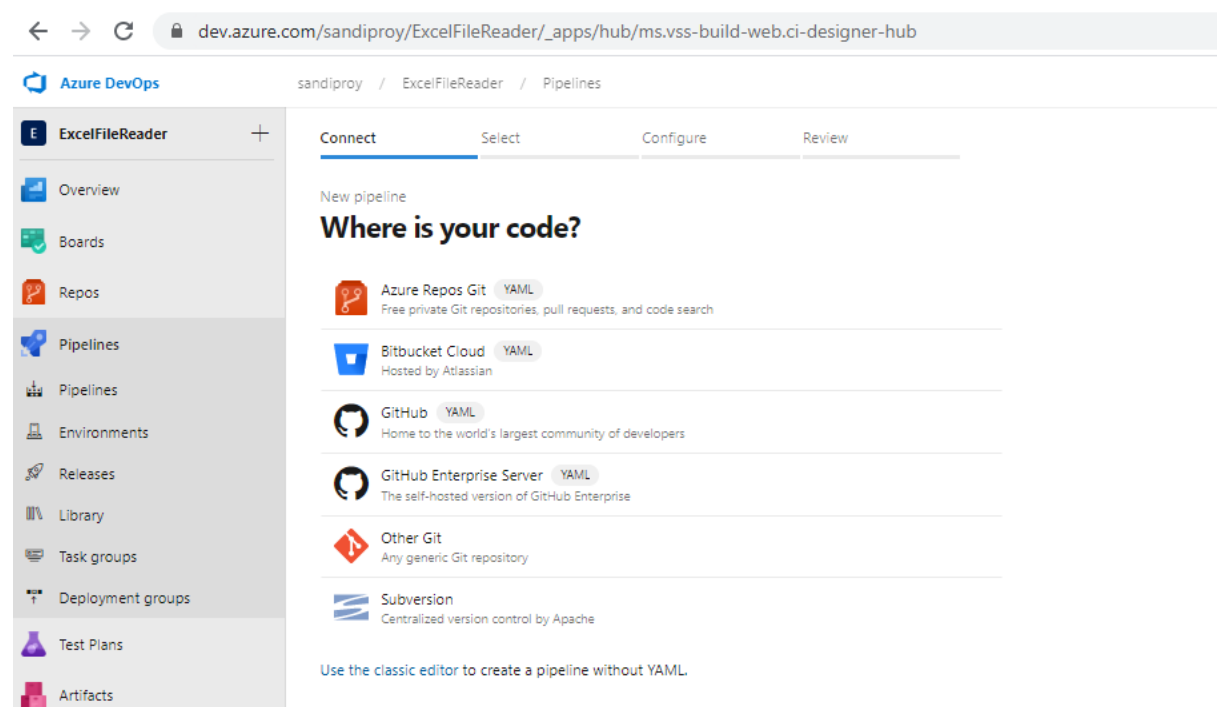
Step 1 – Navigate to Azure DevOps app and create new DevOps organization which would be hosting projects.



Step 2 – Once the organization is defined, you are ready to go ahead and create your DevOps project.

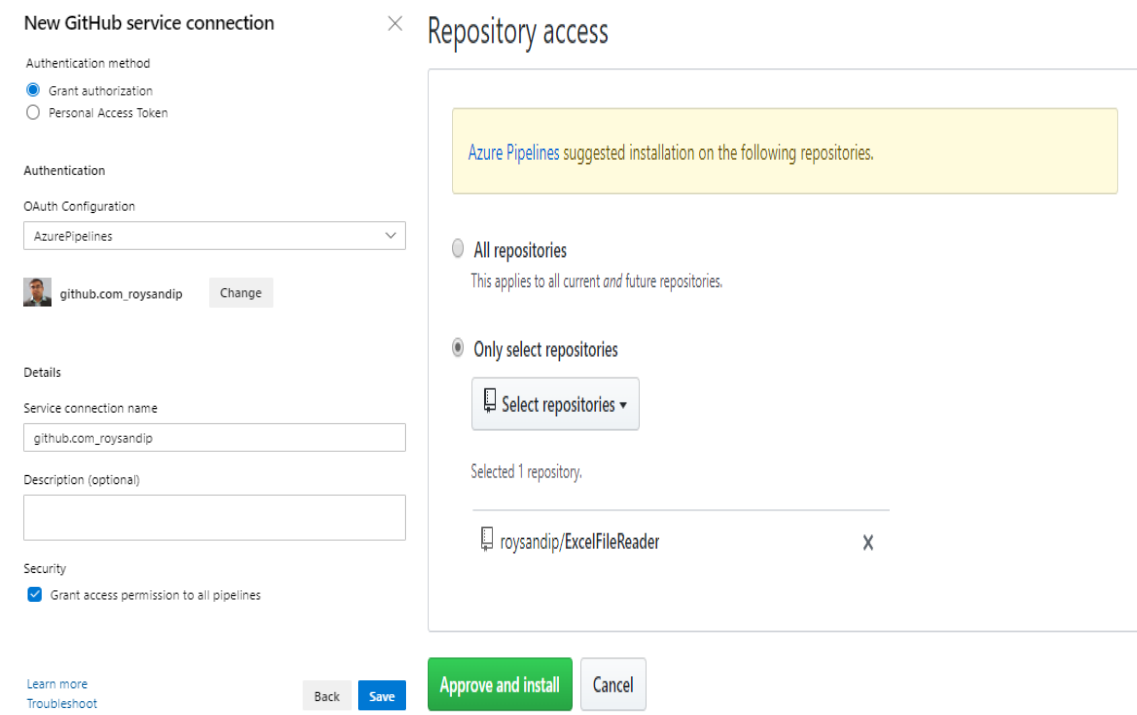
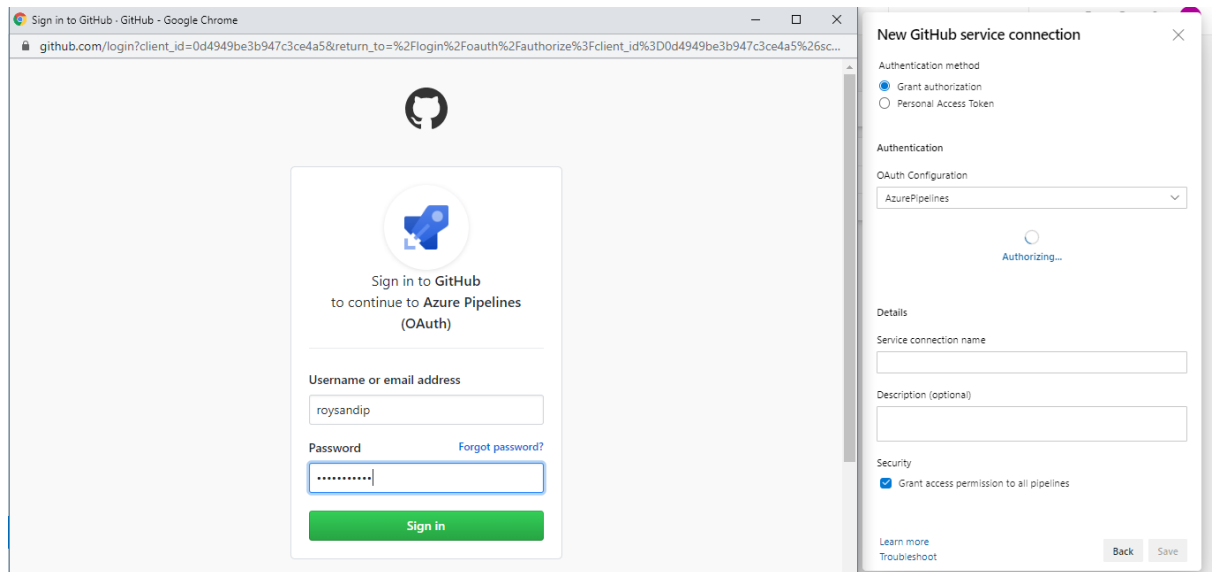
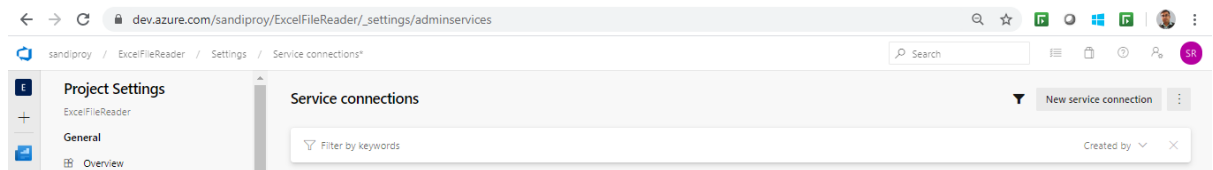


Upon the project skeleton is created, start creating the pipeline with connecting to your repository first.

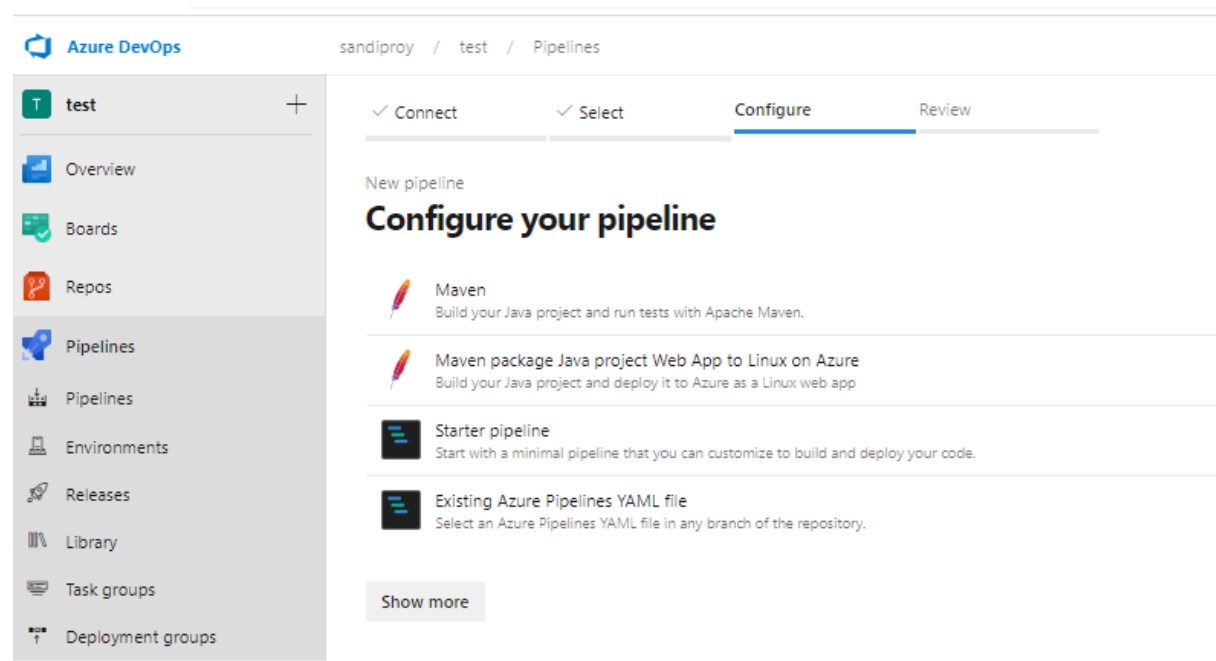


Once you select any of the repository type, Azure DevOps will try to connect to that using OAuth authentication mechanism (one time process only). This is basically authorizing Azure DevOps project account to interact/commit directly into your repository account.

Navigate through Project Settings > Service connections > New service connection > GitHub > Authorize



Once a particular project is selected from GitHub (I'll be using one of my simple Maven Java project hosted on GitHub repo @<https://github.com/roysandip/ExcelFileReader>) with public access) to be imported, automatically it will prompt for probable pipeline type (as shown below). As the project, we are importing here is Maven standalone project, we choose the first option i.e. Maven.



Once done, your project created with basic pipeline yml file where basic configurations are stored. Save it and dry run once for basic sanity.

```

trigger:
- master

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx1024m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.8'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: true
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    #codeCoverageTool: 'jaCoCo'
    #codeCoverageTool: 'Cobertura'
    #codeCoverageFailIfEmpty: true
    goals: 'package'
    sonarQubeRunAnalysis: true
    sqMavenPluginVersionChoice: 'latest'

```

Please note that for our case, we have configured code coverage enablement (through JaCoCo) well within Maven pom.xml and so we don't need to configure the same at Azure Pipeline level (yml file)

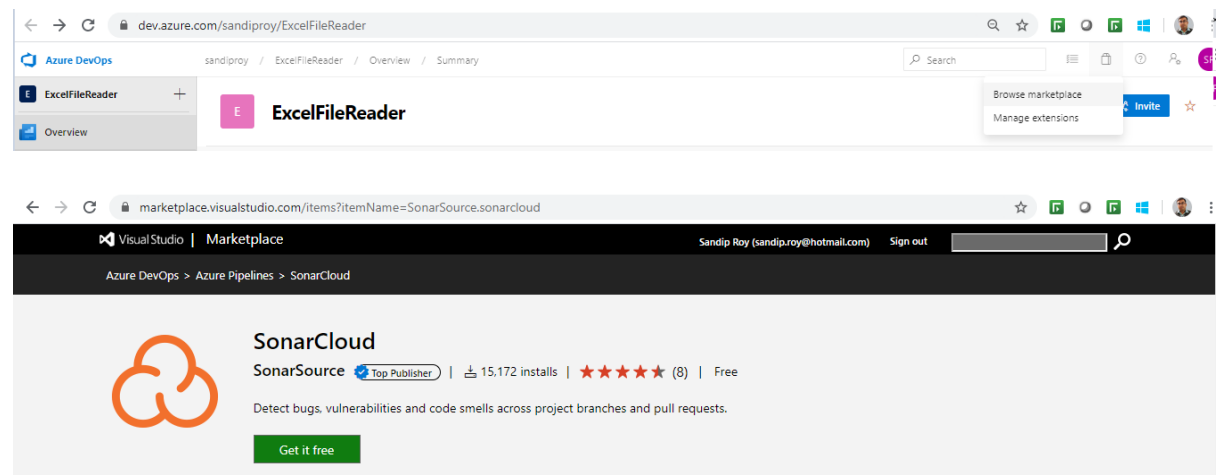
```

<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.8.2</version>
  <executions>
    <execution>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <execution>
      <id>report</id>
      <phase>test</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>

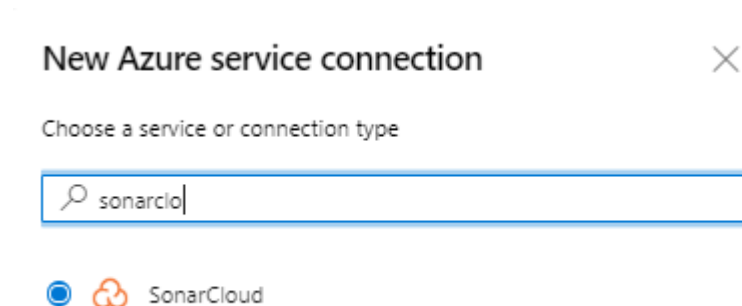
```


Step 3 – Integrate Azure DevOps project with SonarCloud (SaaS version of SonarQube) with following steps. Alternatively we can create Azure Container instances from SonarQube Docker instance available in Azure portal.

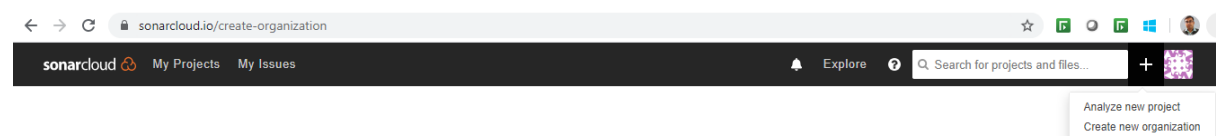
- a) Search for SonarCloud within Azure DevOps Marketplace and make that available within your Azure account.



- b) Once SonarCloud plugin is installed, navigate through Azure DevOps Project Settings > Service connections > New service connection > SonarCloud >



Note – Setup of SonarCloud organization/project @ <https://sonarcloud.io> in details is out of scope for this document but on high level you need to create an organization and project under it as below:



Create an organization

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

1 Enter your organization details

Key*

Up to 255 characters. All chars must be lower-case letters (a to z), digits or dash (but dash can neither be trailing nor heading).

[Add additional info ▼](#)

Continue

2 Choose a plan

Analyze projects - Set up manually

Organization*

sandiproj sandiproj ▼

[Create another organization](#)

Project key* ?

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

Display name* ?

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

Once the above are created, you need to create token (using it for authentication purpose instead of SonarCloud username/password) using navigation *My Account > Security > Generate Token* in SonarCloud.



Sandip Roy


[Profile](#)[Security](#)[Notifications](#)[Organizations](#)

Tokens


If you want to enforce security by not providing credentials of a real SonarCloud user to run your code scan or to invoke web services, you can provide a User Token as a replacement of the user login. This will increase the security of your installation by not letting your analysis user's password going through your network.

Generate Tokens

Generate



New token "my_sample" has been created. Make sure you copy it now, you won't be able to see it again!

 Copy

b6c552e2f780ab2a82ff0e24ae0c4a68b7fa689b

1

| Name | Last use | Created | |
|----------|------------|----------------|--------|
| my_token | 3 days ago | April 25, 2020 | Revoke |

- c) Once project token is ready, use it in SonarCloud service connection at project settings level in Azure DevOps as shown below:

New SonarCloud service connection



Authentication

SonarCloud Token

Authentication Token generated through SonarCloud (go to My Account > Security > Generate Tokens)

Verify

Details

Service connection name

my-sample-sonarcloud-conn

Description (optional)

Security

☒ Grant access permission to all pipelines

[Learn more](#)
[Troubleshoot](#)

Back

Verify and save



1

Put the token code as obtained from step above

Once SonarCloud settings is done, click “Verify and save” button to make sure everything is configured correctly.

Just couple of additional tasks, need to be in place apart from Maven build task at pipeline (yaml) level and that is - CopyFiles task to copy the artifact (the jar in this case) from Maven build target directory to artifact staging directory and then from there PublishBuildArtifacts task publishes the same to container drop location.

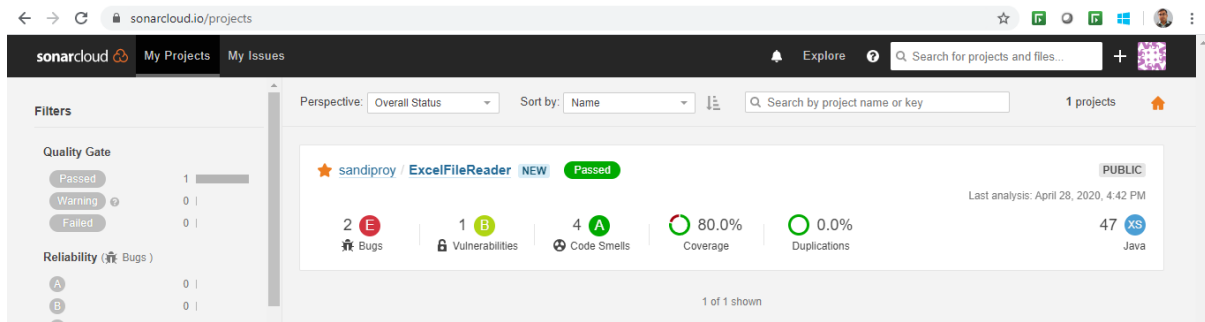
```
- task: CopyFiles@2
  inputs:
    sourceFolder: '$(Agent.BuildDirectory)/s/target'
    contents: '*.jar'
    targetFolder: '$(Build.ArtifactStagingDirectory) '

- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: '$(Build.ArtifactStagingDirectory) '
    artifactName: 'drop'
    publishLocation: 'Container'
```

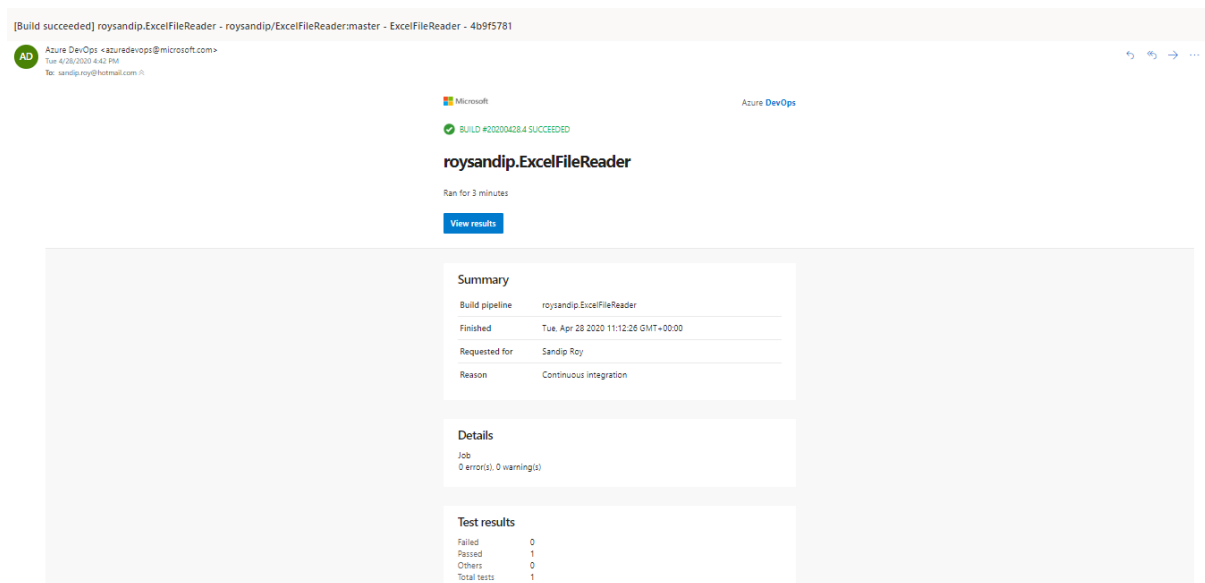
At this point, all the configuration are done from continuous integration (CI) perspective and once any code change is committed into GitHub automatically Azure Pipeline hook should be called for build, test case execution, code coverage and finally code quality through SonarCloud.

The screenshot shows the 'Jobs in run #20200428.4' for the pipeline 'roysandip.ExcelFileReader'. The job list includes: Initialize Job (2s), Checkout roysandip/Ex... (4s), Maven (3m 5s), CopyFiles (<1s), PublishBuildArtifacts (<1s), Post-job: Checkout ro... (<1s), and Finalize Job (<1s). The 'Job' details panel on the right shows a successful run with the following steps: 1. Pool: Azure Pipelines, 2. Image: ubuntu-latest, 3. Agent: Hosted Agent, 4. Started: Today at 4:39 PM, 5. Duration: 3m 13s, 6. Job preparation parameters, 7. 1 artifact produced, 8. 100% tests passed, 9. Job live console data, 10. Finishing: Job.

The screenshot shows the 'Artifacts' page for the build. It lists two published artifacts: 'drop' (4 KB) and 'ExcelFileReader-0.0.1-SNAPSHOT.jar' (4 KB). The 'drop' artifact is expanded, showing the file 'ExcelFileReader-0.0.1-SNAPSHOT.jar'.



Build Status Email sent to user



Once the CI pipeline is working, it's time to build the release (CD) pipeline and this release/deployment process may represent a wide variety of events like deploy the artifact to server, SSH artifact to certain location, publish some reports and many more.

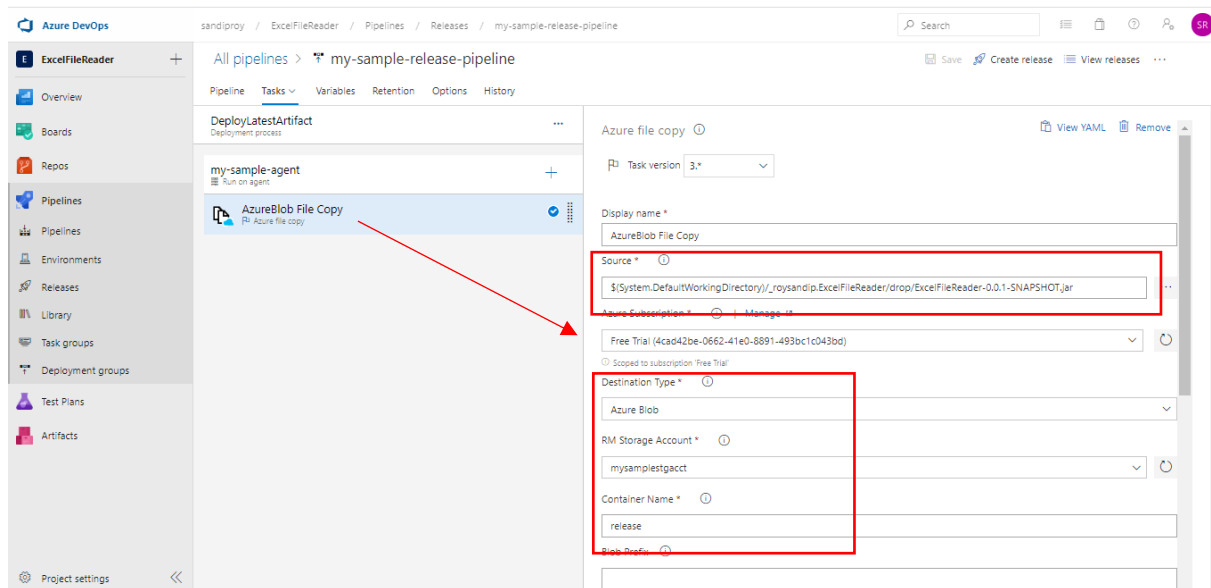
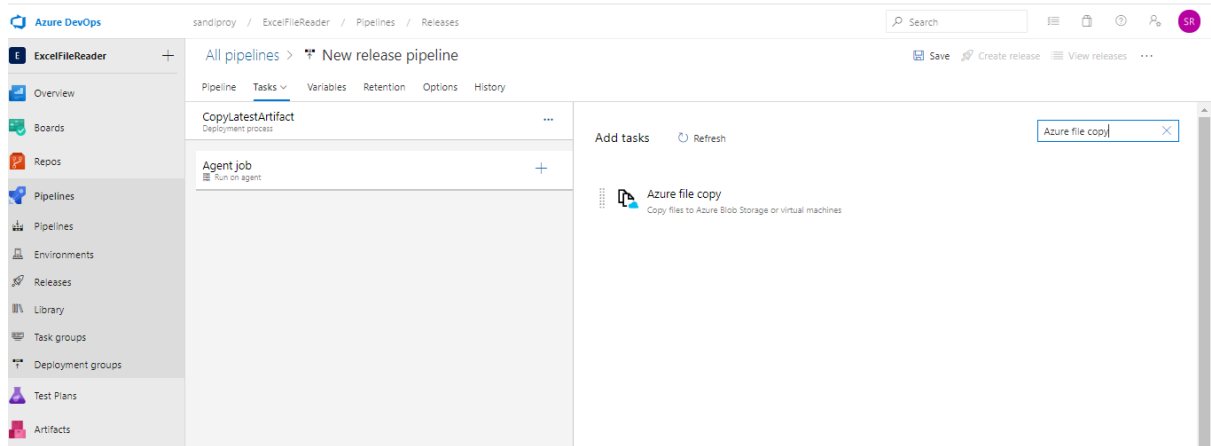
For our case, as mentioned before we would just push our artifact (jar) to specific ADLS/Blob storage location as part of our release/deployment process

Step 4 – Navigate to Releases tab from your Azure DevOps project and then create new release pipeline

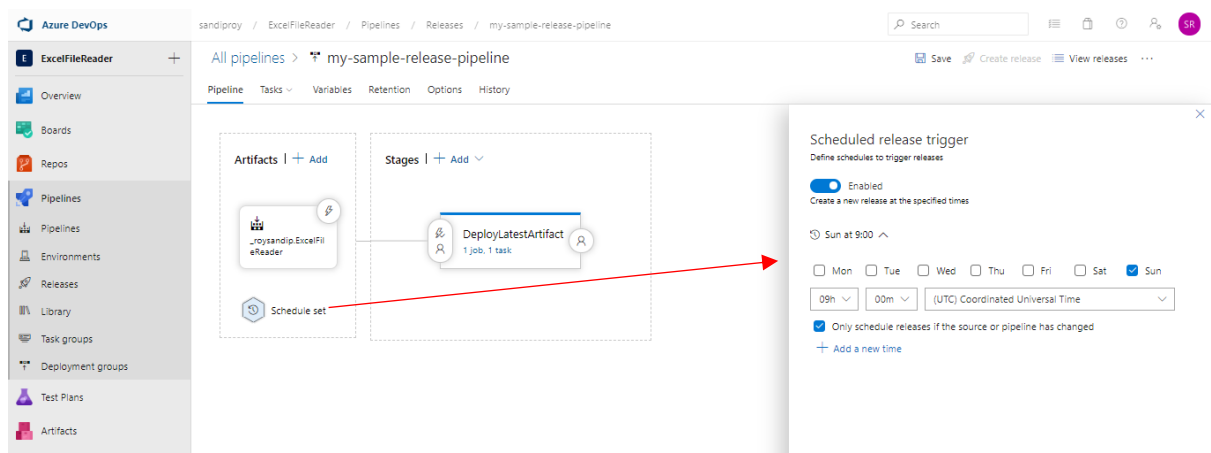
Azure DevOps interface showing the "New release pipeline" setup. The left sidebar lists navigation options: Overview, Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The main area displays the "New release pipeline" configuration with a warning: "You cannot save a release pipeline that contains zero stages." The configuration includes an "Artifacts" section with a "+ Add artifact" button and a "Stages" section with a "+ Add stage" button. A "Schedule not set" icon is visible in the Artifacts section.

Azure DevOps interface showing the "Add an artifact" dialog. The dialog is titled "Add an artifact" and lists source types: Build, Azure Repos, GitHub, and TFVC. The "Build" source type is selected. The "Project" dropdown is set to "ExcelFileReader". The "Source (build pipeline)" dropdown is set to "_roysandip.ExcelFileReader". The "Default version" dropdown is set to "Latest". The "Source alias" field is set to "_roysandip.ExcelFileReader". A note states: "The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of _roysandip.ExcelFileReader published the following artifacts: drop." The "Add" button is at the bottom right. A red arrow points from the "Add artifact" button in the previous screenshot to this dialog.

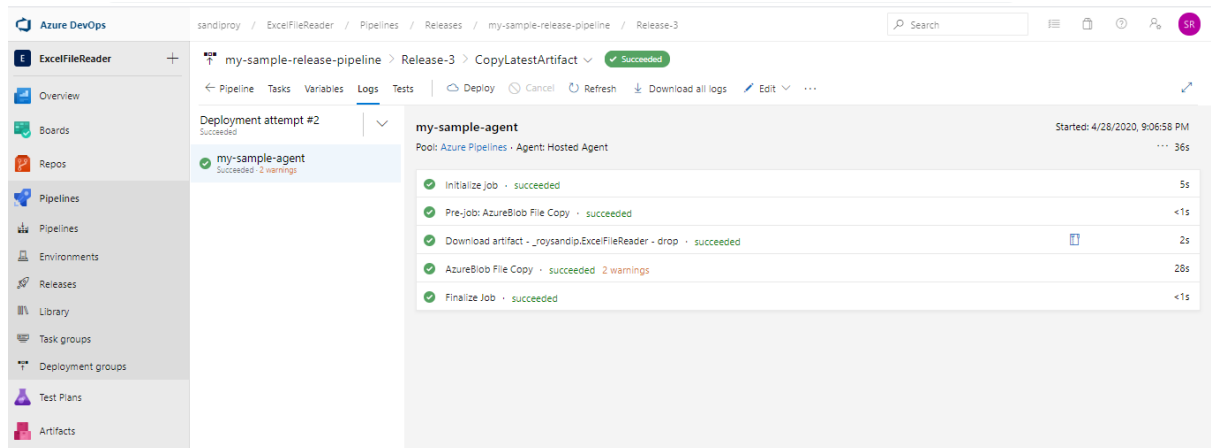
Azure DevOps interface showing the "Stage" configuration dialog. The dialog is titled "Stage" and shows the "CopyLatestArtifact" stage. The "Stage name" field is set to "CopyLatestArtifact". The "Stage owner" is set to "Sandip Roy". A red arrow points from the "CopyLatestArtifact" stage in the previous screenshot to this dialog.



Step 5 - Once above setting are done, schedule release/deployments through schedule i.e. when to go for deployment subjected to code changes.



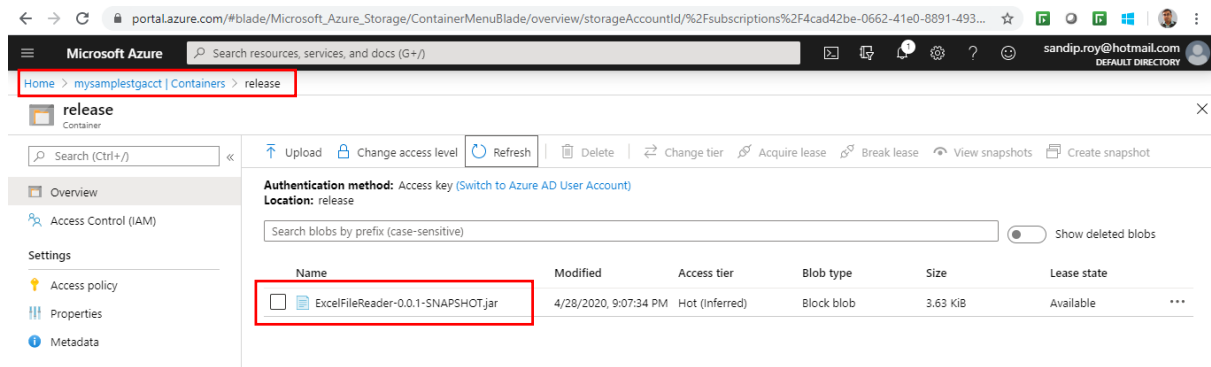
For testing purpose you can do one round of manual run just to see if everything is working or not.



Azure DevOps interface showing the execution of a release pipeline. The pipeline is named 'my-sample-release-pipeline' and is in the 'Release-3' state. The 'CopyLatestArtifact' task is successful. The 'my-sample-agent' is running on the 'Azure Pipelines - Agent: Hosted Agent' pool. The agent's log shows the following steps:

- Initialize job · succeeded (5s)
- Pre-job: AzureBlob File Copy · succeeded (<1s)
- Download artifact - _roysandip.ExcelFileReader - drop · succeeded (2s)
- AzureBlob File Copy · succeeded 2 warnings (28s)
- Finalize job · succeeded (<1s)

Once the release pipeline is executed, artifact (jar) is copied to desired Blob storage.



Microsoft Azure portal showing the 'release' container. The 'Overview' tab is selected, and the 'ExcelFileReader-0.0.1-SNAPSHOT.jar' artifact is listed in the table. The artifact is a 'Block blob' of size '3.63 KIB' and is in 'Available' state.

| Name | Modified | Access tier | Blob type | Size | Lease state |
|------------------------------------|-----------------------|----------------|------------|----------|-------------|
| ExcelFileReader-0.0.1-SNAPSHOT.jar | 4/28/2020, 9:07:34 PM | Hot (Inferred) | Block blob | 3.63 KIB | Available |

5. References

<https://docs.microsoft.com/en-us/azure/devops/pipelines/?view=azure-devops>
<https://azureddevopslabs.com/labs/vstsextend/sonarcloud/>