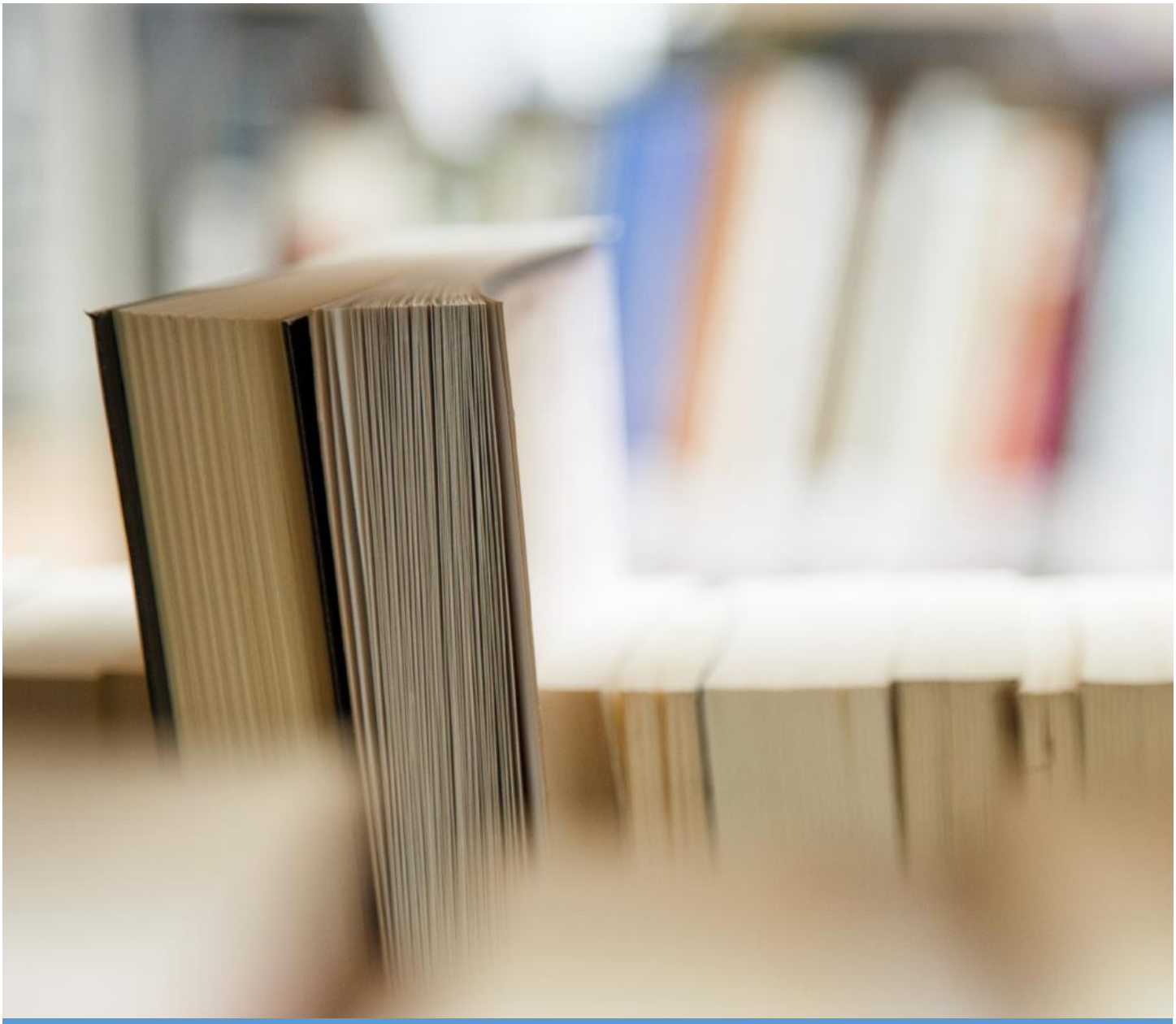


Build CI/CD Pipeline using Azure DevOps

Roy, Sandip (Cognizant)

5/1/20

Table of Contents

1. Introduction

This document outlines how to use Azure DevOps to implement **C**ontinuous **I**ntegration and **C**ontinuous **D**eployment (CICD) to constantly and consistently test and build your code and ship it to any target. For our case, we will consider CICD 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.

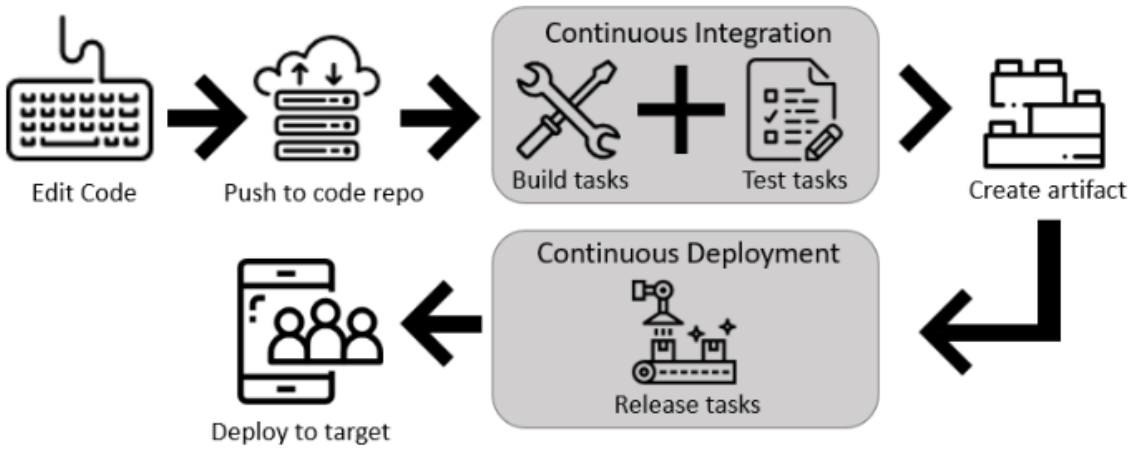
2. Solution Components

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 using 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.

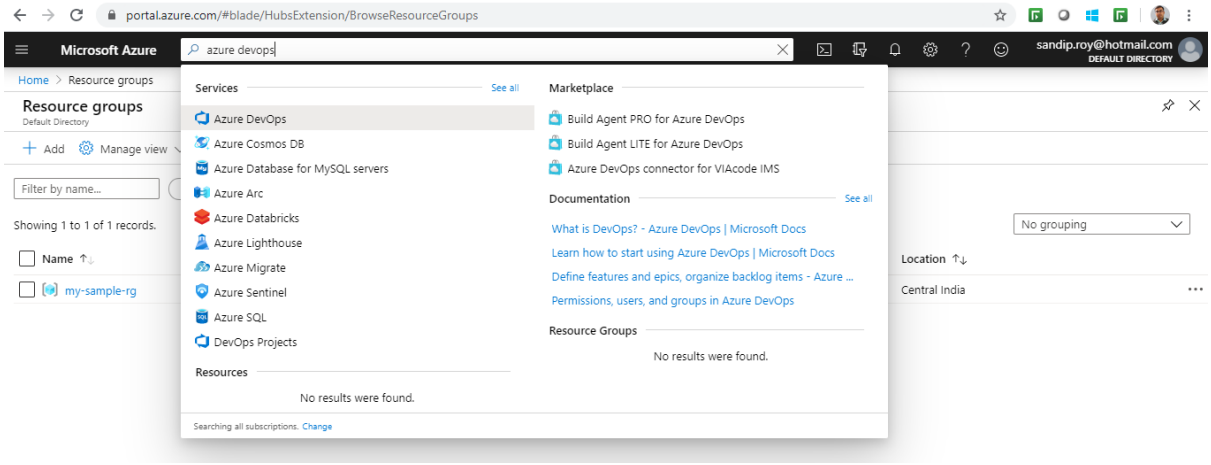
All above components are integrated with **Azure Pipelines** (cloud-hosted pipelines for Linux, macOS and Windows to build web, desktop and mobile applications and deploy the same to any cloud or on-premises environment).

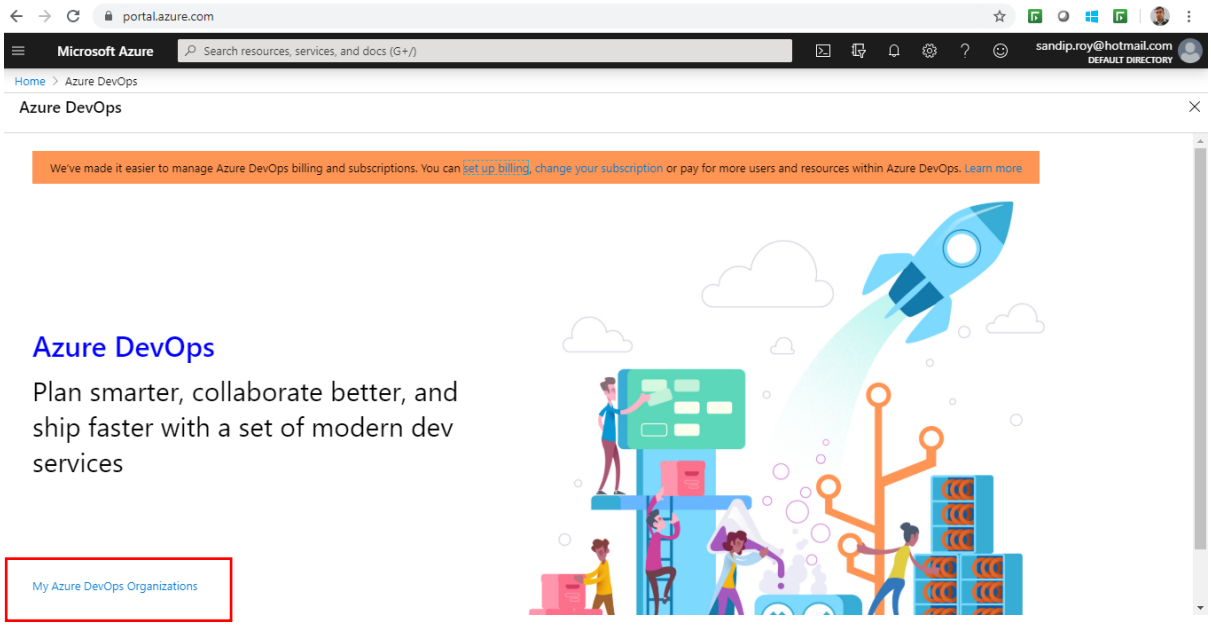
3. Solution Diagram

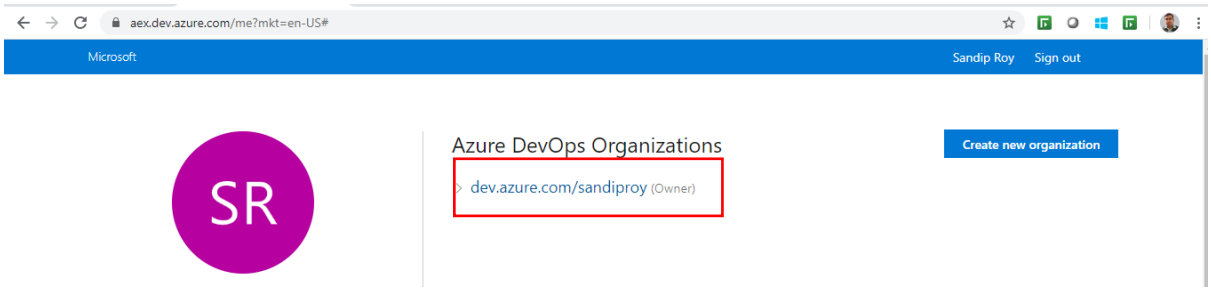


4. Detailed Steps

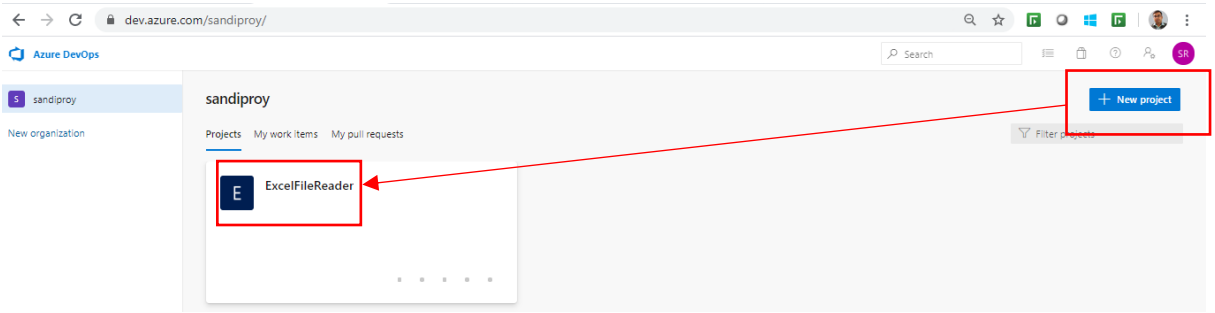
Step 1 – Navigate to Azure DevOps app and create new DevOps organization which would be hosting projects.



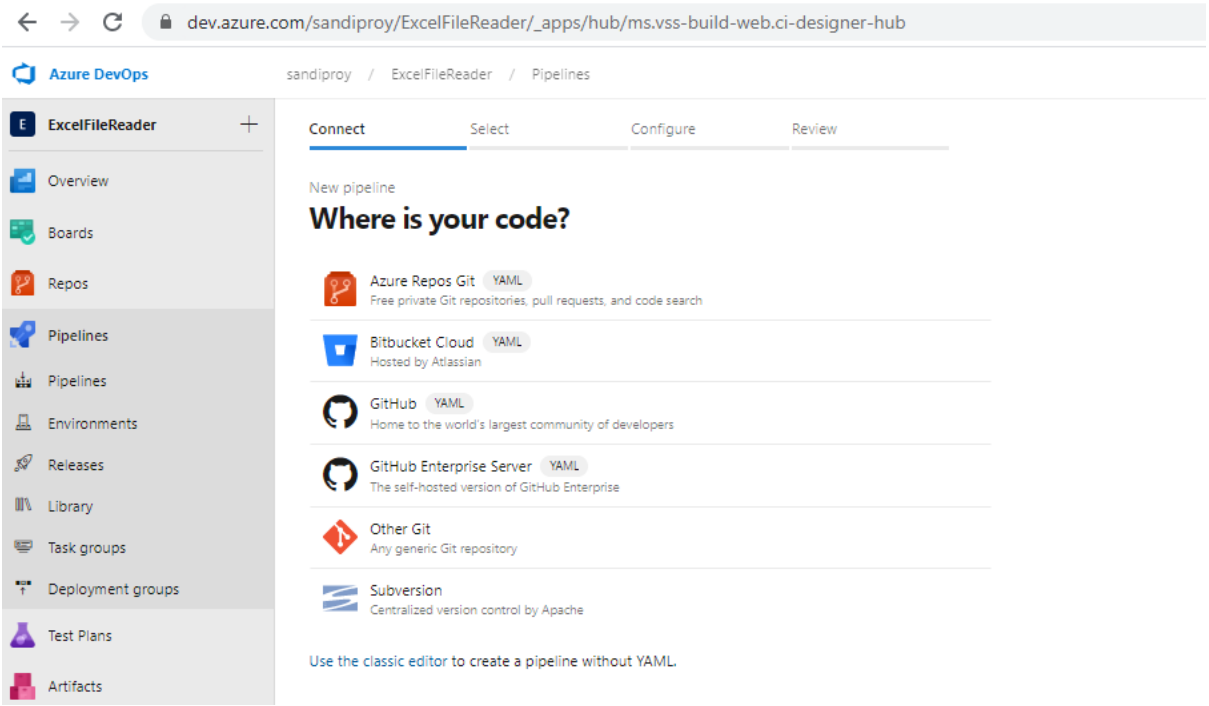




Step 2 – Once the organization is defined, you are ready to go ahead and create your DevOps project.

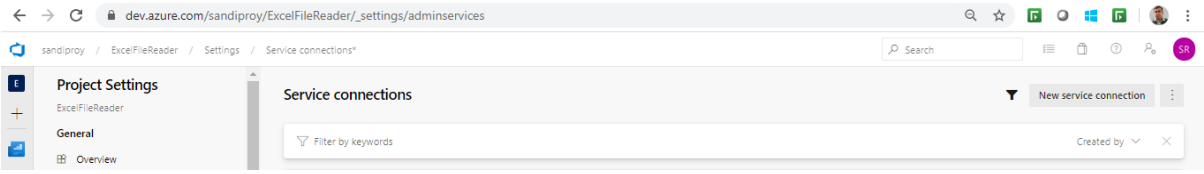


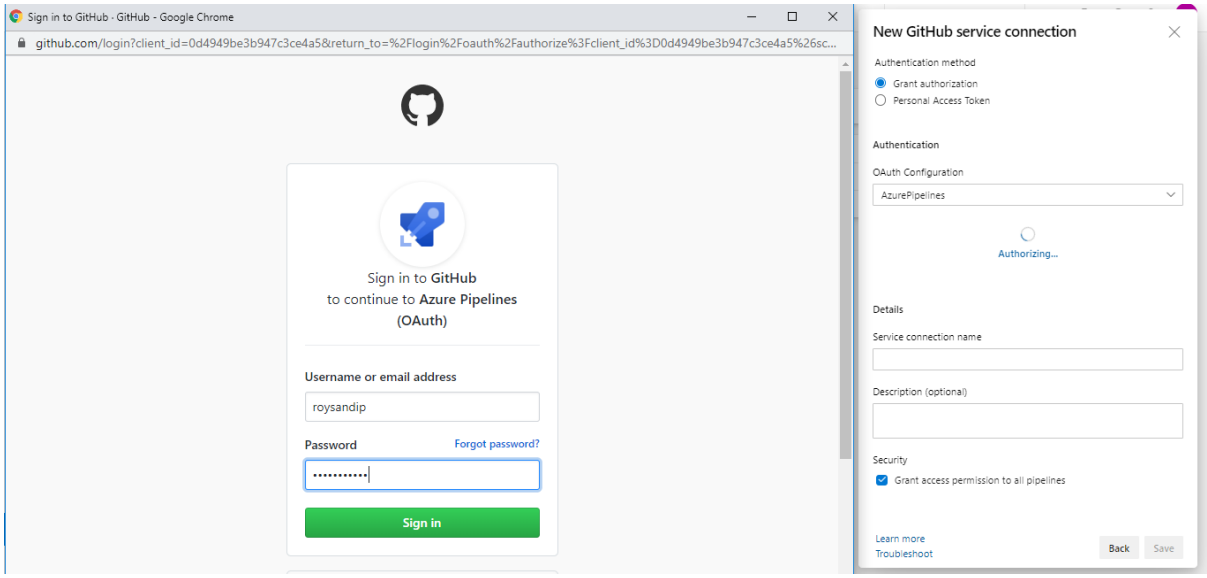
Upon the project skeleton is created, start creating the pipeline with connecting to your repository first.

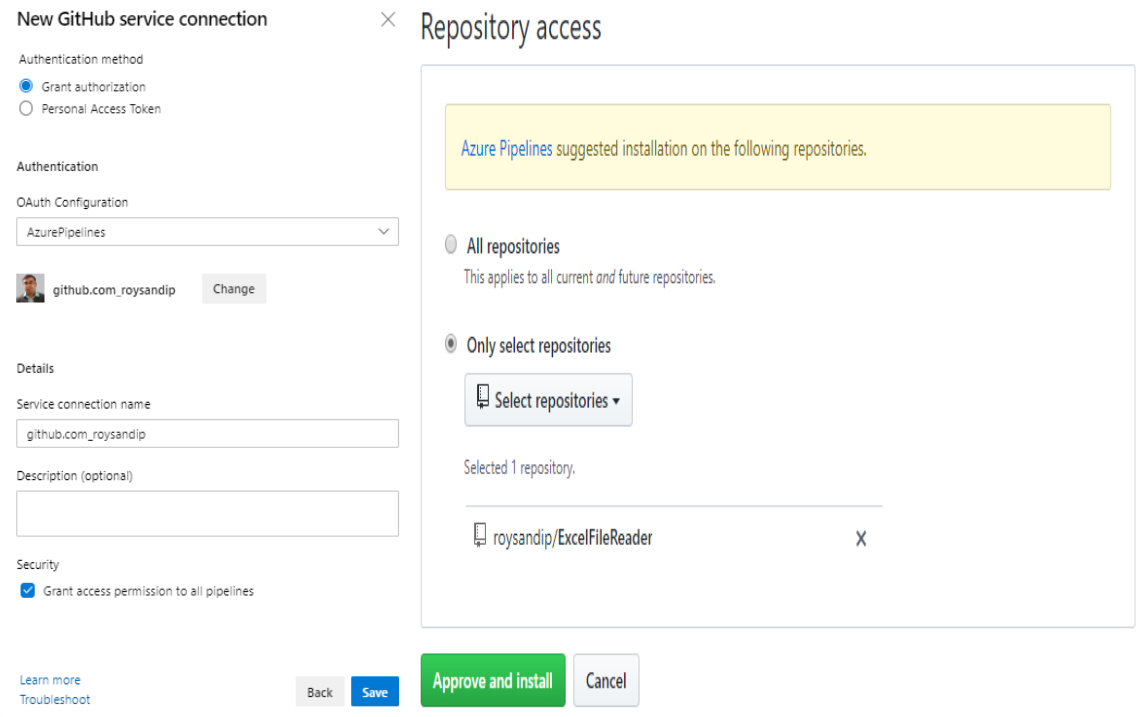


Once you select any of the repository type, Azure DevOps will try to connect to that using OAuth authentication mechanism (one time process only). This is basically authorizing Azure DevOps project account to interact/commit directly into your repository account.

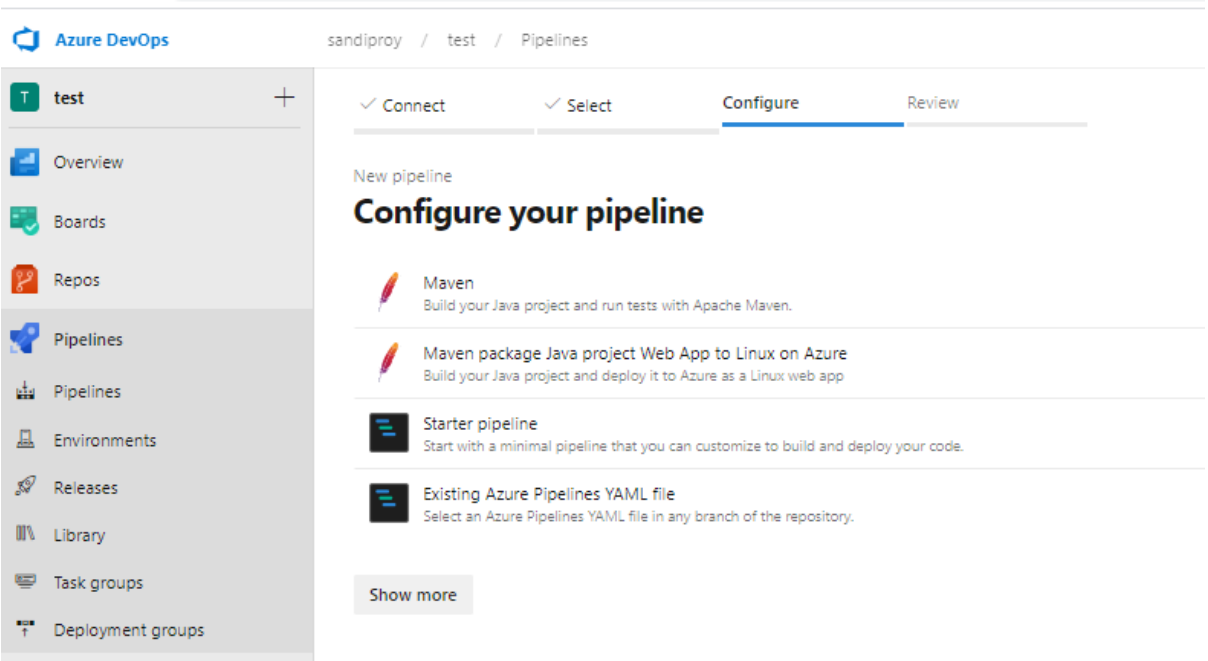
Navigate through Project Settings > Service connections > New service connection > GitHub > Authorize







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.

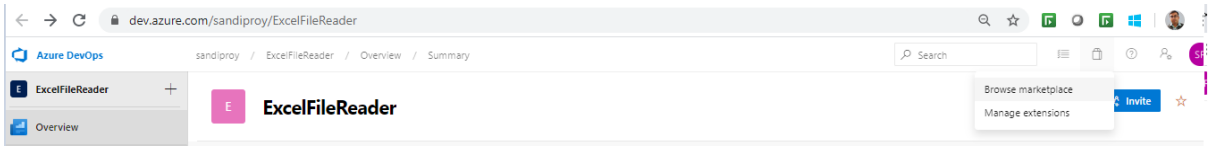
```yaml
trigger:
- master

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'
```

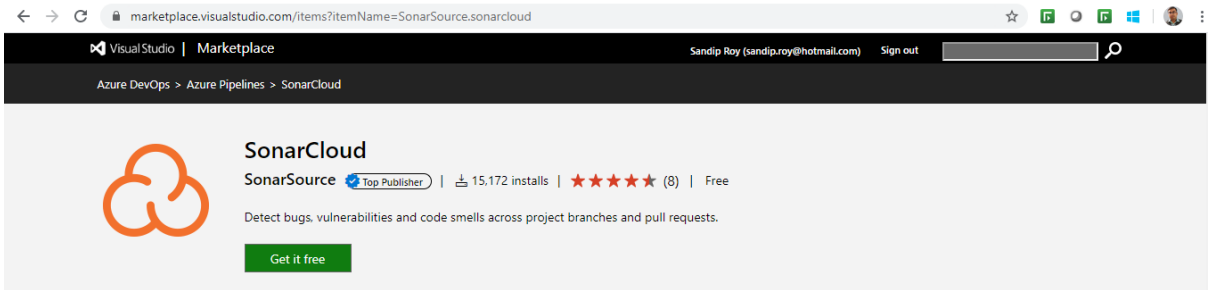
Please note that for our case, we have configured code coverage enablement (through JaCoCo) well within Maven pom.xml and so we don't need to configure the same at Azure Pipeline level (yml file)

```xml
<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>
```

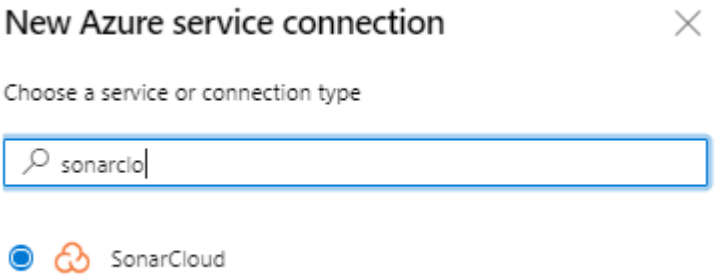
Step 3 – Integrate Azure DevOps project with SonarCloud (SaaS version of SonarQube) with following steps. Alternatively we can create Azure Container instances from SonarQube Docker instance available in Azure portal.

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

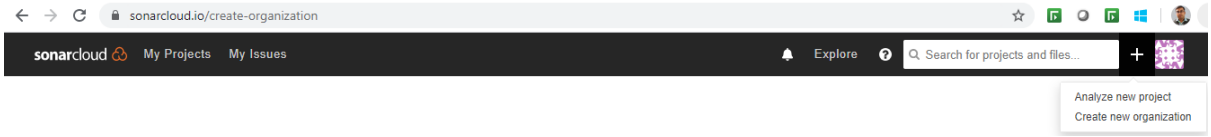




b) Once SonarCloud plugin is installed, navigate through Azure DevOps Project Settings > Service connections > New service connection > SonarCloud >



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*

sandiproy sandiproy

Create another organization

Project key*

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

Display name*

Up to 255 characters

Public

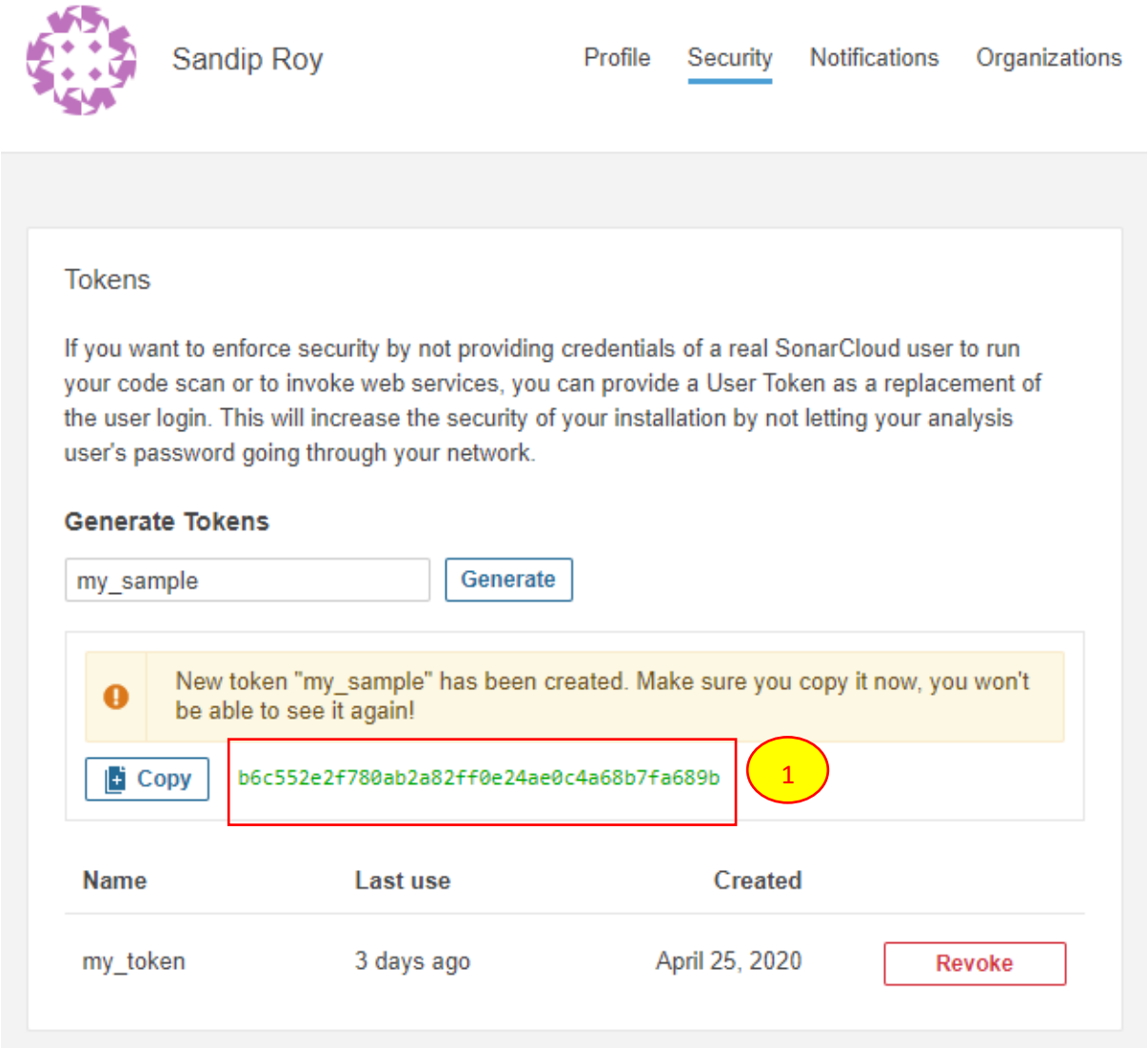
Anyone will be able to browse your source code and see the result of your analysis.

Private

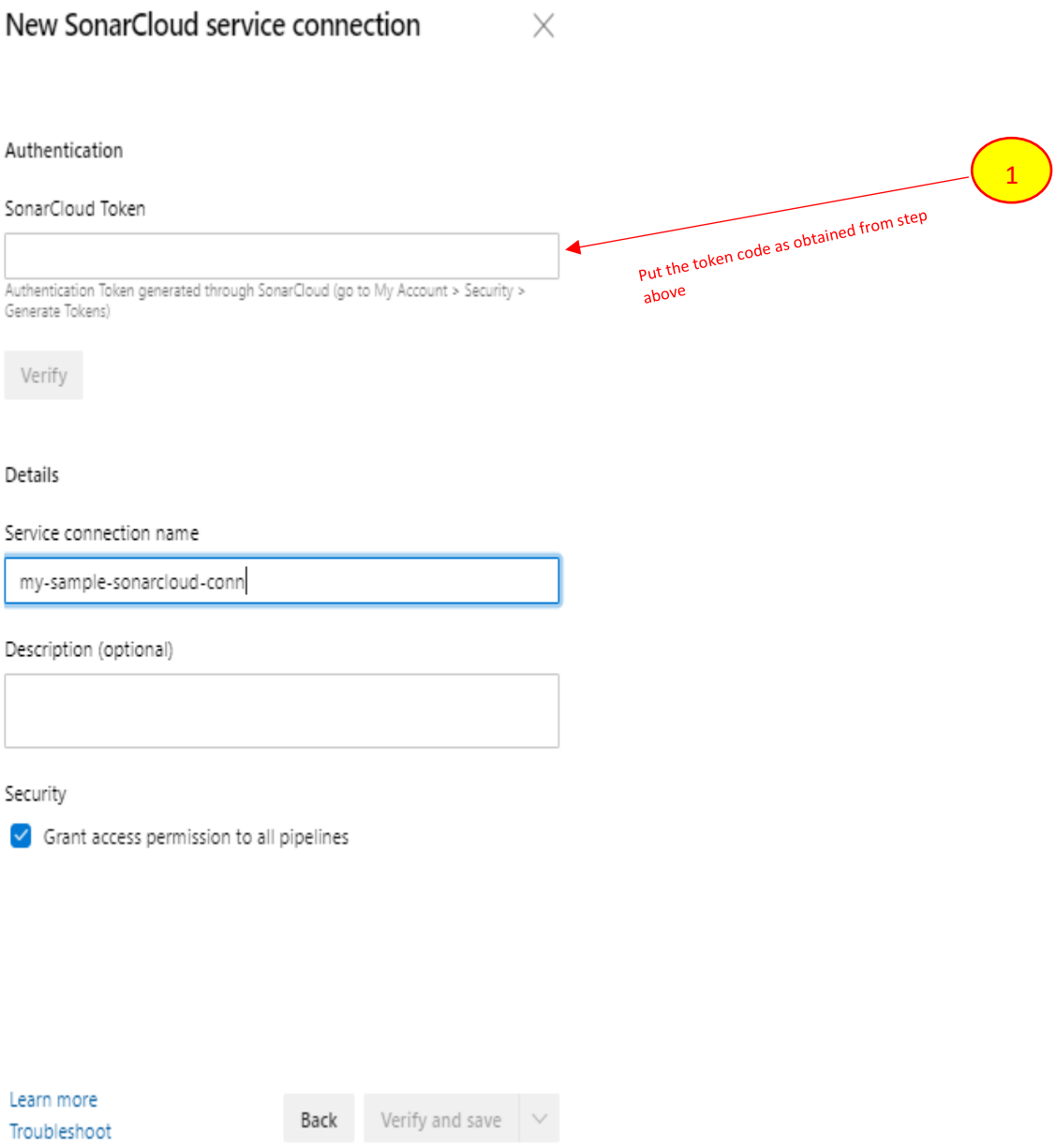
Only members of the organization will be able to browse your source code and see the result of your analysis.

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.



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

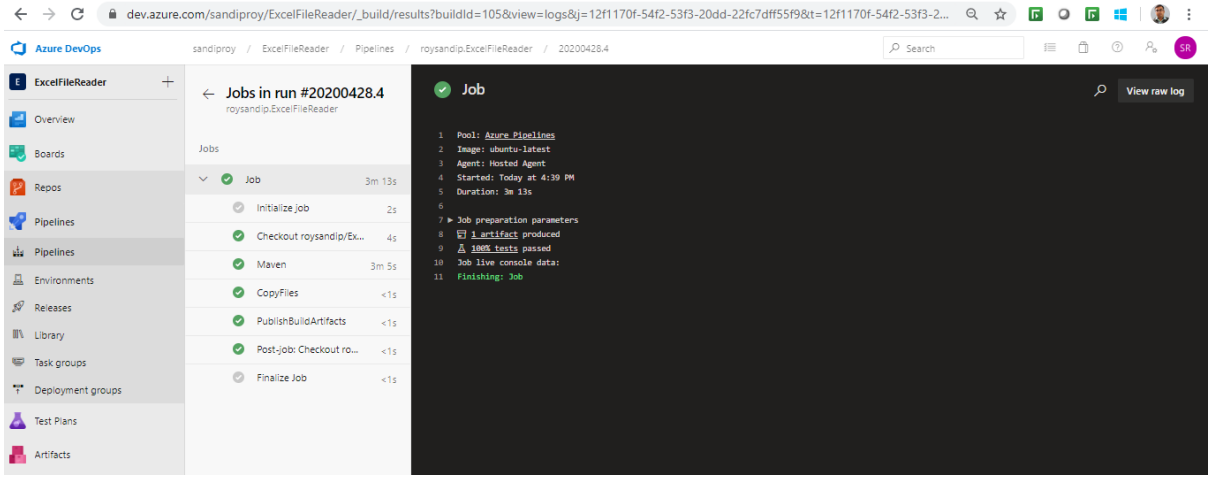


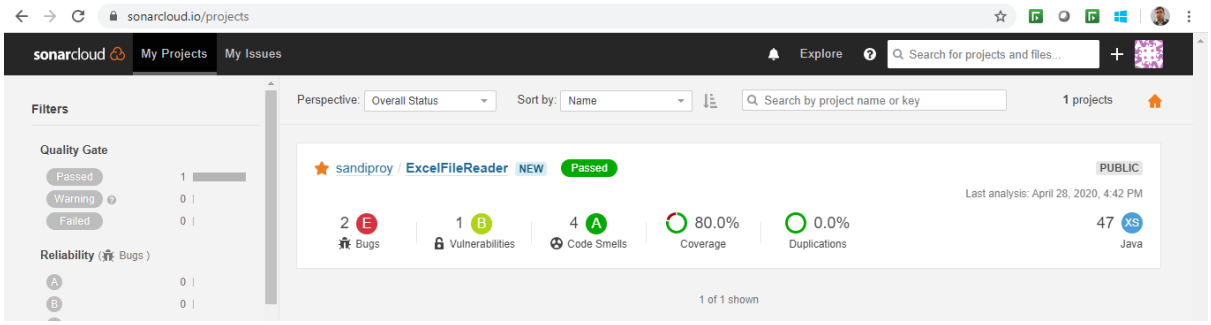
Once SonarCloud settings is done, click “Verify and save” button to make sure everything is configured correctly.

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.

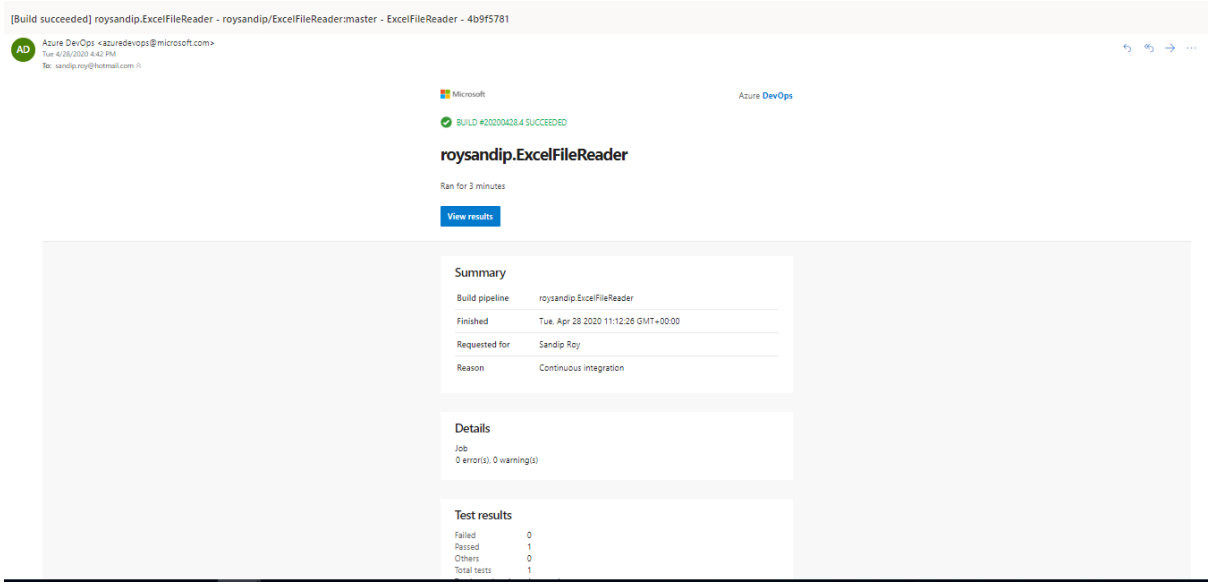
```
- task: CopyFiles@2
  inputs:
    sourceFolder: '$(Agent.BuildDirectory)/s/target'
    contents: '*.jar'
    targetFolder: '$(Build.ArtifactStagingDirectory)'


- 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 is committed into GitHub automatically Azure Pipeline hook should be called for build, test case excution, code coverage and finally code quality through SonarCloud.





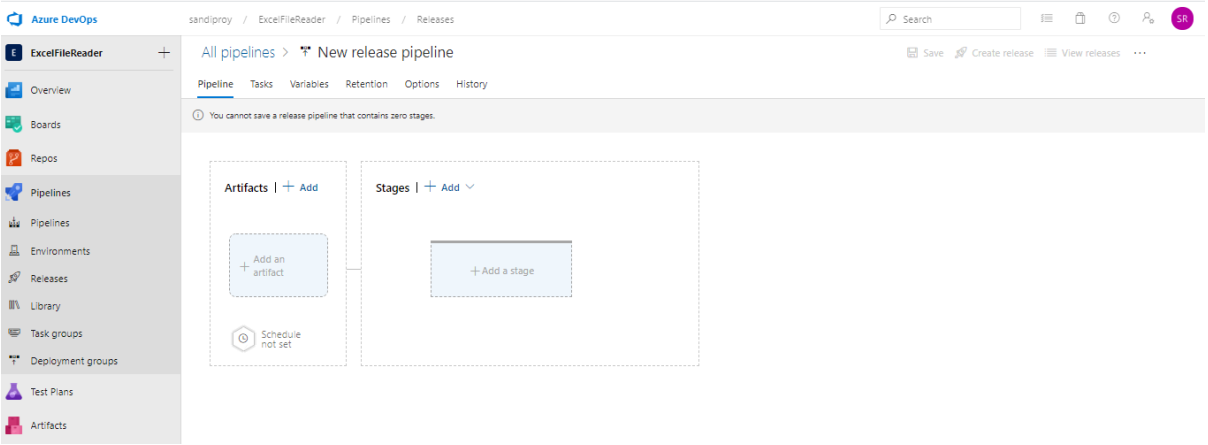
Build Status Email sent to user

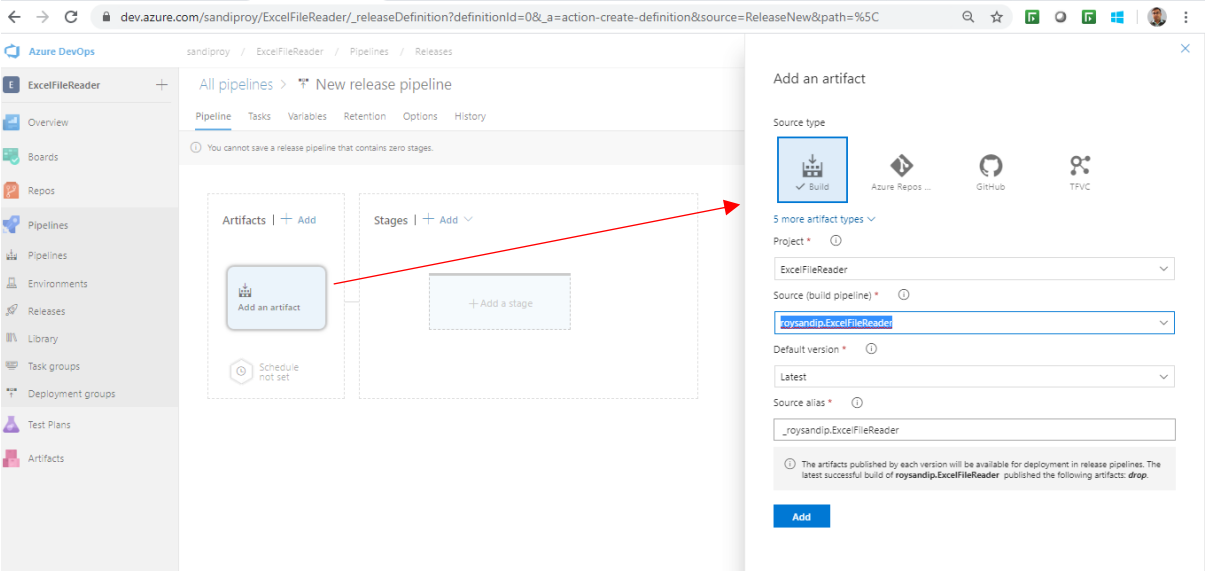


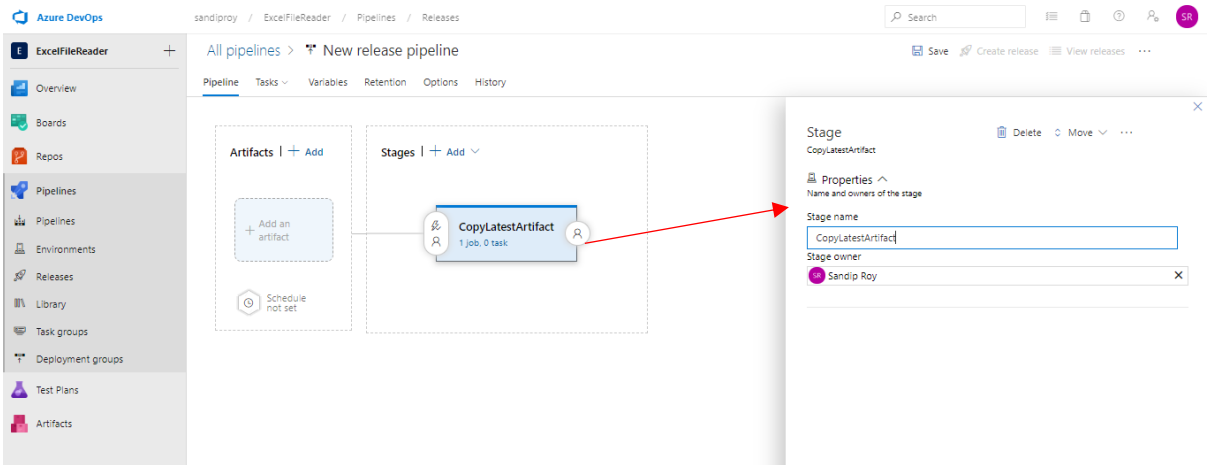
Once the CI pipeline is working, it's time to build the release (CD) pipeline and this release/deployment process may represent a wide variety of events like deploy the artifact to server, SSH artifact to certain location, publish some reports and many more.

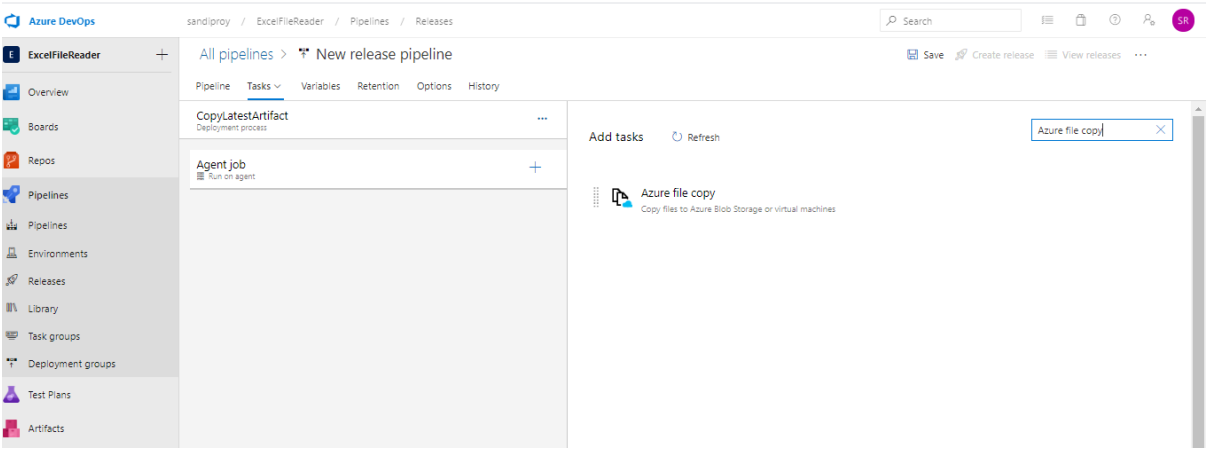
For our case, as mentioned before we would just push our artifact (jar) to specific ADLS/Blob storage location as part of our release/deployment process

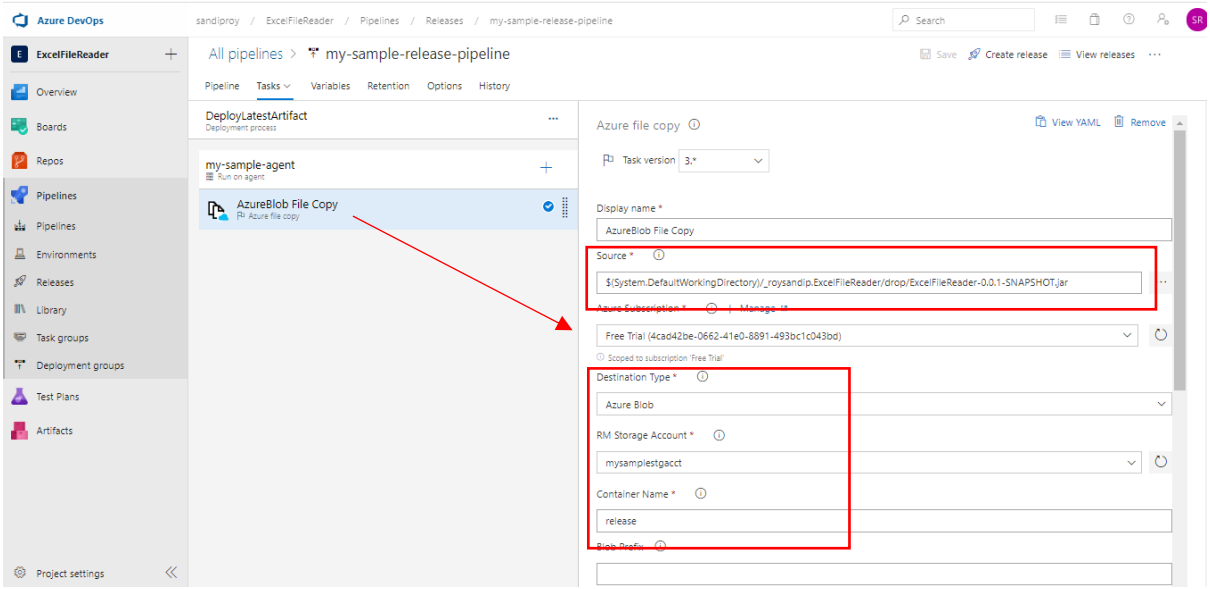
Step 4 – Navigate to Releases tab from your Azure DevOps project and then create new release pipeline


Azure DevOps
sandiproy / ExcelFileReader / Pipelines / Releases
Search
ExcelFileReader
All pipelines > New release pipeline
Save
Create release
View releases
Pipeline
Tasks
Variables
Retention
Options
History
You cannot save a release pipeline that contains zero stages.
Overview
Boards
Repos
Pipelines
Pipelines
Environments
Releases
Library
Task groups
Deployment groups
Test Plans
Artifacts
Artifacts | + Add
Add an artifact
Schedule not set
Stages | + Add
+ Add a stage

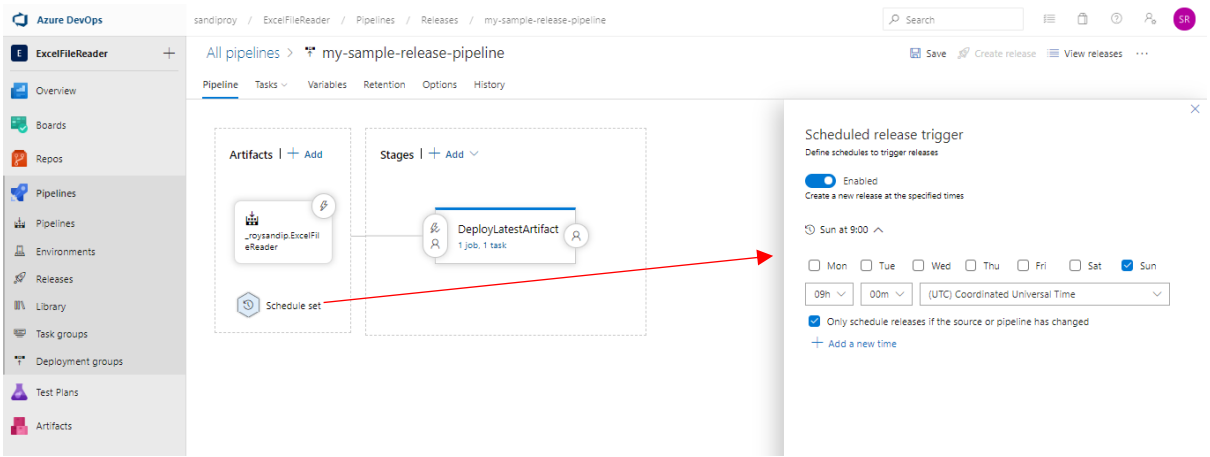

dev.azure.com/sandiproy/ExcelFileReader/_releaseDefinition?definitionId=0&_a=action-create-definition&source=ReleaseNew&path=%5C
Azure DevOps
ExcelFileReader
All pipelines > New release pipeline
Pipeline
Tasks
Variables
Retention
Options
History
You cannot save a release pipeline that contains zero stages.
Artifacts | + Add
Add an artifact
Stages | + Add
+ Add a stage
Add an artifact
Source type
Build
Azure Repos ...
GitHub
TFVC
5 more artifact types
Project *
ExcelFileReader
Source (build pipeline) *
roysandip.ExcelFileReader
Default version *
Latest
Source alias *
_roysandip.ExcelFileReader
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.
Add


Azure DevOps
sandiproy / ExcelFileReader / Pipelines / Releases
Search
ExcelFileReader
All pipelines > New release pipeline
Save
Create release
View releases
Pipeline
Tasks
Variables
Retention
Options
History
Artifacts | + Add
Add an artifact
Schedule not set
Stages | + Add
CopyLatestArtifact
1 job, 0 task
Stage
Delete
Move
CopyLatestArtifact
Properties
Name and owners of the stage
Stage name
CopyLatestArtifact
Stage owner
Sandip Roy

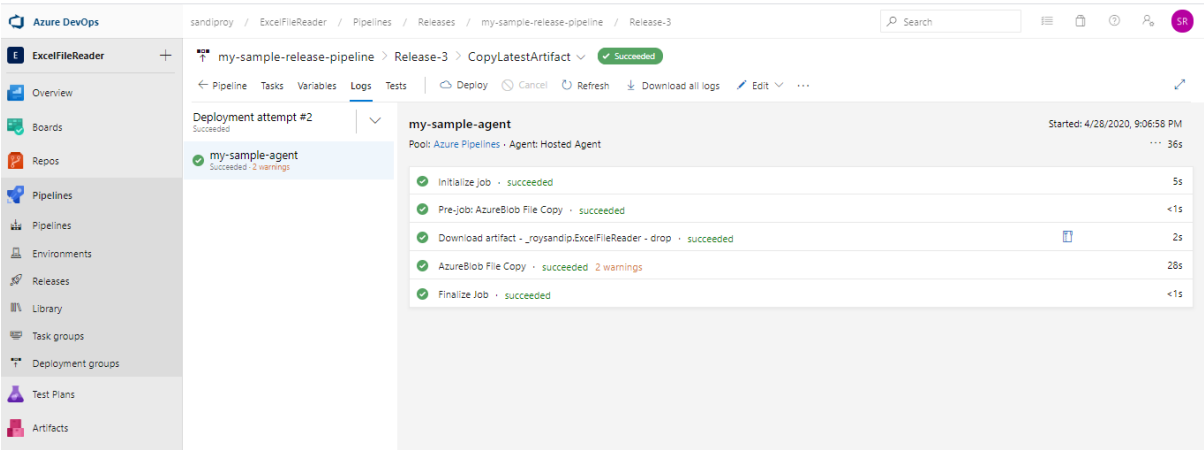




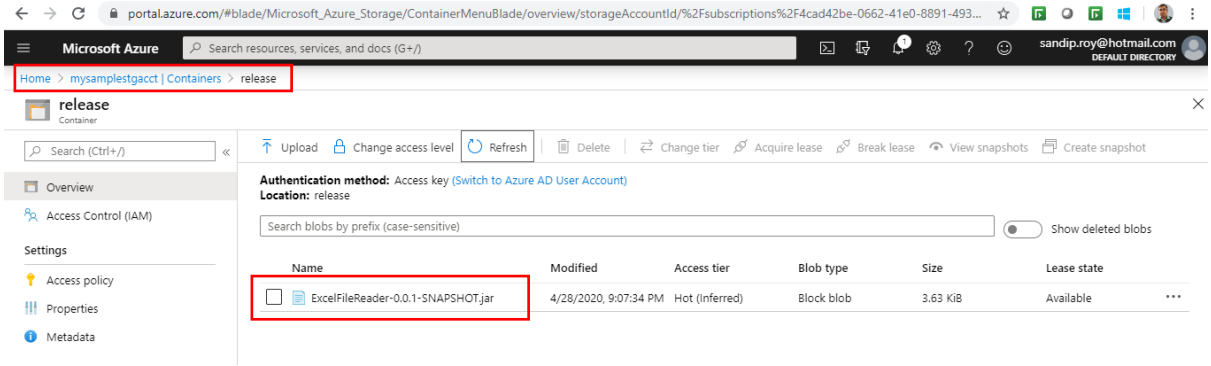
Step 5 - Once above setting are done, schedule release/deployments thorugh schedule i.e. when to go for deployment subjected to code changes.



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



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



5. References

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