RAMAIAH
Institute of Technology

DevOps Continuous Integration and Continuous Deployment Pipeline

RAMAIAH
Institute of Technology

Managing Your Source Code with Git

Introduction to DevOps

- **Version Control System** (VCS) or more commonly as a **Version Control Manager** (VCM)
- The goals of these VCSes are mainly to do the following:
  - Allow collaboration of developers' code.
  - Retrieve the code.
  - Version the code.
  - Track code changes.
- The implementation of a CI/CD process can only be done with a VCS as a prerequisite.

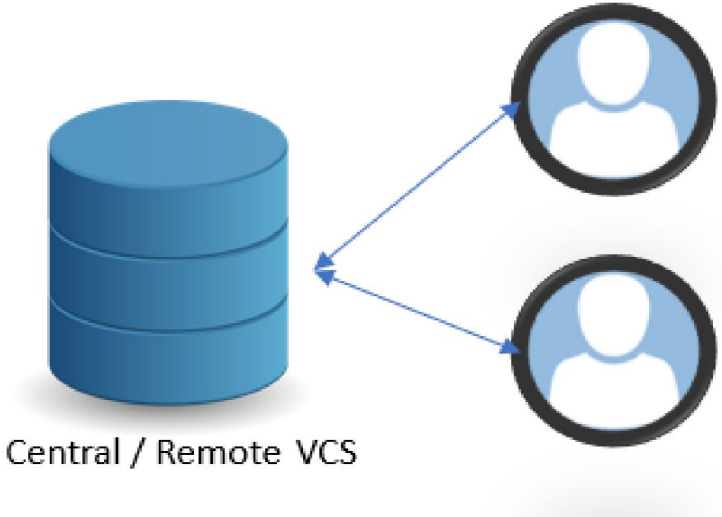
Overviewing Git and its command lines

- There are two types of VCS:
  1. Centralized Version Control System
  2. Distributed Version Control System

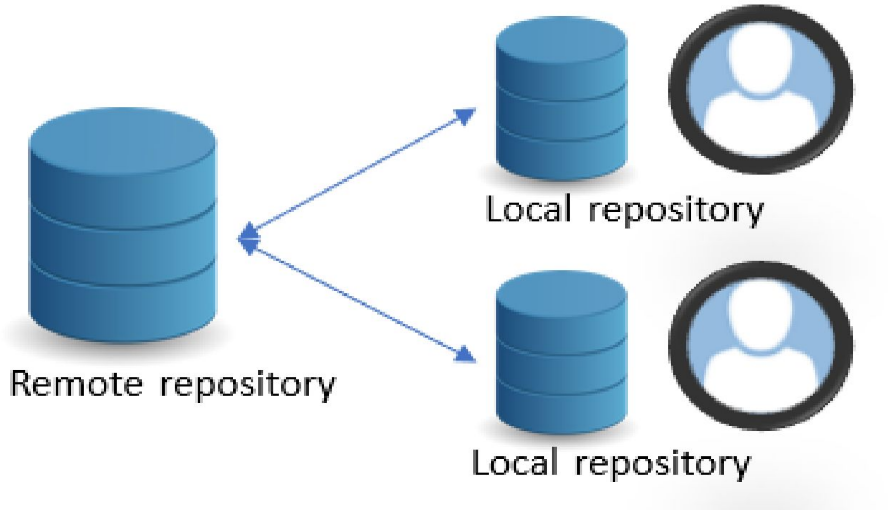
- The first type to emerge is the Centralized Version Control System, such as SVN, CVS, Subversion, and TFVC (or SourceSafe).
- These systems consist of a remote server that centralizes the code of all developers.

Overviewing Git and its command lines

- **Centralized Version Control System**



- **Distributed Version Control System**



Overviewing Git and its command lines

- **Centralized Version Control System**
- All developers can archive and retrieve their code on the remote server.
- The system allows better collaboration between teams and a guarantee of code backup.
- **Drawbacks:**
  - In case of no connection between the developers and the remote server, no more archiving or code recovery actions can be performed.
  - If the remote server no longer works, the code, as well as the history, will be lost.

Overviewing Git and its command lines

- **Distributed Version Control System**
- The second type of CVS is a distributed system
- Ex: Git or Mercurial
- These systems consist of a remote repository and a local copy of this repository on each developer's local machine.
- Even in the event of disconnection from the remote repository, the developer can continue to work with the local repository, and synchronization will be done when the remote repository is accessible again.
- A copy of the code and its history is also present in the local repository.

Overviewing Git and its command lines

- The first type to emerge is the centralized systems, such as SVN, CVS, Subversion, and TFVC (or SourceSafe).

RAMAIAH
Institute of Technology

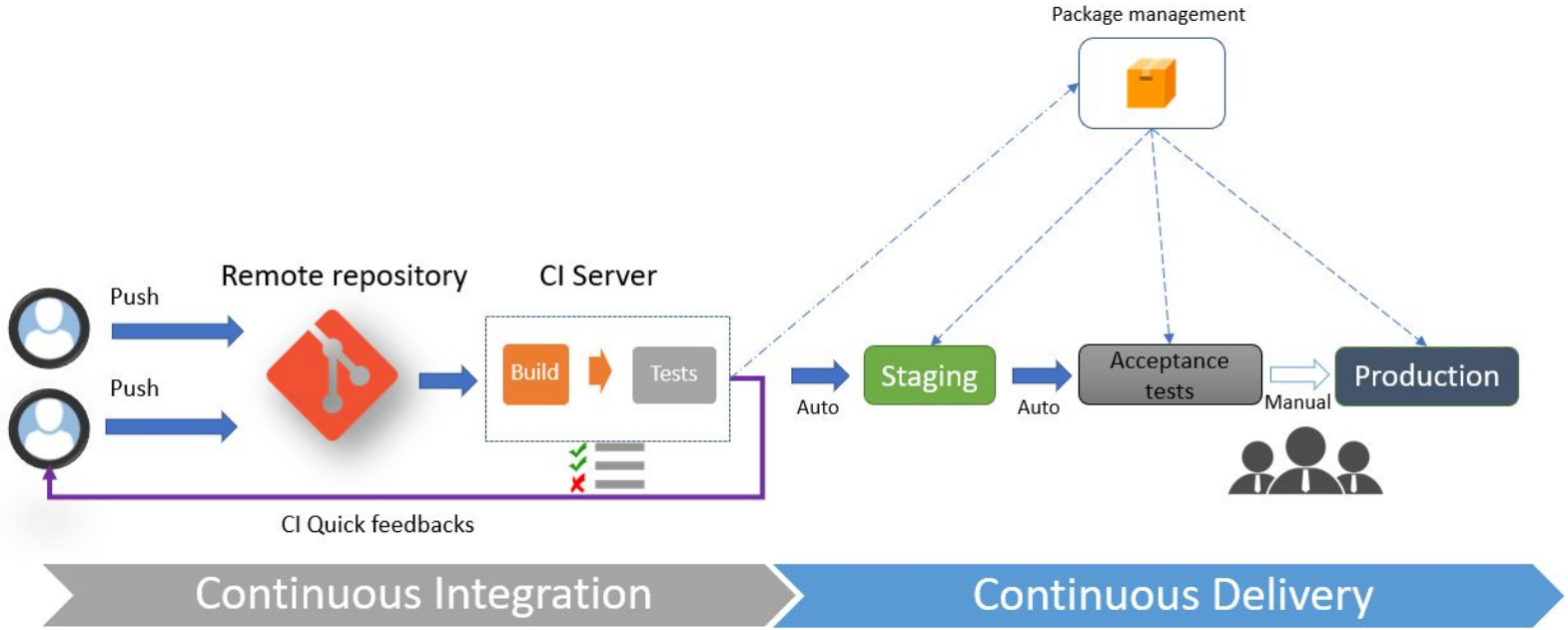
Continuous Integration and Continuous Delivery

Introduction

- One of the main pillars of DevOps culture is the implementation of continuous integration and deployment processes.
- **Continuous integration (CI)** is a process that provides rapid feedback on the consistency and quality of code to all members of a team.
- It occurs when each user's code commit retrieves and merges the code from a remote repository, compiles it, and tests it.
- **Continuous delivery (CD)** is the automation of the process that deploys an application in different stages (or environments).

The CI/CD Principles

- To implement a CI/CD pipeline, it is important to know the different elements that will be required to build an efficient and safe pipeline.



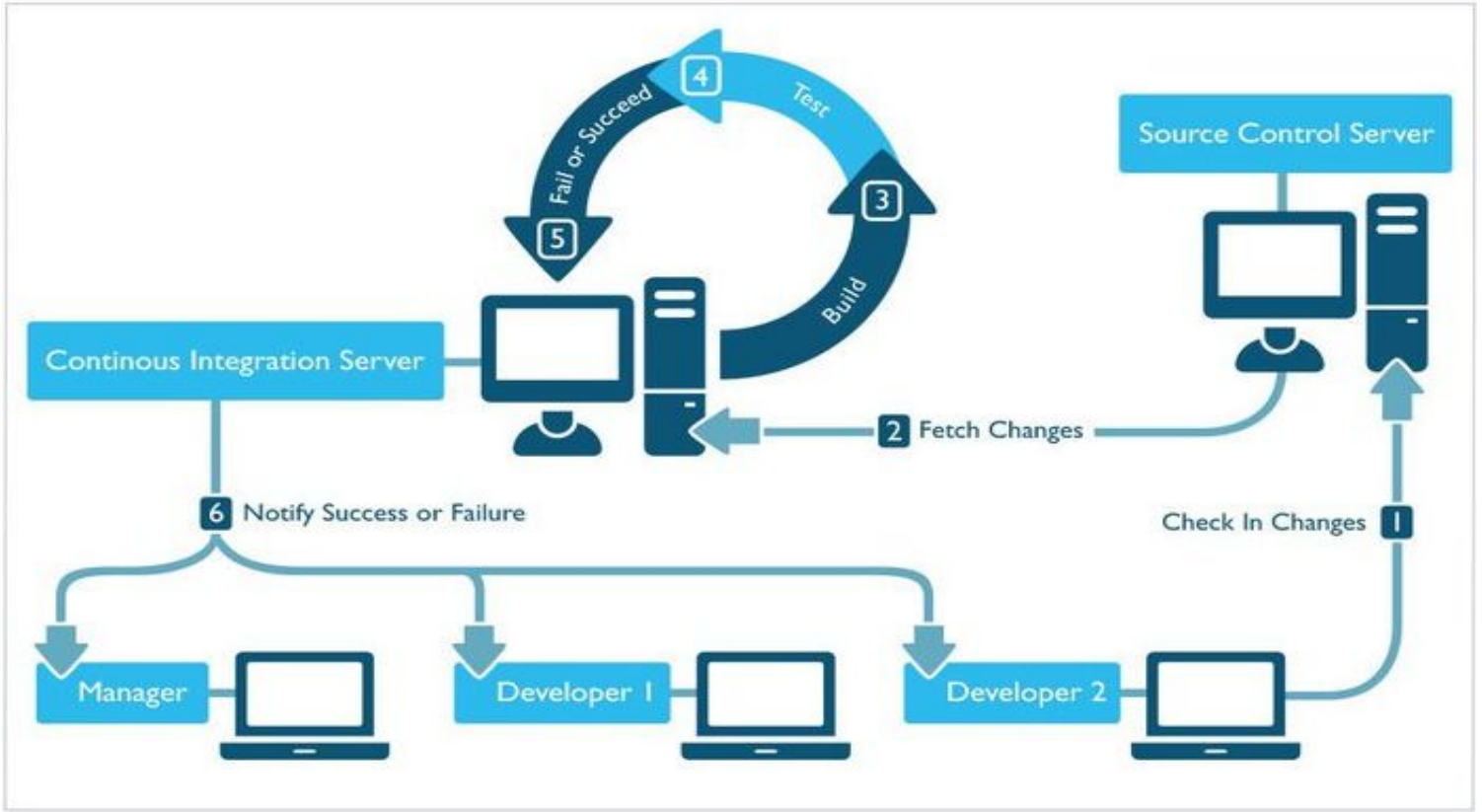
Continuous Integration (CI)

- The CI phase **checks the code archived by the team** members.
- Must be **executed on each commit** that has been pushed to the remote repository.
- The setting up of a **Git-type SCV** is necessary to centralize the code of all the members of a team.
- The team will have to decide on a code branch that will be used for continuous integration.
- Ex: **Master branch**, or the **Develop branch** as part of GitFlow.
- An **active branch** that very regularly **centralizes code changes**.

Continuous Integration (CI)

- CI is achieved by an **automatic task suite** that is **executed on a server**, following similar patterns executed on a developer's laptop that has the necessary tools for continuous integration; this server is called the CI server.
- CI servers (also known as build servers) automatically **compile, build, and test** every new version of **code committed to the central team repository**.
- CI server ensures that the **entire team is alerted any time the central code repository contains broken code**.

Continuous Integration (CI)



Continuous Integration (CI)

- The CI server can be
    1. **On-premise type**, installed in the company data center such as **Jenkins or TeamCity**
    2. **Cloud type** such as **Azure Pipelines or GitLab CI**.
- The tasks performed during the CI phase must be automated and take into account all the elements that are necessary for the verification of the code.
- The tasks performed during the CI phase are generally the **compilation of code** and the **execution of unit tests with code coverage**.

Continuous Integration (CI)

- At the end of the verification tasks, the CI generates an application package that will be deployed on the different environments (also called stages).
- To be able to host this package, we need a package manager, also called a repository manager
- Package manager can be on-premise (installed locally) such as Nexus, Artifactory or ProGet, or a SaaS Solution such as Azure Pipelines, Azure Artifacts, or GitHub Package Registry.
- This package must also be neutral in terms of environment configuration and must be versioned in order to deploy the application in a previous version if necessary.

Continuous Delivery (CD)

- Once the application has been packaged and stored in a package manager during CI, the **Continuous Delivery process is ready to retrieve the package and deploy it in different environments**.
- The deployment in each environment consists of a succession of automated tasks that are also executed on a remote server that has access to the different environments.
- **Necessary to involve Devs, Ops, and also the security team** in the implementation of CI/CD tools and processes.

Continuous Delivery (CD)

- This union of people with the tools and processes will **deploy applications on the different servers or cloud resources**, following the network rules & company's security standards.
- During the deployment phase, necessary to **modify the configuration of the application** in the generated package in order for it to be adapted to the target environment.
- Necessary to **integrate a configuration manager** that is already present in common CI/CD tools such as Jenkins, Azure Pipelines, or Octopus Deploy.

Continuous Delivery (CD)

- When there is a **new configuration key**, it is good practice for every environment, including production, to be entered with the involvement of the Ops team.
- Finally, the triggering of a **deployment can be done automatically**, but for some environments that are more critical (ex: production environments), heavily regulated companies may have gateways that require a **manual trigger with checks** on the people authorized to trigger the deployment.

Continuous Delivery (CD)

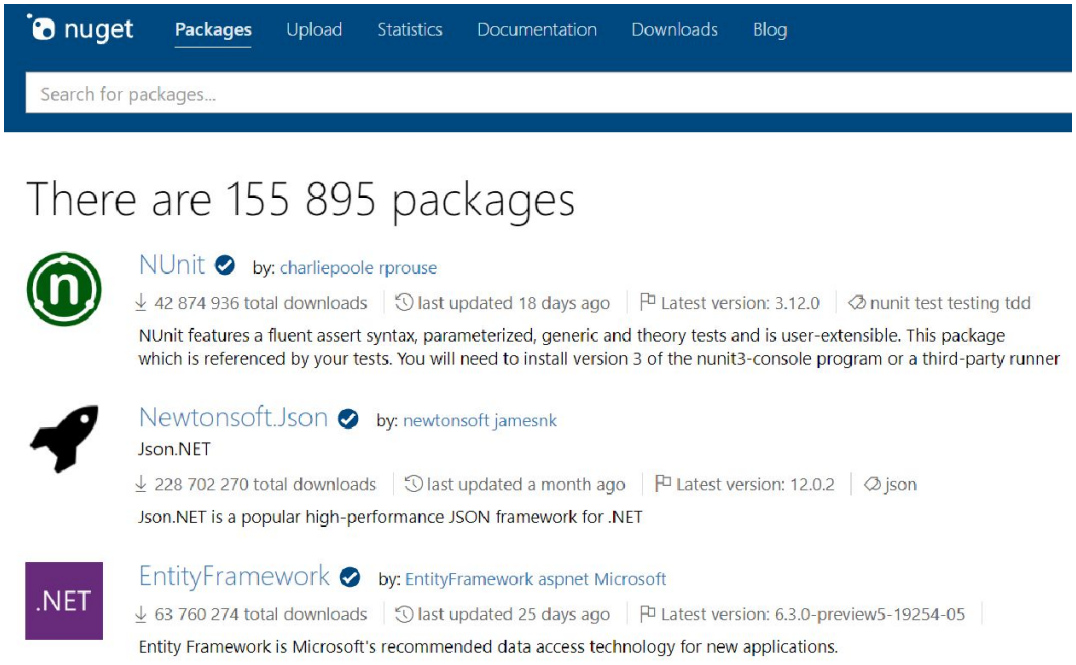
- The different tools for setting up a CI/CD pipeline are as follows:
    1. A source control version
    2. A package manager
    3. A CI server
    4. A configuration manager
- These tools will only be effective in delivering added value to the product if development and operations team work together around them.

Using a Package Manager

- Version Control System (VCS) or more commonly as a Version Control Manager (VCM)
  - The implementation of a CI/CD process can only be done with a VCS as a prerequisite.
  - Public package managers: **NuGet, npm, Maven, Bower, and Chocolatey**.
  - These package managers provide frameworks or tools for developers in different languages and platforms.

Using a Package Manager

- **NuGet package manager, which publicly provides more than 150K .NET Frameworks**
- Advantages:
  - Developer don't have to store the packages with the application sources.
  - Can add a reference in a configuration file, so the packages will be automatically retrieved.



Using a Package Manager

- In an enterprise application, although developers use packages from public managers, **some elements that are generated in an enterprise must remain internal**.
- **Frameworks** (such as NuGet or npm libraries) are **developed internally** and **cannot be exposed publicly**.
- In the CI/CD pipeline, we **need to make a package for our application** and **store it in a package manager** that will be **private to the company**.

Using a Package Manager

- **Private NuGet and npm repository:**
  - Create your own local repository to centralize your NuGet or npm packages.
  - For npm, install it locally with the npm local-npm package.
  - The problem with installing one repository per package type method is that we need to install and maintain a repository and its infrastructure for the different types of packages.

Using a Package Manager

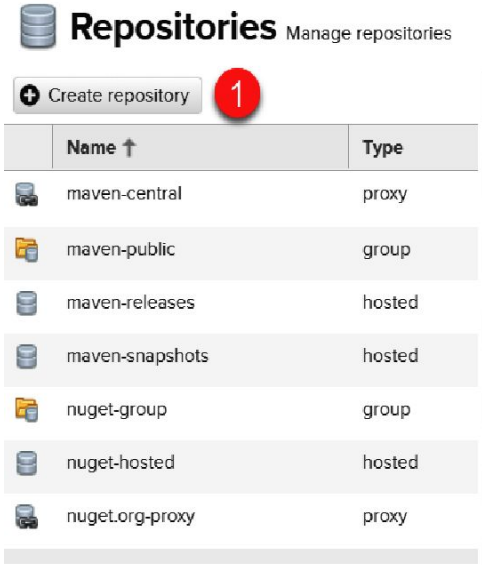
- **Nexus Repository OSS :**

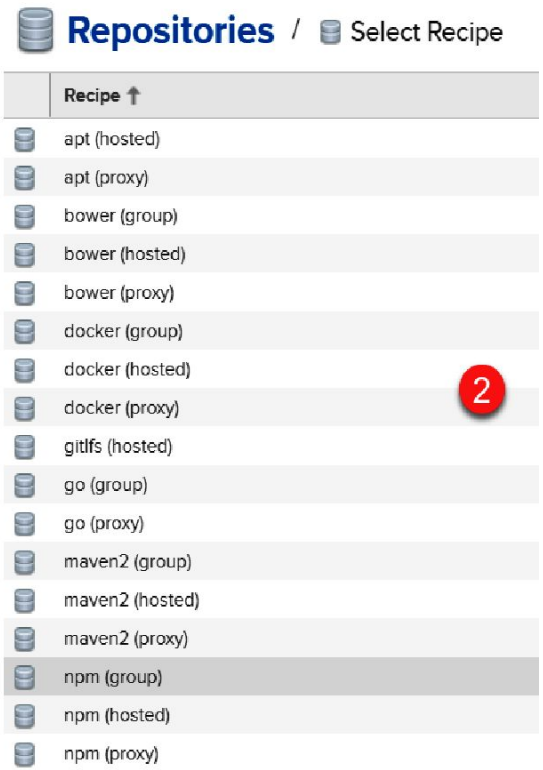
- Nexus Repository is a product of the Sonatype company, which specializes in DevSecOps tools that integrate security controls in the code of applications.
- Nexus Repository exists in an open source/free version.
- Nexus is a high-performance and widely used enterprise repository, but it requires effort to install and maintain it.
- Once Nexus Repository is installed, we must create a repository by following these steps:

Using a Package Manager

- **Nexus Repository OSS :**

1. In the Repositories section, click on the Create repository button.
2. Then choose the type of packages (for example, npm, NuGet, or Bower) that will be stored in the repository, as shown in the following screenshot:





Using a Package Manager

- **Azure Artifacts :**
- Azure Artifacts is one of the services provided by Azure DevOps.
- It is hosted in the cloud, and therefore, allows managing private package feeds.
- The packages supported today are NuGet, npm, Maven, Gradle, Python, and also universal packages.
- The main difference from Nexus is that in Azure Artifacts, the feed is not by package type.
- And one feed can contain different types of packages.

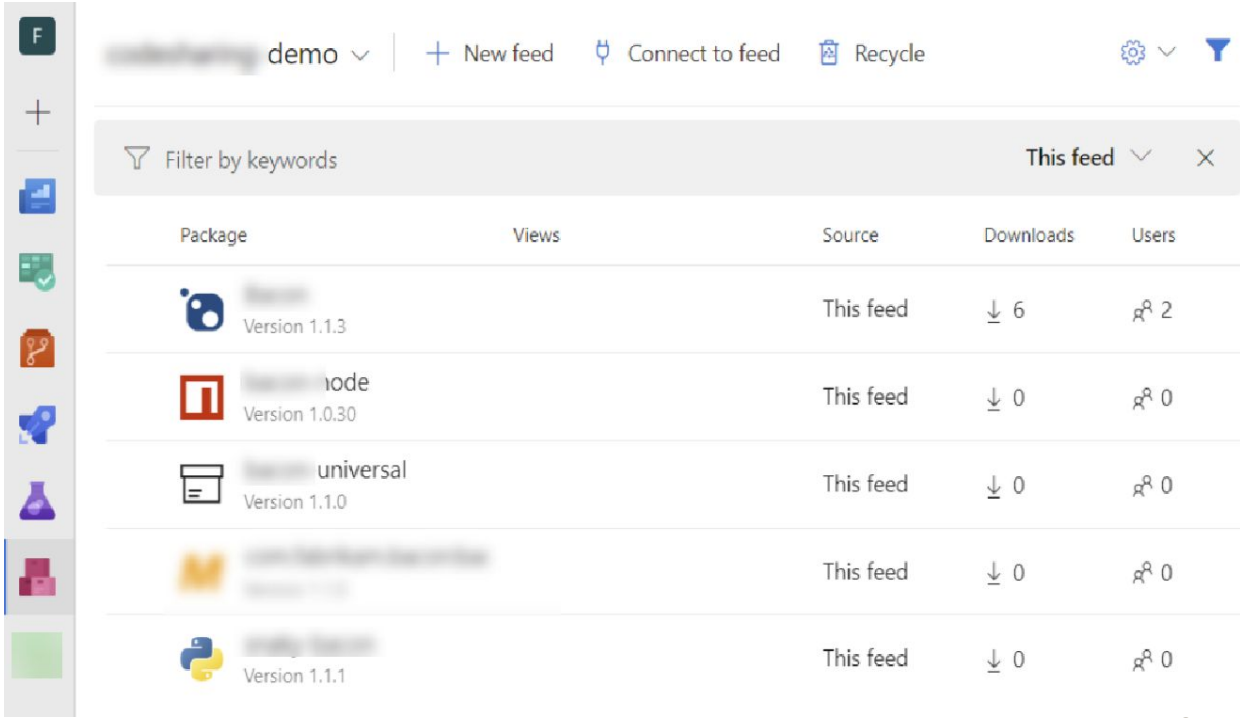
Using a Package Manager

- **Azure Artifacts :**
- **Advantages**:
- It is fully integrated with other Azure DevOps services such as Azure Pipelines, which allows managing CI/CD pipelines.
- In Azure Artifacts, there is also a type of package called universal packages that allows storing all types of files (called a package) in a feed that can be consumed by other services or users.
- Azure Artifacts is in SaaS offering mode, so there is no installation or infrastructure to manage.

Using a Package Manager

- **Azure Artifacts :**
- Azure Artifacts feed containing

1. NuGet package
2. npm package
3. Universal package
4. Maven package
5. Python package



Using Jenkins

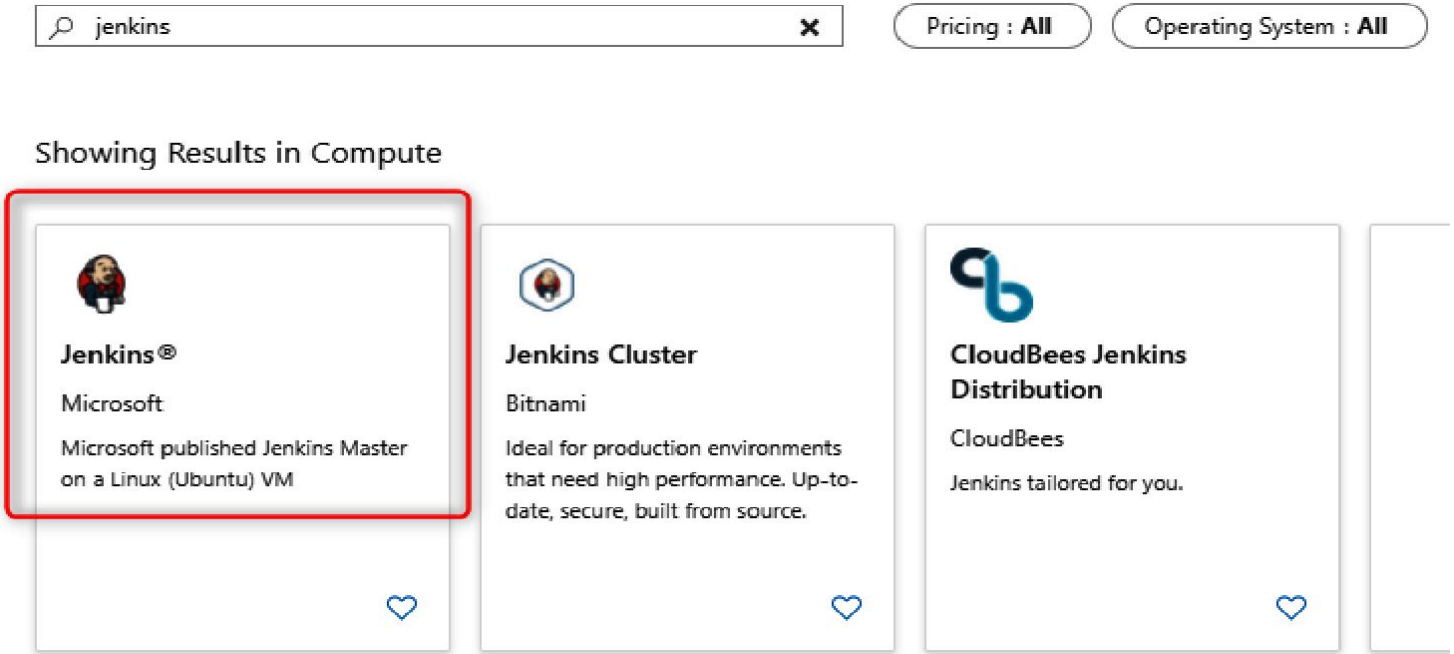
- Jenkins is one of the oldest continuous integration tools, initially released in 2011.
- It is open source and developed in Java.
- Jenkins has become famous due to the large community working on it and its plugins.
- More than 1,500 Jenkins plugins.
- If one of your tasks does not have a plugin, then you can create it yourself.

Installing and configuring Jenkins

- Jenkins is a cross-platform tool that can be installed on any type of support, such as VMs or even Docker containers.
- Azure Marketplace already contains a VM with Jenkins and its prerequisites already installed.
- Steps to create an Azure VM with Jenkins and its basic configuration:
- 1. To get all the steps to create an Azure VM with Jenkins already installed, read the documentation available here: https:/ / docs. microsoft. com/ en-us/ azure/jenkins/ install- jenkins- solution- template.

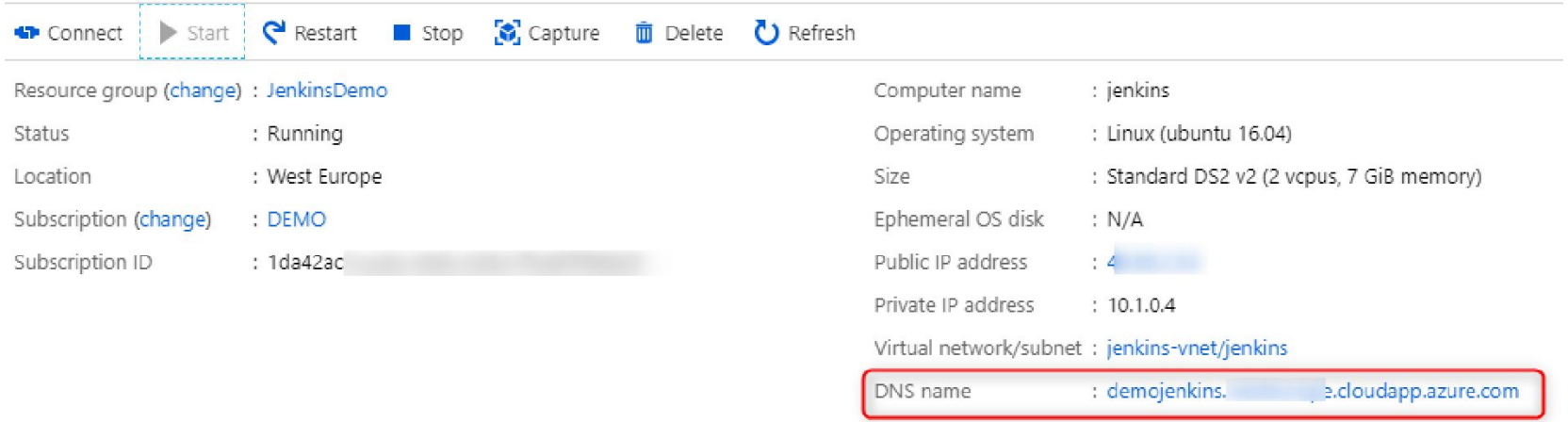
Installing and configuring Jenkins

- The following screenshot shows Jenkins integration on Azure Marketplace:



Installing and configuring Jenkins

- 2. Once installed and created, we will access it in the browser by providing its URL in the Azure portal in the DNS name field, as shown in the following screenshot:

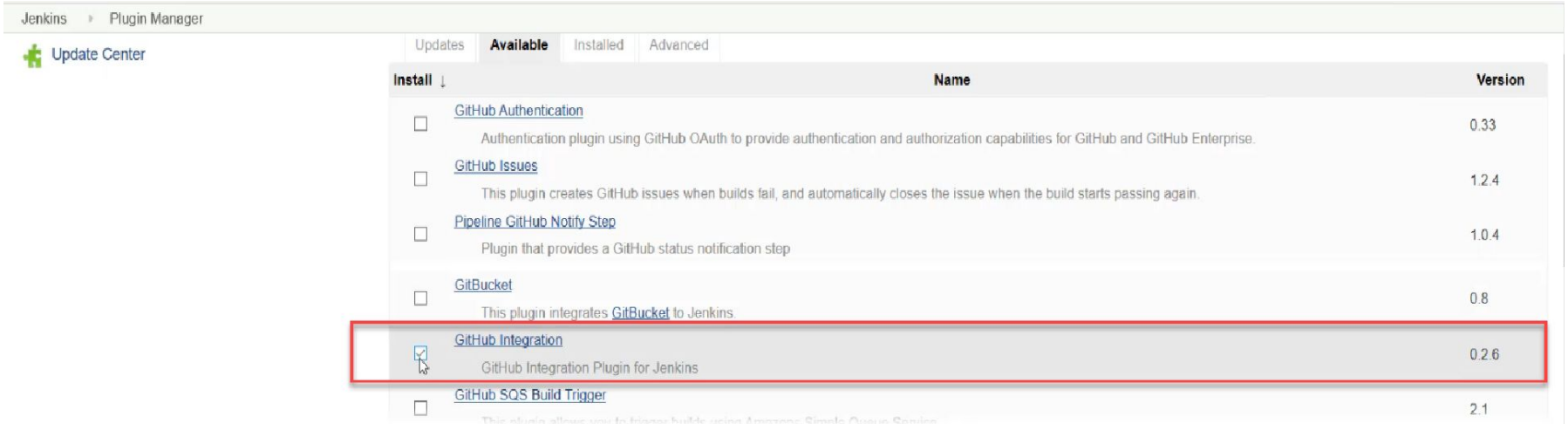


Installing and configuring Jenkins

- 3. Follow the displayed instructions on the Jenkins home page to enable access to this Jenkins instance via secure SSL tunneling.
- 4. Then, follow the configuration instructions on the Unlock Jenkins displayed on the Jenkins screen. Once the configuration is complete, we get Jenkins ready to create a CI job.
- In order to use GitHub features in Jenkins, install the GitHub integration plugin from the Jenkins plugin management.
- The following screenshot shows the installation of the GitHub plugin:

Installing and configuring Jenkins

- The following screenshot shows the installation of the GitHub plugin:



- Now configure GitHub with a webhook for its integration with Jenkins.

Configuring a GitHub webhook

- In order for Jenkins to run a new job, first create a webhook in the GitHub repository.
- This webhook will be used to notify Jenkins as soon as a new push occurs in the repository.
- To do this, follow these steps:

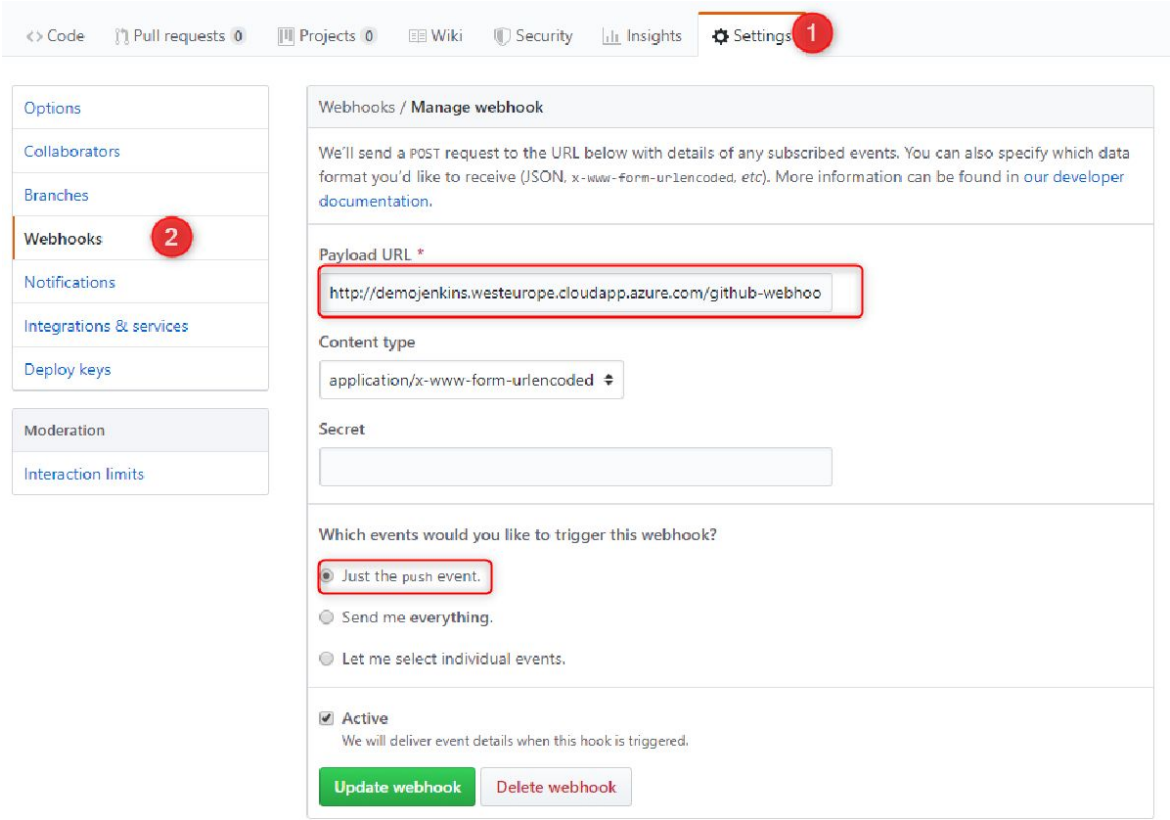
1. In the GitHub repository, go to the Settings | Webhooks menu.
2. Click on the Add Webhook button.
3. In the Payload URL field, fill in the URL address of Jenkins followed by /github-webhook/, leave the secret input as it is, and choose the Just the push event option.

Configuring a GitHub webhook

- To do this, follow these steps:

    4. Validate the webhook.

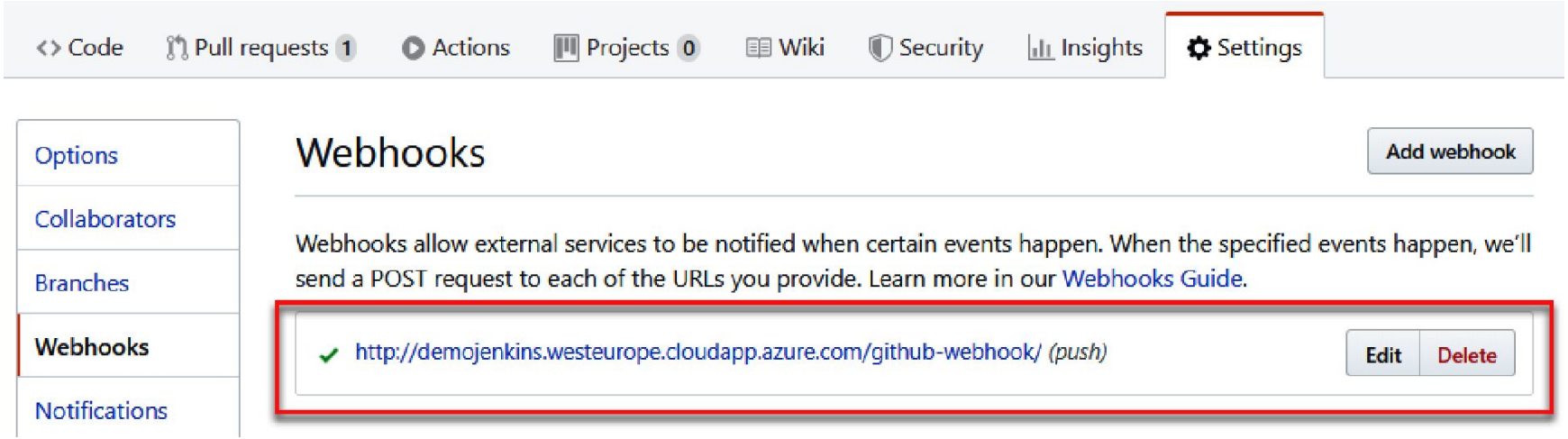
- Configuration of a GitHub webhook for Jenkins:



Configuring a GitHub webhook

- To do this, follow these steps:

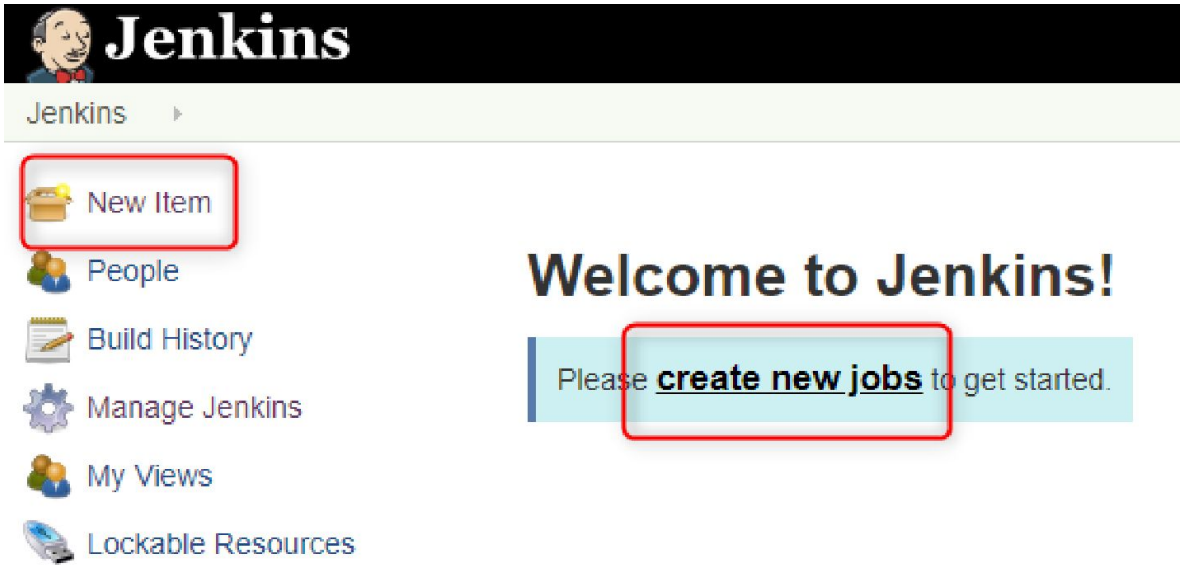
5. Finally, we can check on the GitHub interface, as shown in the following screenshot, that the webhook is well configured and that it communicates with Jenkins.



Configuring a Jenkins CI job

- To configure Jenkins, follow these steps:

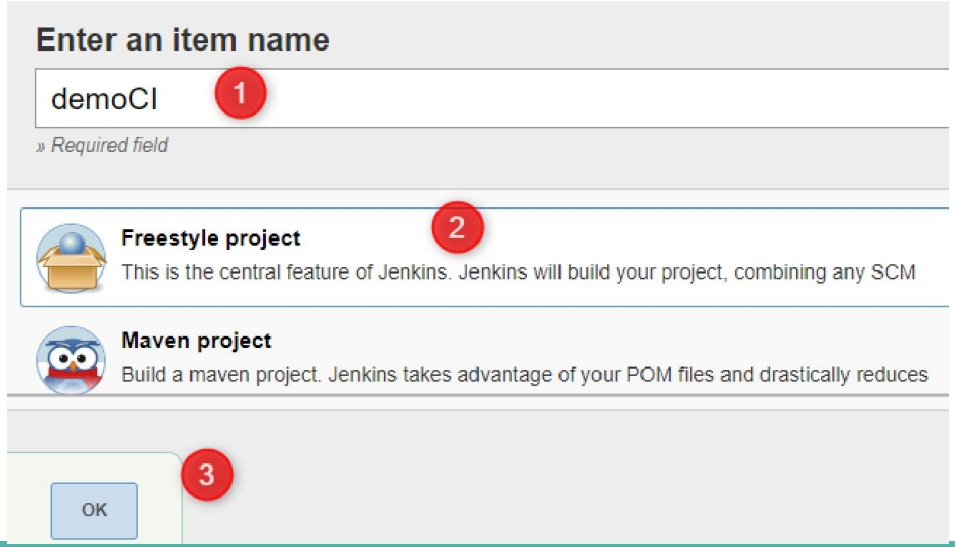
1. First, create a new job, by clicking on New Item or on the create new jobs link:



Configuring a Jenkins CI job

- To configure Jenkins, follow these steps:

2. On the job configuration form, enter the name of the job, and choose the Freestyle project template, then validate that by clicking on OK:

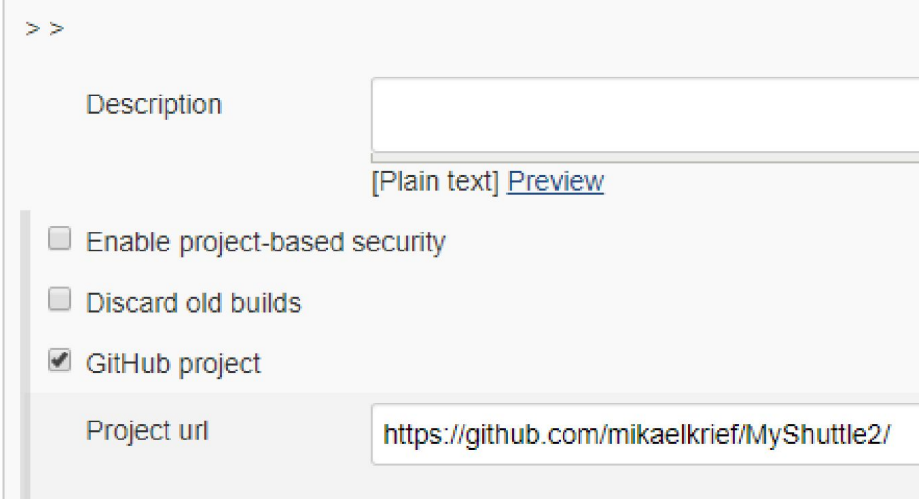


Configuring a Jenkins CI job

- To configure Jenkins, follow these steps:

3. Then, configure the job with the following parameters:

    - In the GitHub project input, enter the URL of the GitHub repository as follows:

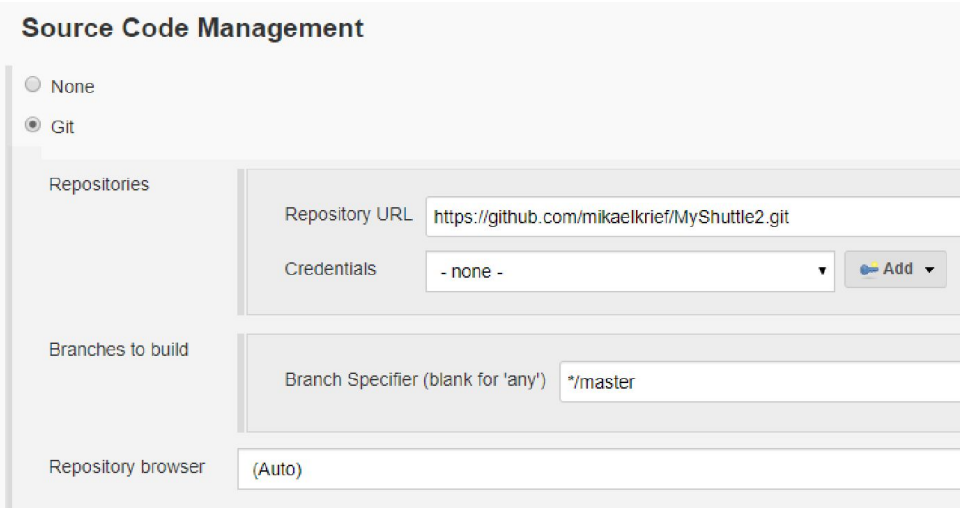


Configuring a Jenkins CI job

- To configure Jenkins, follow these steps:

3. Then, configure the job with the following parameters:

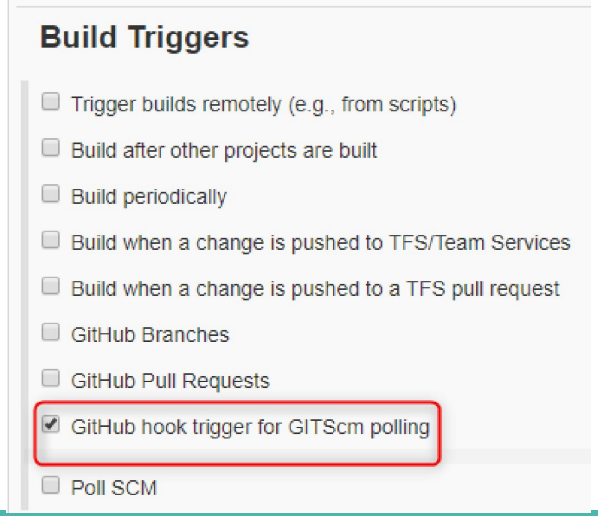
    - In the Source Code Management section, enter the URL of the GitHub repository and the code branch, like this:



Configuring a Jenkins CI job

- To configure Jenkins, follow these steps:

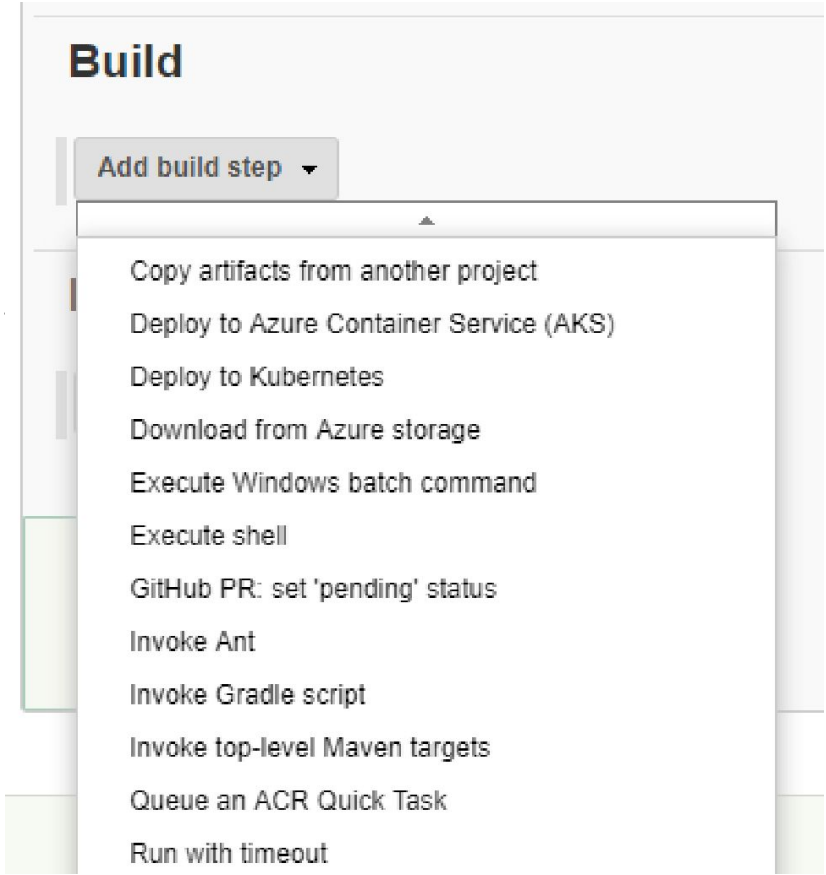
3. Then, configure the job with the following parameters:
    - In the Build Triggers section, check the GitHub hook trigger for GITScm polling box, like this:



Configuring a Jenkins CI job

- To configure Jenkins, follow these steps:

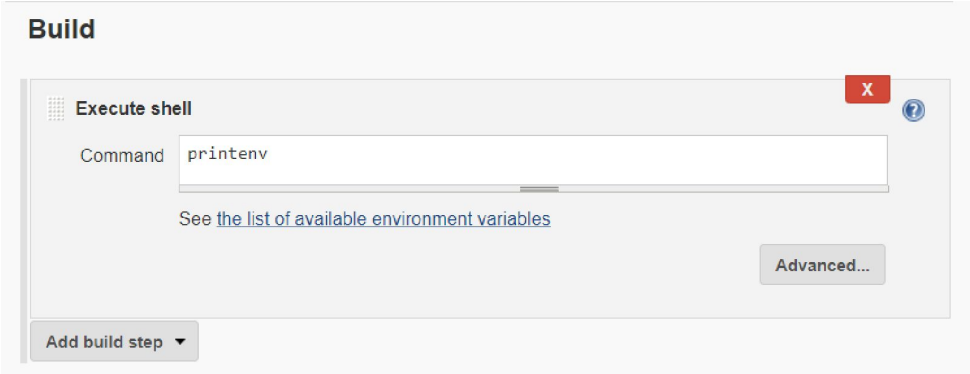
3. Then, configure the job with the parameters:
    - In the Build section, in the actions drop-down the Execute shell step.
    - You can add as many actions as necessary (compilation, file copies, and tests):



Configuring a Jenkins CI job

- To configure Jenkins, follow these steps:

3. Then, configure the job with the following parameters:
    - Inside the textbox of the shell command, enter the printenv command to be executed during the execution of the job pipeline:
4. Then, finish the configuration by clicking on Apply and then on the Save button.

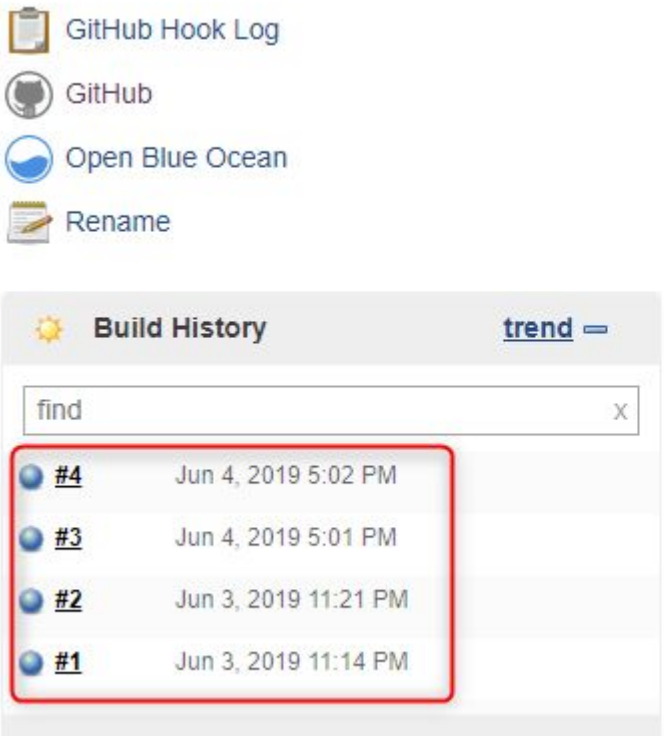


Executing the Jenkins job

- To test job execution, perform these steps:

1. Modify the code of GitHub repository, for example, by modifying the Readme.md file.
2. Then, commit to the master branch directly from the GitHub web interface.
3. After making this commit, the DemoCI job is queued up and running in Jenkins.
4. By clicking on the job, then on the link of the Console Output menu, see the job execution logs.

Executing the Jenkins job



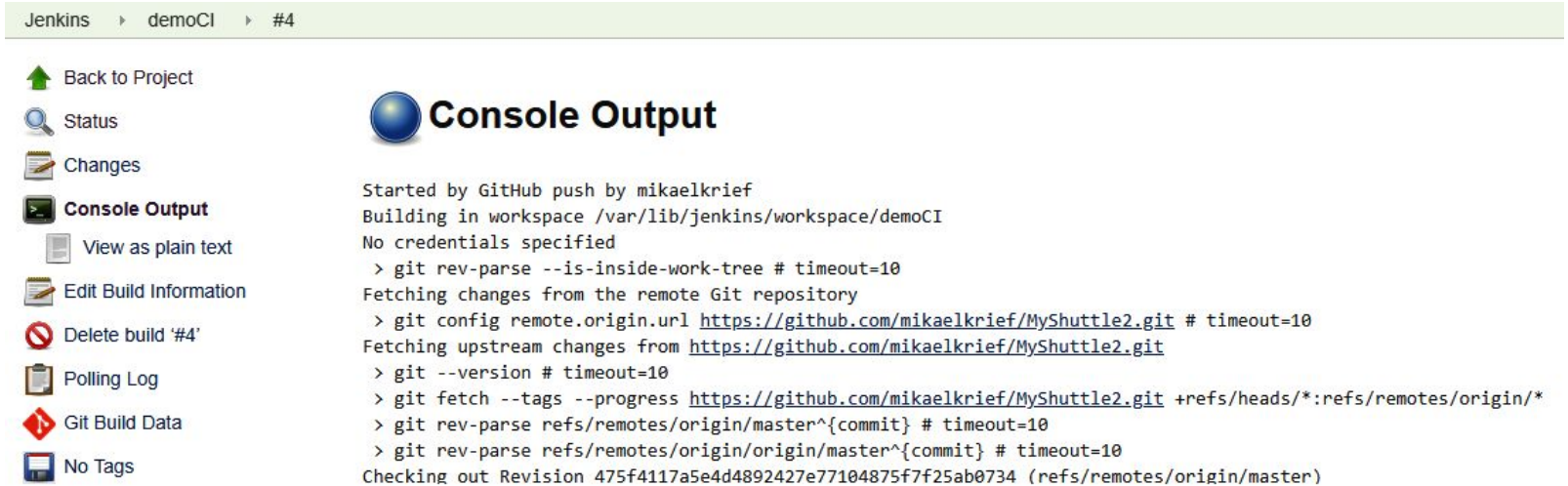
Permalinks

- Last build (#4), 18 sec ago
- Last stable build (#4), 18 sec ago
- Last successful build (#4), 18 sec ago
- Last completed build (#4), 18 sec ago

Executing the Jenkins job

- To test job execution, perform these steps:

4. By clicking on the job, then on the link of the Console Output menu, see the job execution logs.



Using Azure Pipelines

- Azure Pipelines is one of the services offered by Azure DevOps.
- Previously known as Visual Studio Team Services (VSTS).
- Azure DevOps is a complete DevOps platform that is fully accessible via a web browser and requires no installation.
- It is very useful for following reasons:
  - The DevOps tools manage its code in Version Control System (VCS)
  - It manages a project in agile mode
  - It deploys applications in a CI/CD pipeline, to centralize packages
  - It performs manual test plans

Using Azure Pipelines

Service Name	Description
Azure Repos	It is a Source Code Versioning System.
Azure Boards	It is a service for project management in agile mode with sprints, backlogs, and boards.
Azure Pipelines	It is a service that allows the management of CI/CD pipelines.
Azure Artifacts	It is a private package manager.
Azure Test Plans	It allows you to make and manage a manual test plan.

Using Azure Pipelines

- Azure DevOps is free for up to five users. Beyond that, there is a license version with per user costs.
- There is also Azure DevOps Server, which is the same product as Azure DevOps, but it installs itself on premises.
- To register with Azure DevOps and create an account, called an organization, we need either a Microsoft live account or a GitHub account, and to follow these steps:

1. In your browser, go to this URL: https://azure.microsoft.com/en-us/services/devops/

Using Azure Pipelines

2. Click on the Signup button.
3. On the next page, choose the account to use (either Live or GitHub).
4. As soon as we register, the first step suggested is to create an organization with a unique name of your choice and the Azure location, for example, BookLabs for the name of the organization, and West Europe for the location.
5. In this organization, we will now be able to create projects with our CI/CD pipeline.

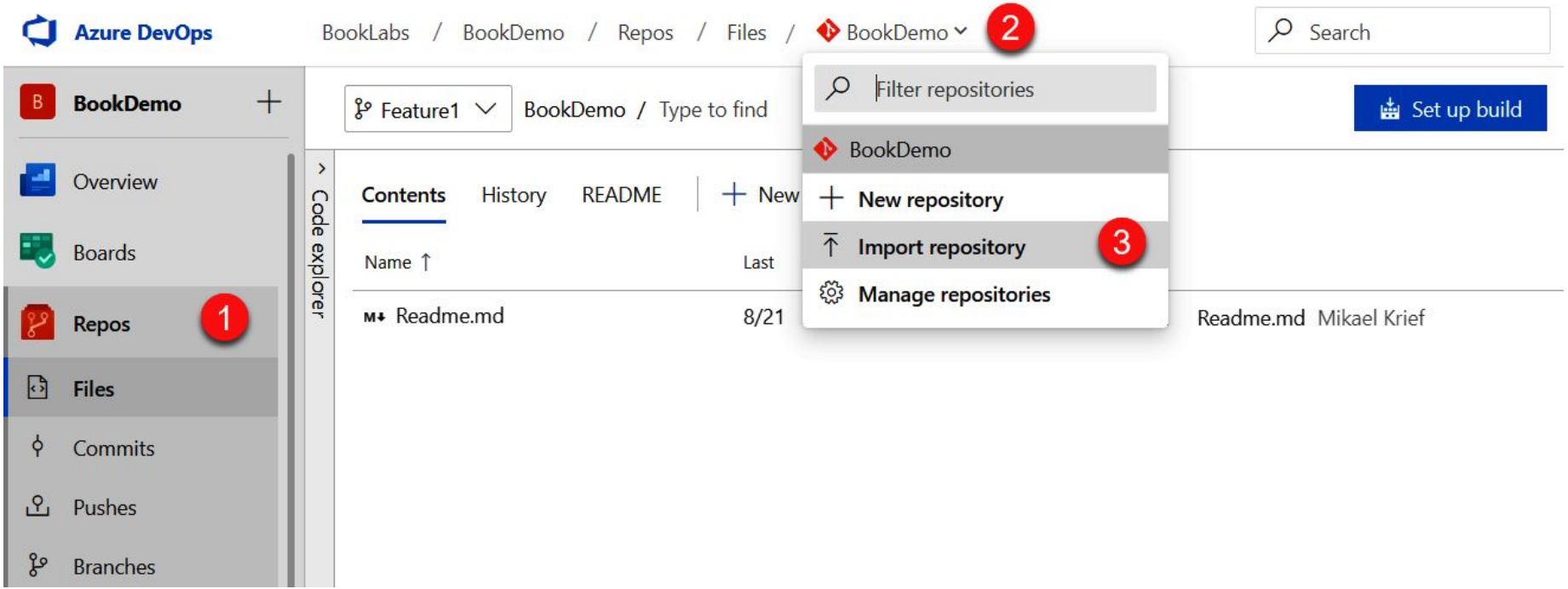
Using Azure Pipelines

- **Versioning of the code with Git in Azure Repos:**
- First prerequisite for setting up a continuous integration process is to have the application code versioned in an SVC, and do this in Azure Repos by following these steps:

1. To start the lab, create a new project.
2. Then, in Azure Repos, import code from another Git repository by using the Import repository option of the repository menu.
3. Once the Import a Git repository window opens, enter the URL of the Git repository whose sources we want to import.

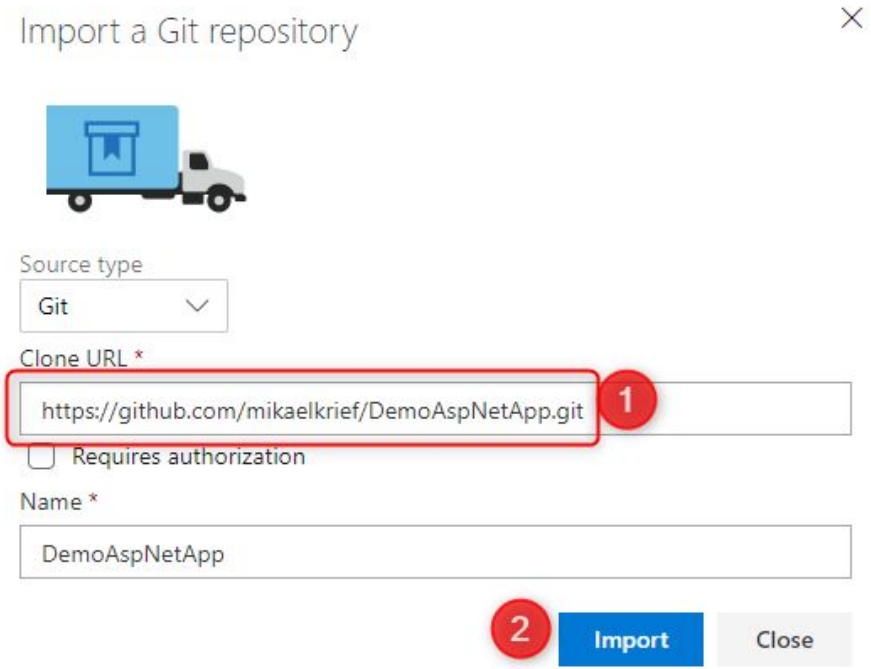
Using Azure Pipelines

- **Versioning of the code with Git in Azure Repos:**



Using Azure Pipelines

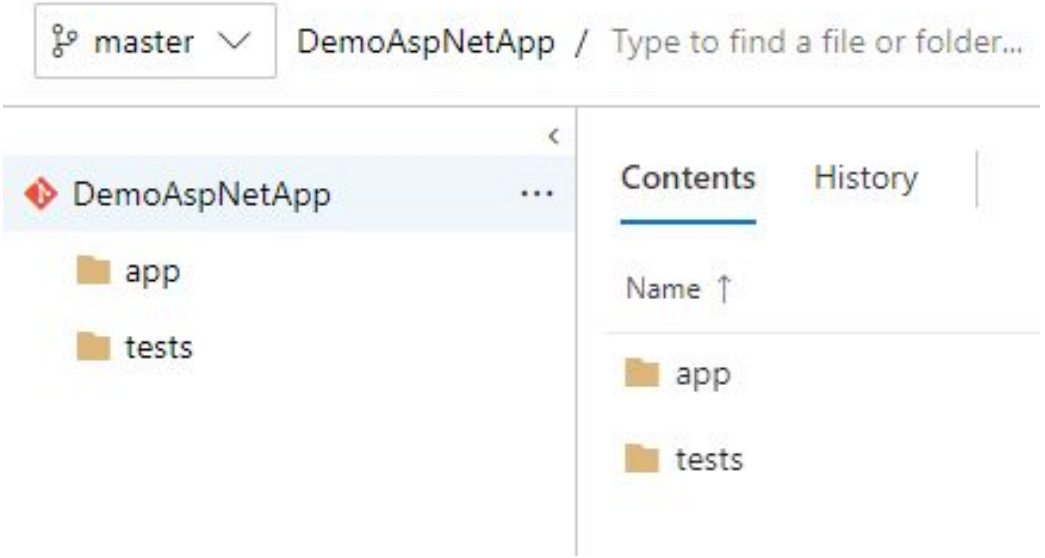
- **Versioning of the code with Git in Azure Repos:**



Using Azure Pipelines

- **Versioning of the code with Git in Azure Repos:**

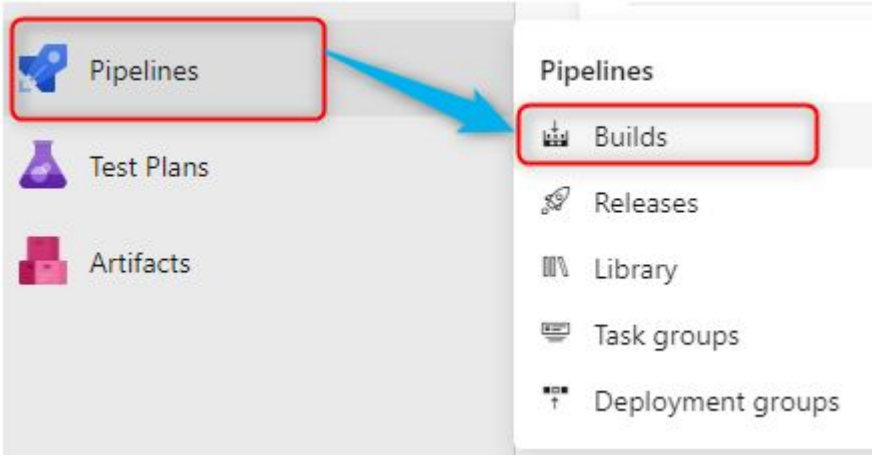
4. Click on the Import button, then see that the code is imported into the repository.



Using Azure Pipelines

- **Creating the CI pipeline:**
- Create a CI pipeline in Azure Pipelines by following these steps:

1. To create this pipeline, open the Pipelines | Builds menu.
2. Then, click on the New pipeline button.

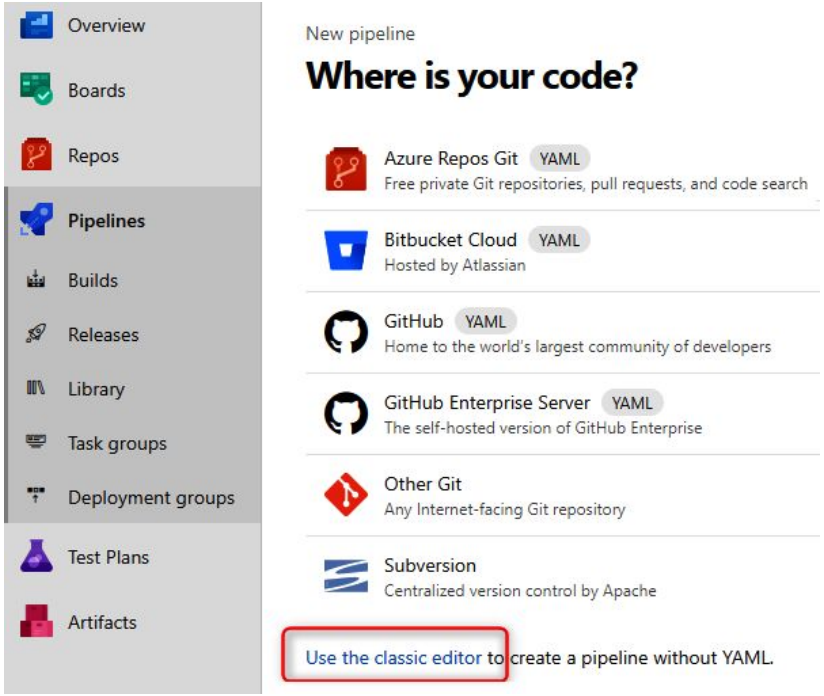


Using Azure Pipelines

- **Creating the CI pipeline:**

3. For the configuration mode, we choose the Use the classic editor option:

- In Azure Pipelines, there is the choice
  - The classic editor mode (configure via a graphical interface)
  - The YAML pipeline mode, which involves using a YAML file that describes the configuration of the pipeline.



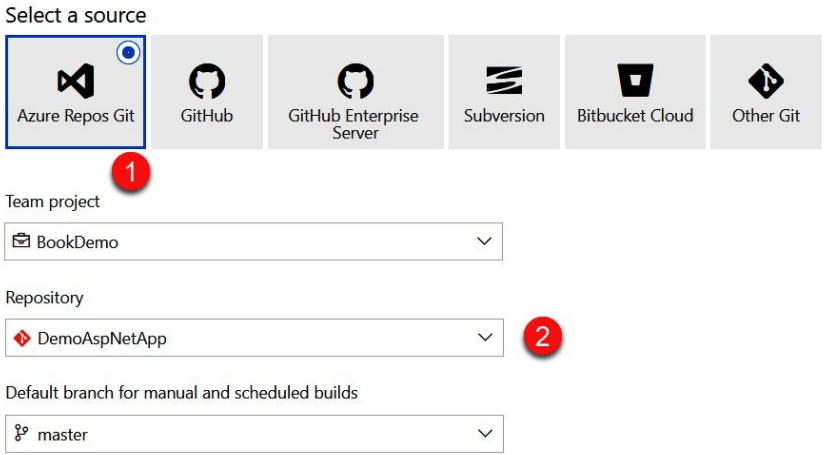
Using Azure Pipelines

- **Creating the CI pipeline:**

4. The first configuration step of the pipeline consists of selecting the repository that contains the application's sources.

- Azure Pipelines supports several types of Git, such as Azure Repos, GitHub, Bitbucket, Subversion etc.

- Select Azure Repos Git in repository that contains imported sources.



Using Azure Pipelines

- **Creating the CI pipeline:**

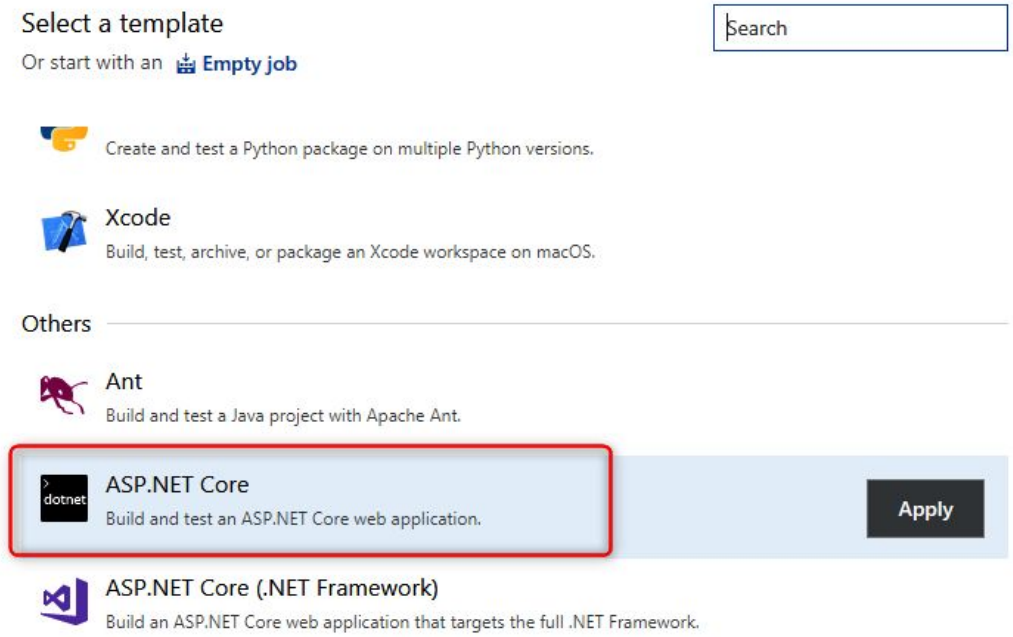
5. Azure Pipelines proposes to select a build template that will contain all the preconfigured build steps; there is also the possibility to start from an empty template.

- The configuration of the build definition consists of four main sections:
  - The variables
  - The steps
  - The triggers
  - The options

Using Azure Pipelines

- **Creating the CI pipeline:**

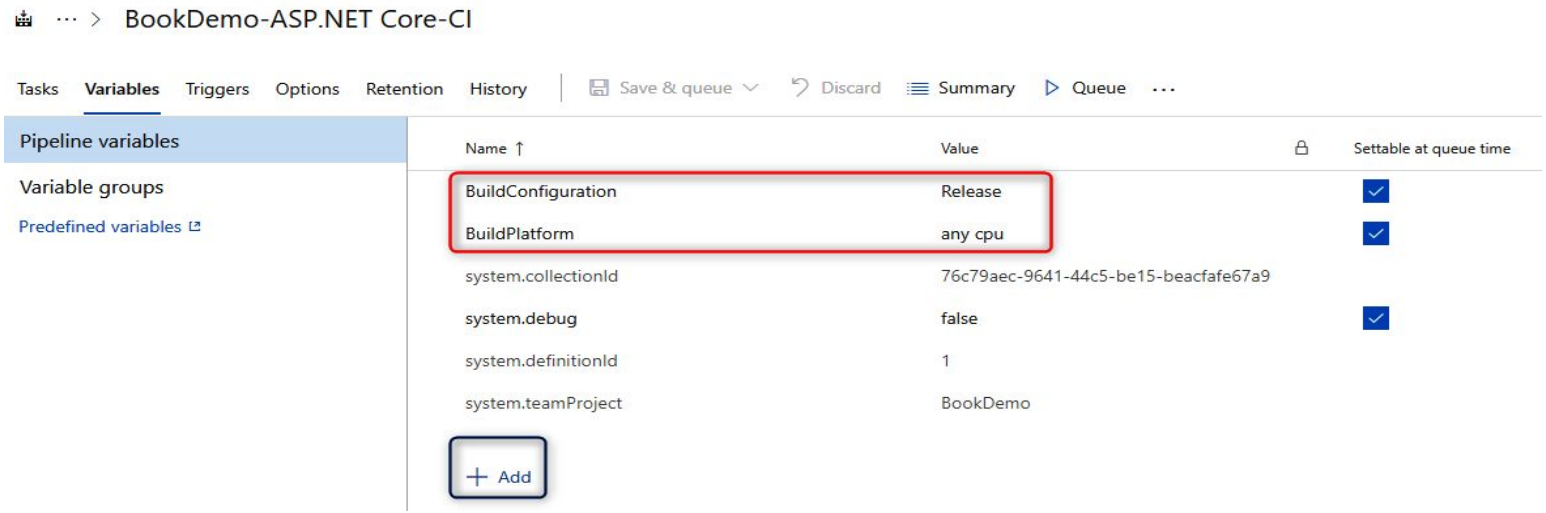
5. Azure Pipelines proposes to select a build template that will contain all the preconfigured build steps; there is also the possibility to start from an empty template.



Using Azure Pipelines

- **Creating the CI pipeline:**

6. Configure the Variables section, which allows to fill in a list of variables in a key form, creating a value that can be used in the steps.



Using Azure Pipelines

- **Creating the CI pipeline:**

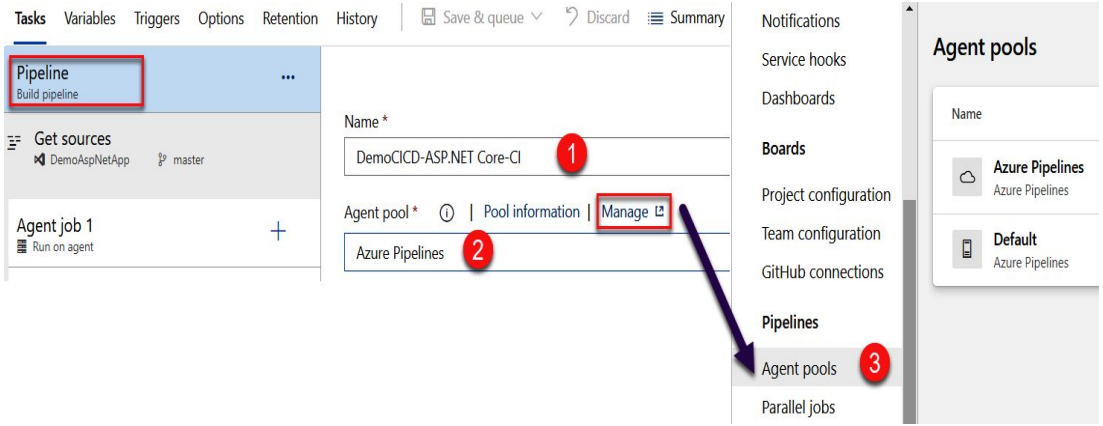
7. Configure the Tasks tab, which contains the configuration of all the steps to be performed in the build.

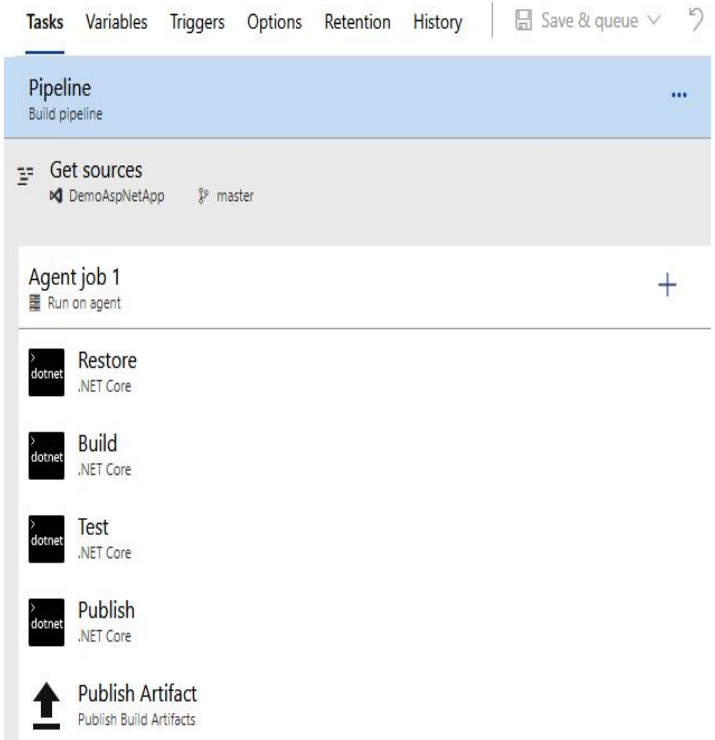
- In the preceding screenshot, the first part is Pipeline, which allows to configure the name of the build definition as well as the agent to use.
- In Azure DevOps, pipelines are executed on agents that are installed on VMs or containers.
- Azure DevOps offers free agents from multiple OSes, called a hosted agent, and also install your own agents, called self-hosted.

Using Azure Pipelines

- **Creating the CI pipeline:**

7. Configure the Tasks tab, which contains the configuration of all the steps to be performed in the build.

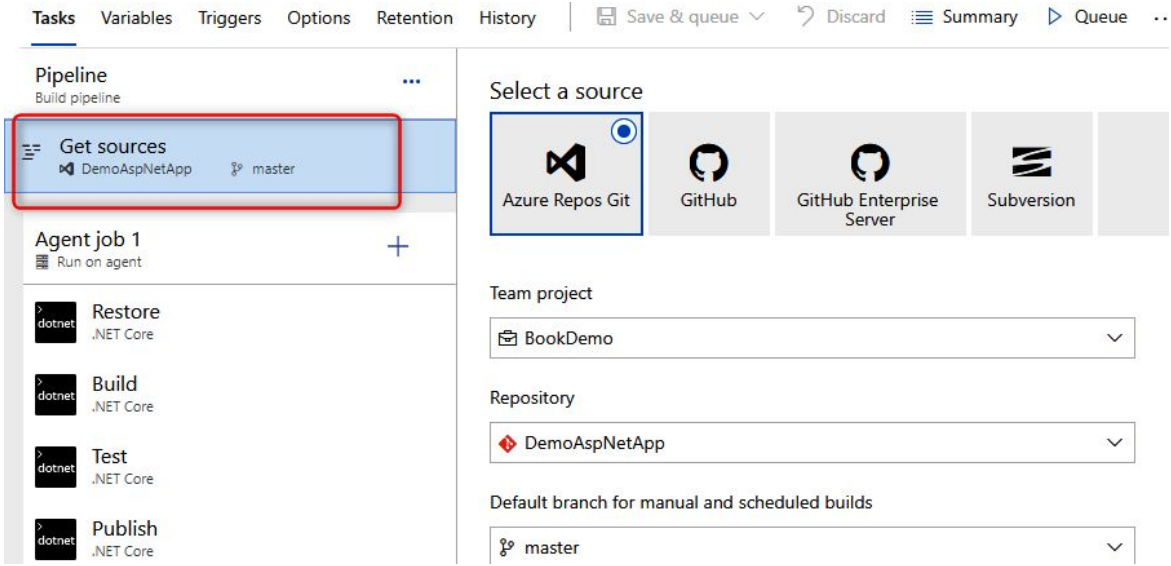




Using Azure Pipelines

- **Creating the CI pipeline:**

8. Next is Get sources phase, which contains the configuration of the sources that we did at the beginning; and can be modified here:

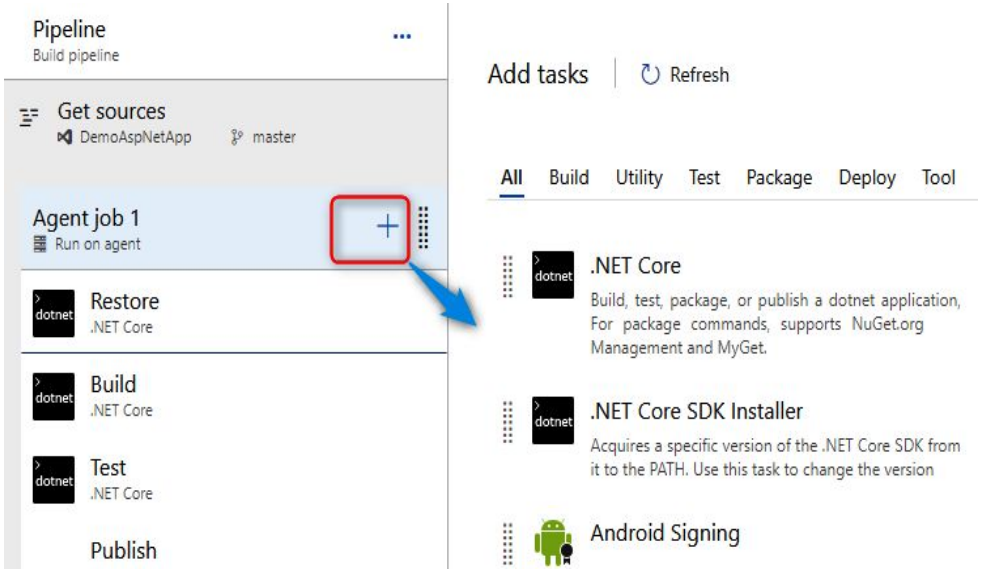


Using Azure Pipelines

- **Creating the CI pipeline:**

9. The Agent job part, which contains the ordered list of tasks to be performed in the pipeline. Each of these tasks is configured in panel on the right.

- We can add tasks by clicking on the + button, and select them from the Azure Pipelines catalog.



Using Azure Pipelines

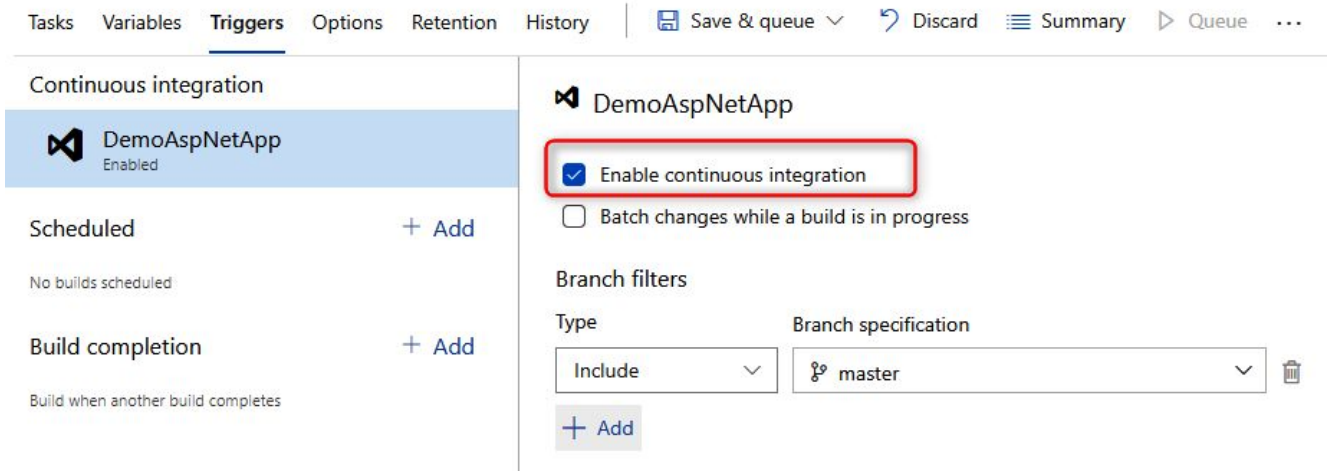
- **Creating the CI pipeline:**
- The five tasks in the CI pipeline:

Step/task	Description
Restore	Restores the packages referenced in the project.
Build	Builds the project and generates binaries.
Test	Runs unit tests.
Publish	Creates a ZIP package that contains the binary files of the project.
Publish Build Artifacts	Defines an artifact that is our ZIP of the application, which we will publish in Azure DevOps, and which will be used in the deployment release, as seen in the previous, *Use package manager*, section.

Using Azure Pipelines

- **Creating the CI pipeline:**

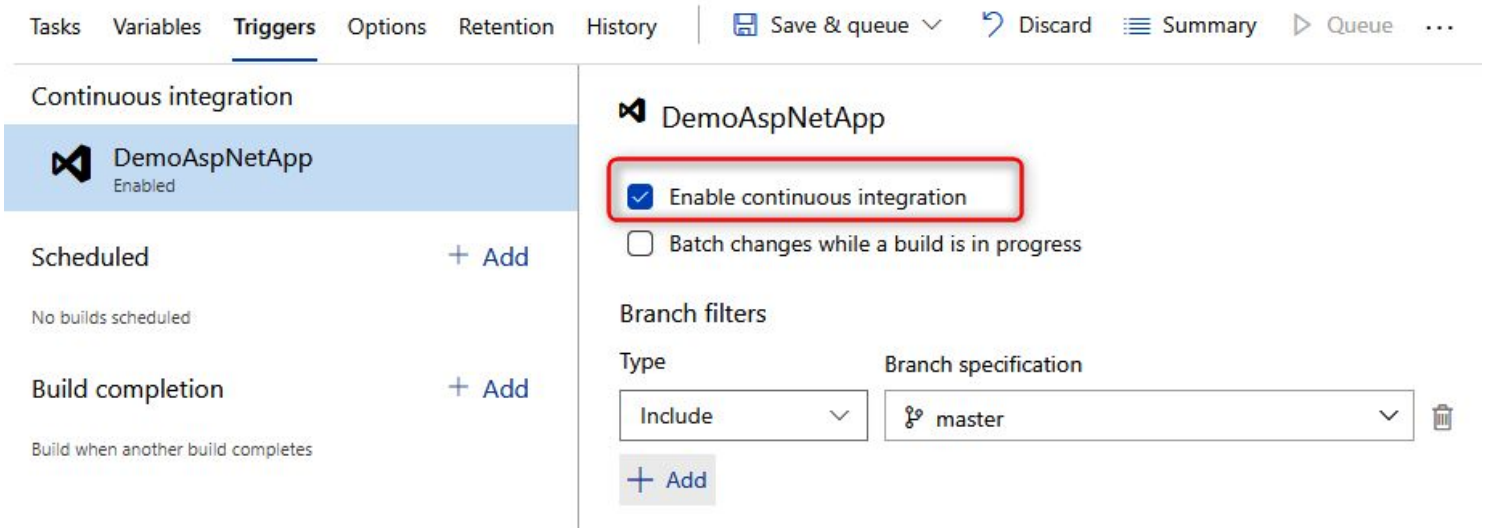
10. The last important configuration of our CI pipeline is the configuration of the build trigger in the Triggers tab to enable continuous integration, as shown in the following screenshot:



Using Azure Pipelines

- **Creating the CI pipeline:**

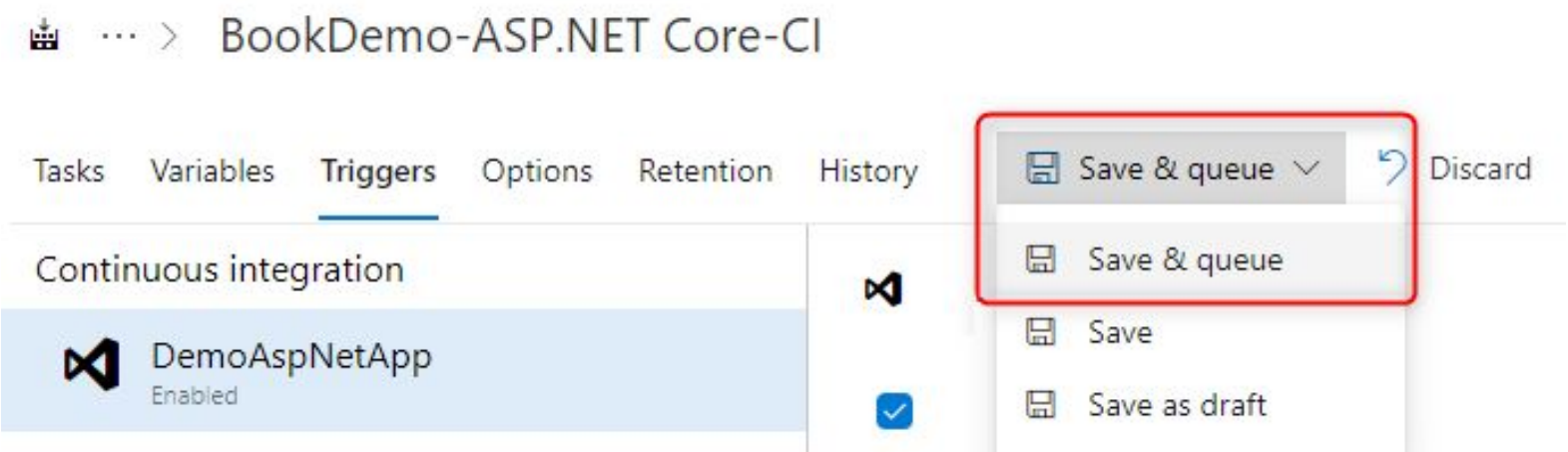
10. The last important configuration of CI pipeline is the configuration of the build trigger in the Triggers tab to enable continuous integration:



Using Azure Pipelines

- **Creating the CI pipeline:**

11. The configuration of CI or Build pipeline is complete; validate and test its execution for the first time by clicking on the Save & queue button:

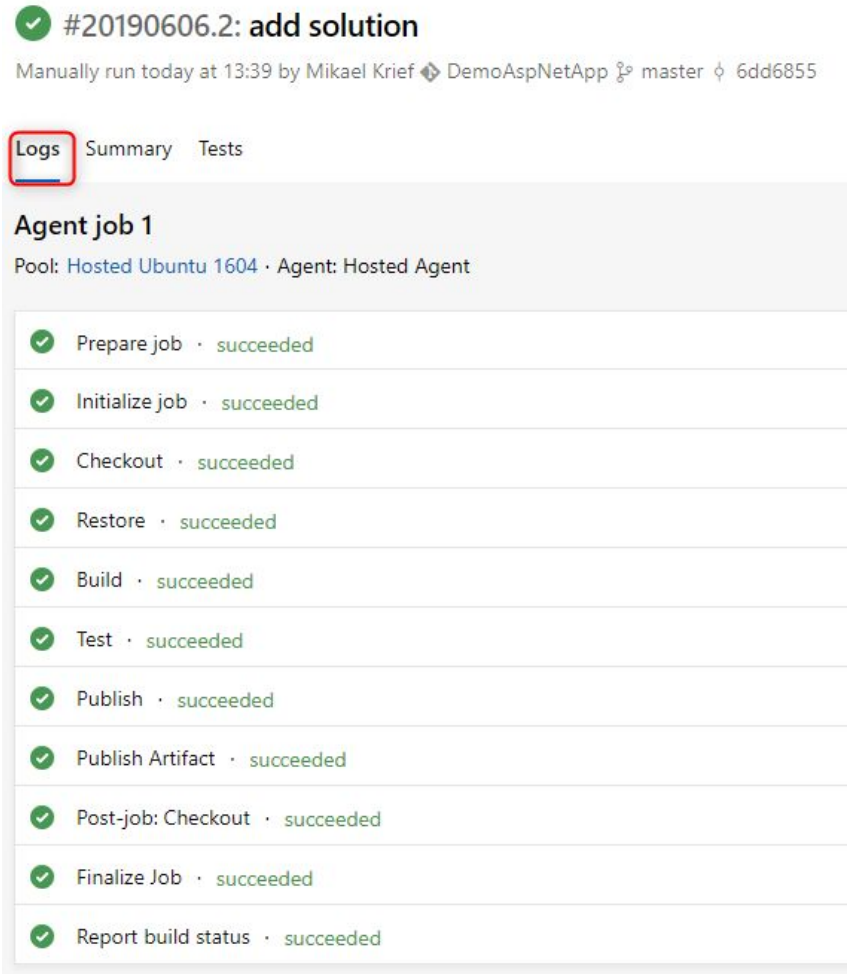


Using Azure Pipelines

- **Creating the CI pipeline:**

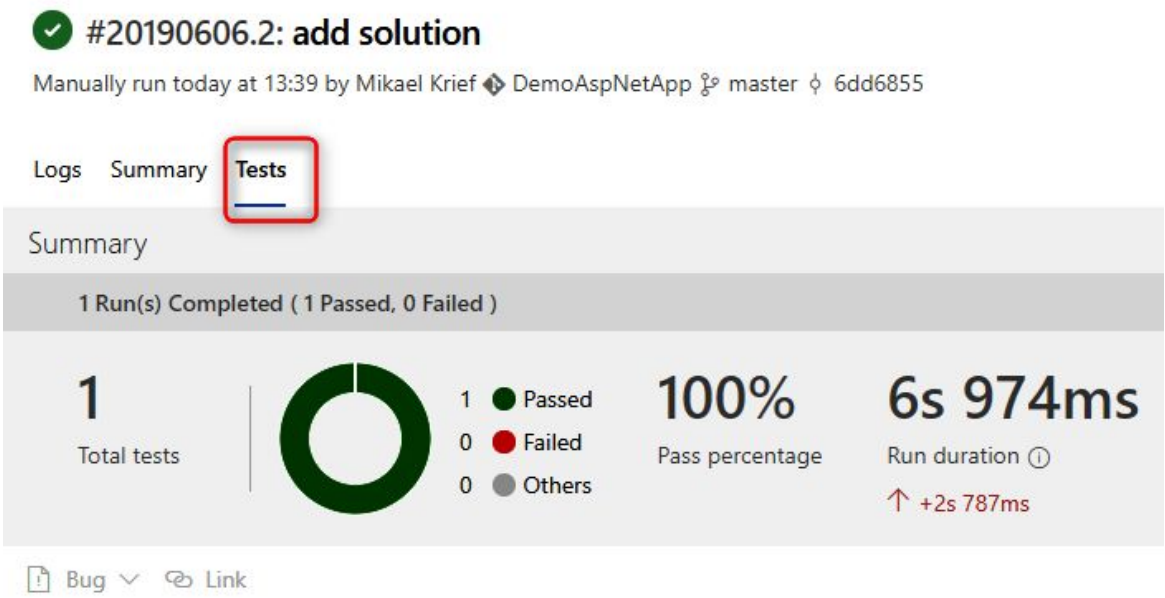
12. At the end of the execution of the build, some information, which helps to analyze the status of the pipeline:

- The following screenshot shows the execution logs:



Using Azure Pipelines

- **Creating the CI pipeline:**
- This displays the details of the execution of each task defined in the pipeline.

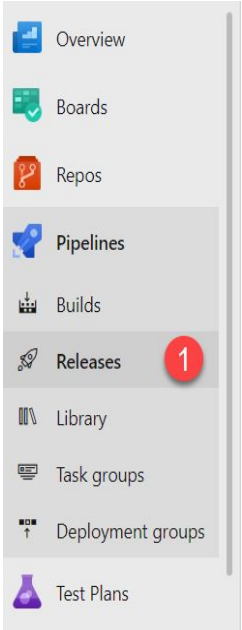


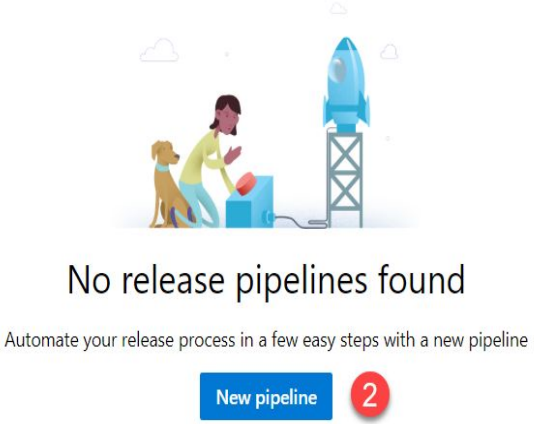
Creating the CD Pipeline: The Release

- In Azure Pipelines, the element that allows deployment in the different stages or environments is called the release.

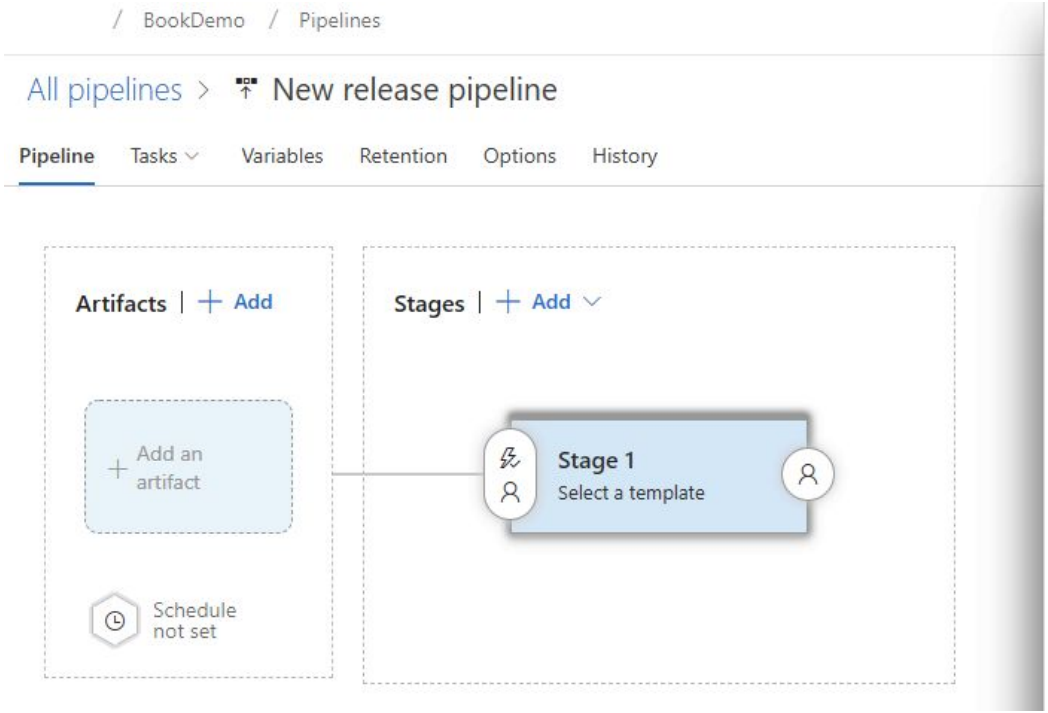
1. To create the release definition, go to the Releases menu and click on New pipeline, as follows:

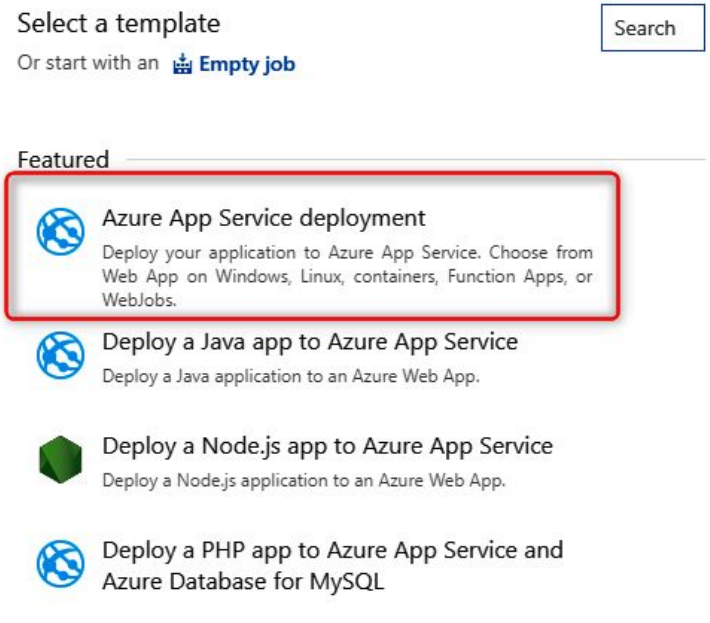
2. As for the build, the first step of the configuration is to choose a template already configured. For this lab choose the Azure App Service deployment template:





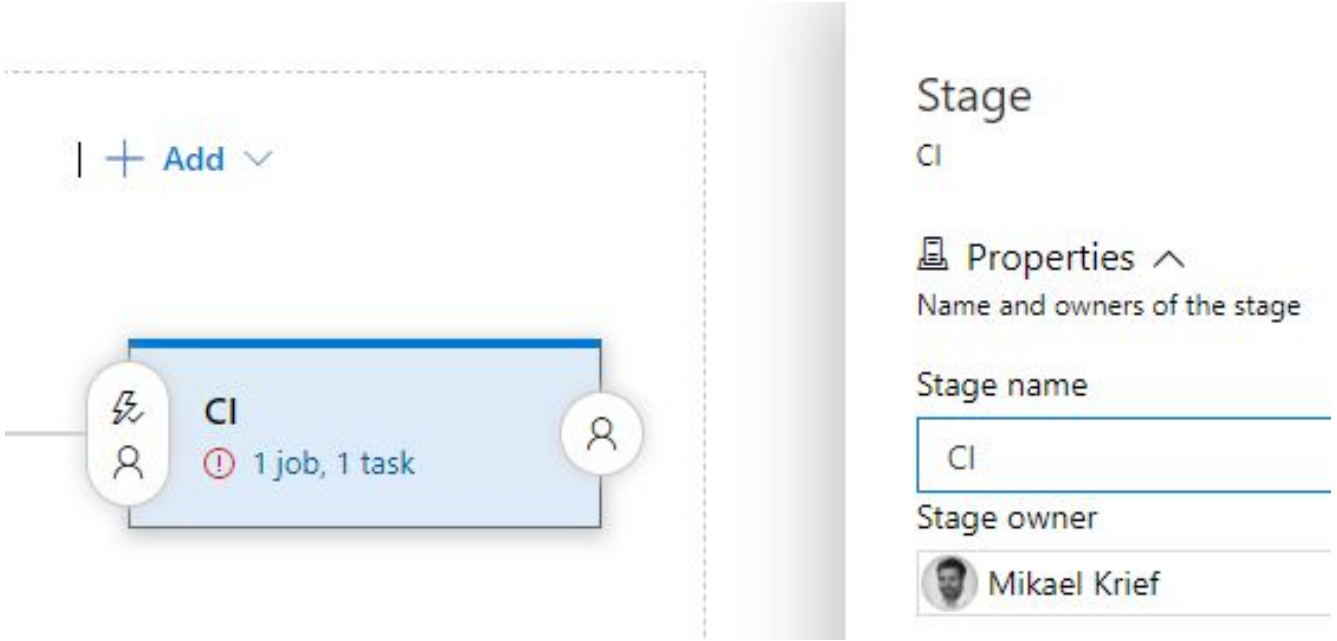
Creating the CD Pipeline: The Release





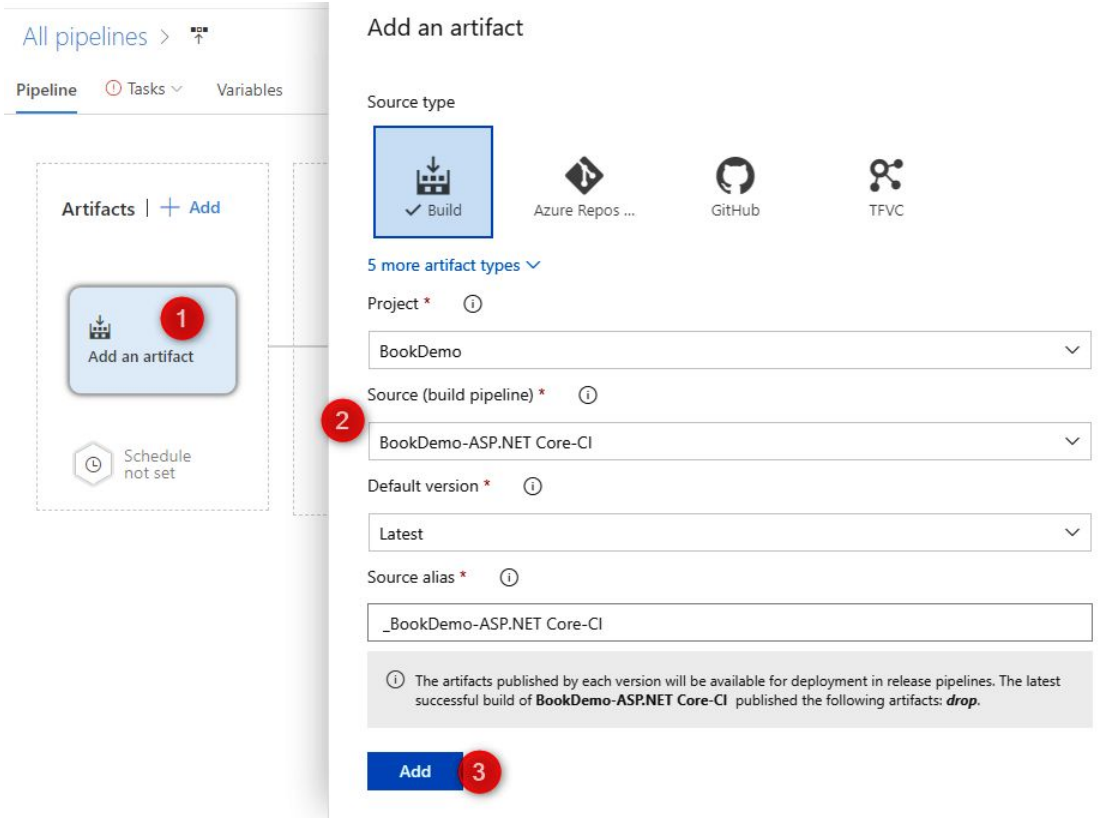
Creating the CD Pipeline: The Release

3. Then, in the next window, the first stage is named, for example, CI as the continuous integration environment:



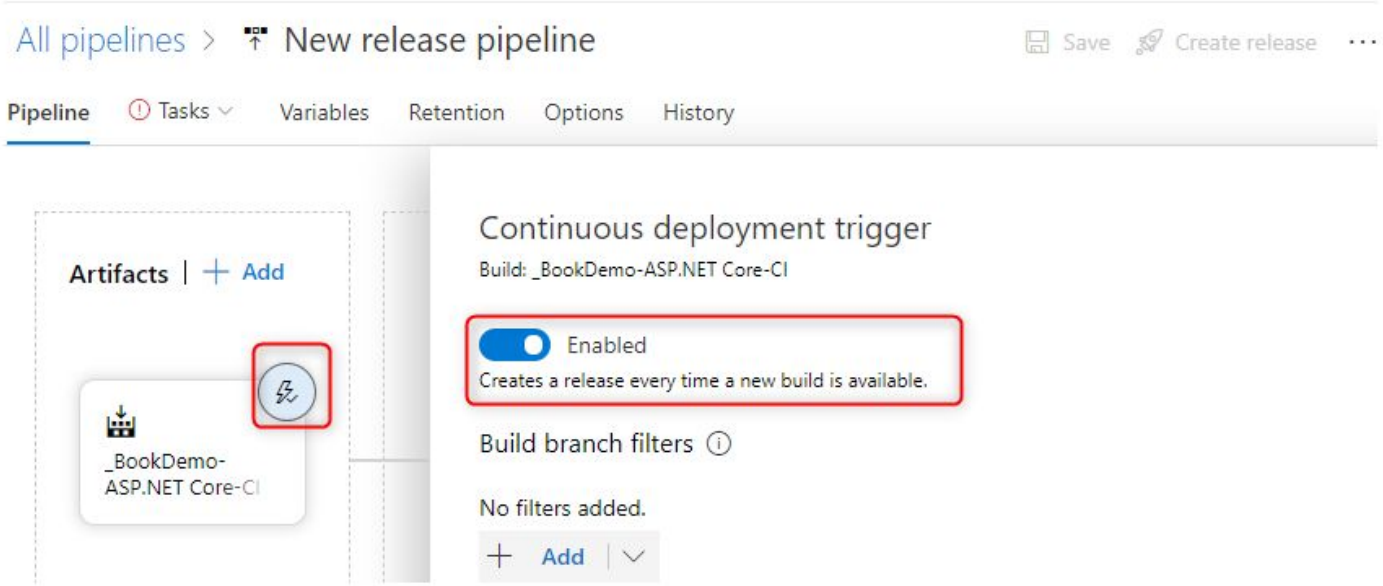
Creating the CD Pipeline: The Release

4. Configure the entry point of the release in the artifacts part by adding an artifact that is the build definition previously created in the Creating the CI pipeline section, as follows:



Creating the CD Pipeline: The Release

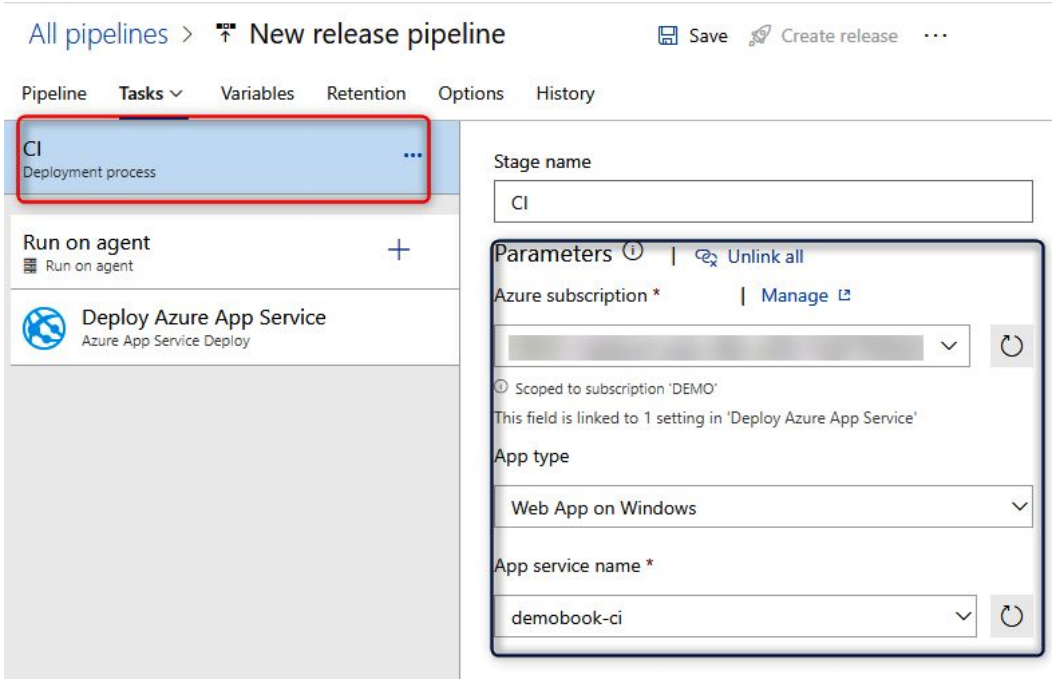
5. Configure the automatic release trigger for each successful build execution:



Creating the CD Pipeline: The Release

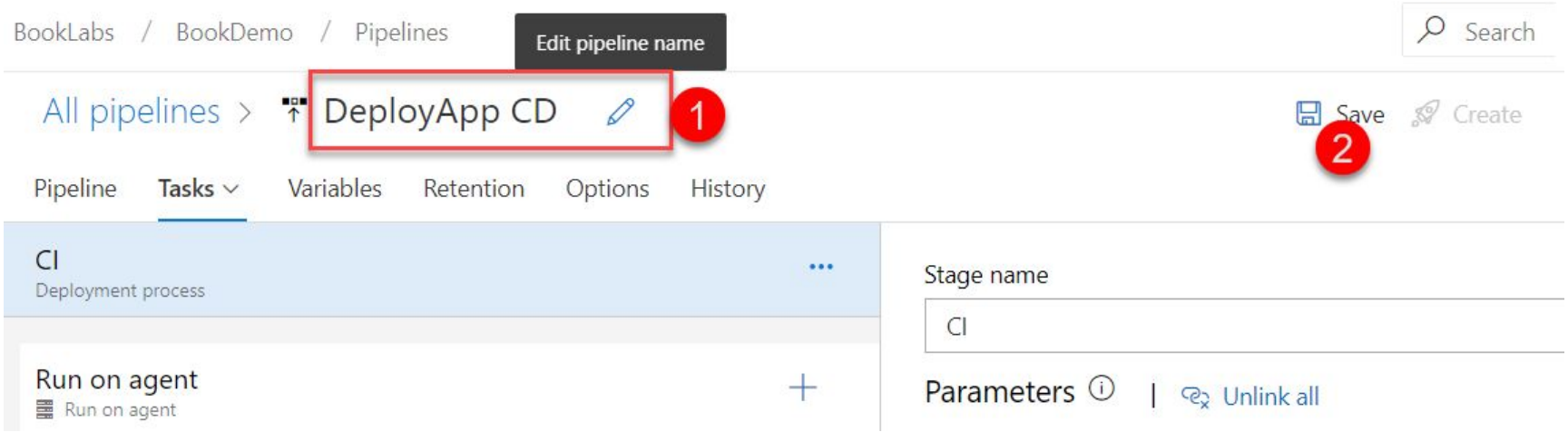
6. Configure the steps that will be executed in the CI stage; by clicking on the stage, we get exactly the same configuration window as the build:

- The agent's choice over the Run on agent section
- The configuration of the steps with their parameters



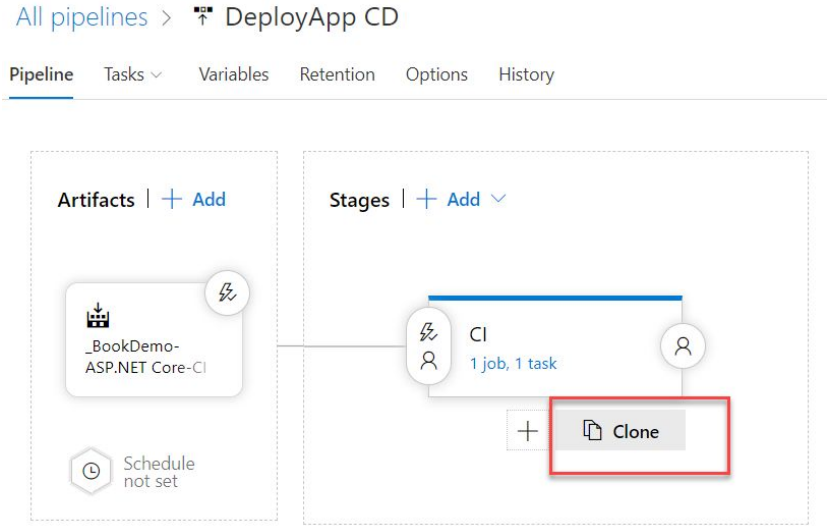
Creating the CD Pipeline: The Release

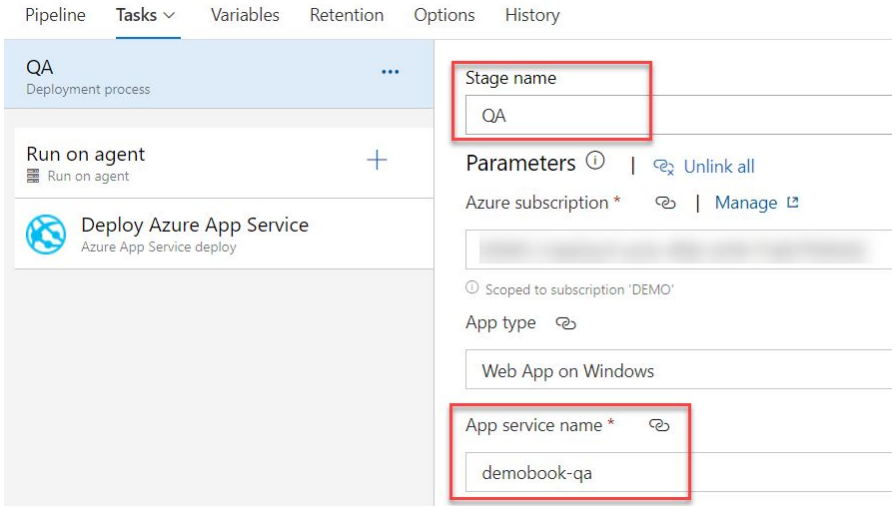
7. Rename the release with a name that simply describes what it does, and then save it, as follows:



Creating the CD Pipeline: The Release

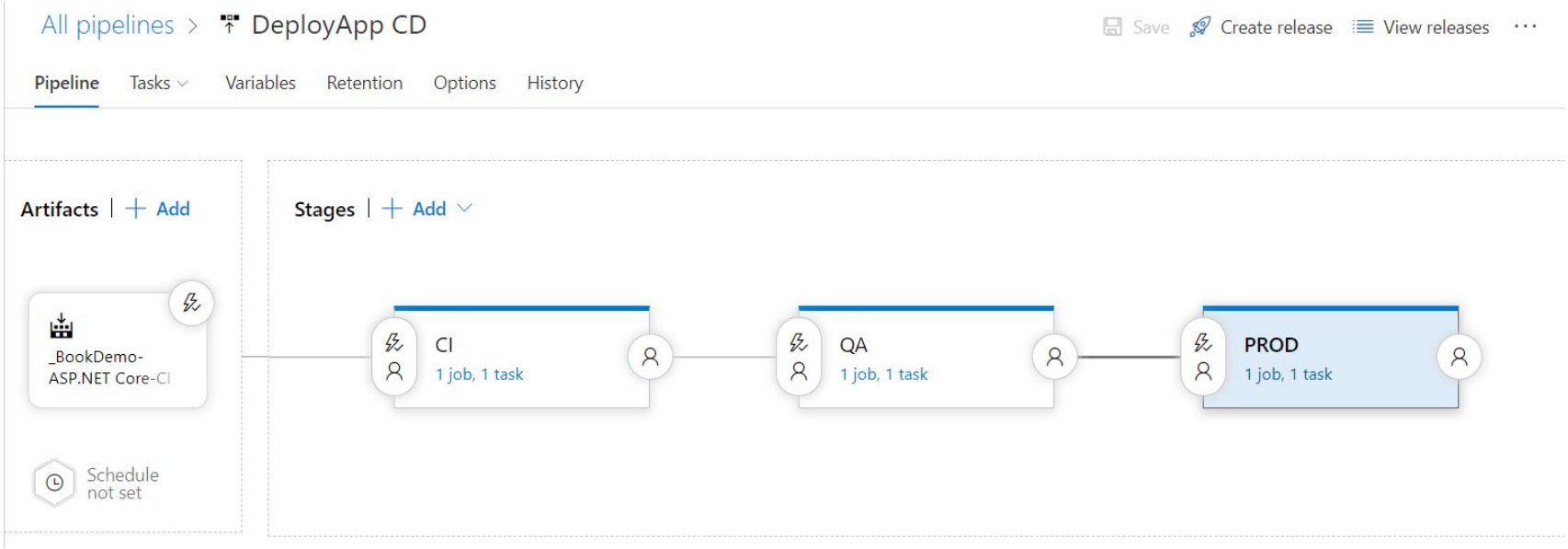
8. Complete the definition of the release with the deployment of the other environments (or stages). To simplify the manipulation, clone the CI environment settings in the release, and change the name of the app service name settings to the name of the web app:





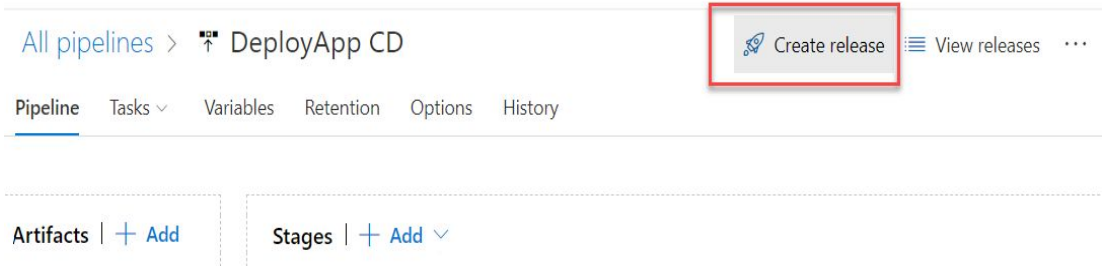
Creating the CD Pipeline: The Release

9. We finally get the release definition as follows:

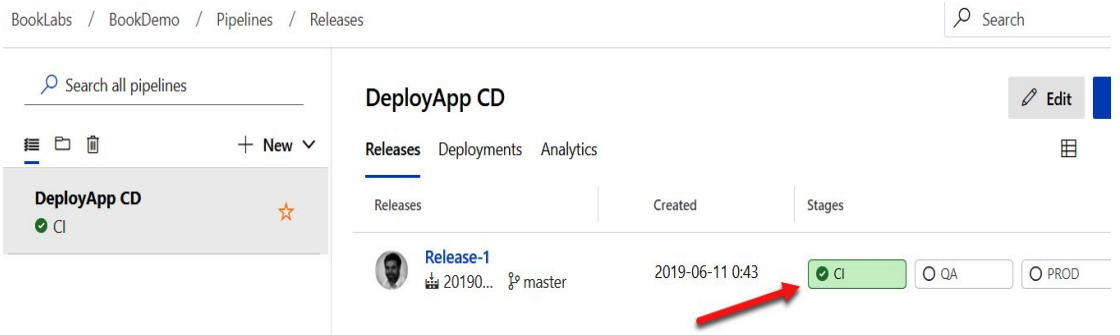


Creating the CD Pipeline: The Release

10. To trigger the deployment of application, create a new release by clicking on the Create release button:



11. At the end of its execution, see its deployment status:



Using GitLab CI

- Creating the CI/CD pipelines with Jenkins and Azure Pipelines.
- GitLab CI is one of the services offered by GitLab like Azure DevOps, is a cloud platform with the following:
    1. A source code manager
    2. A CI/CD pipeline manager
    3. A board for project management

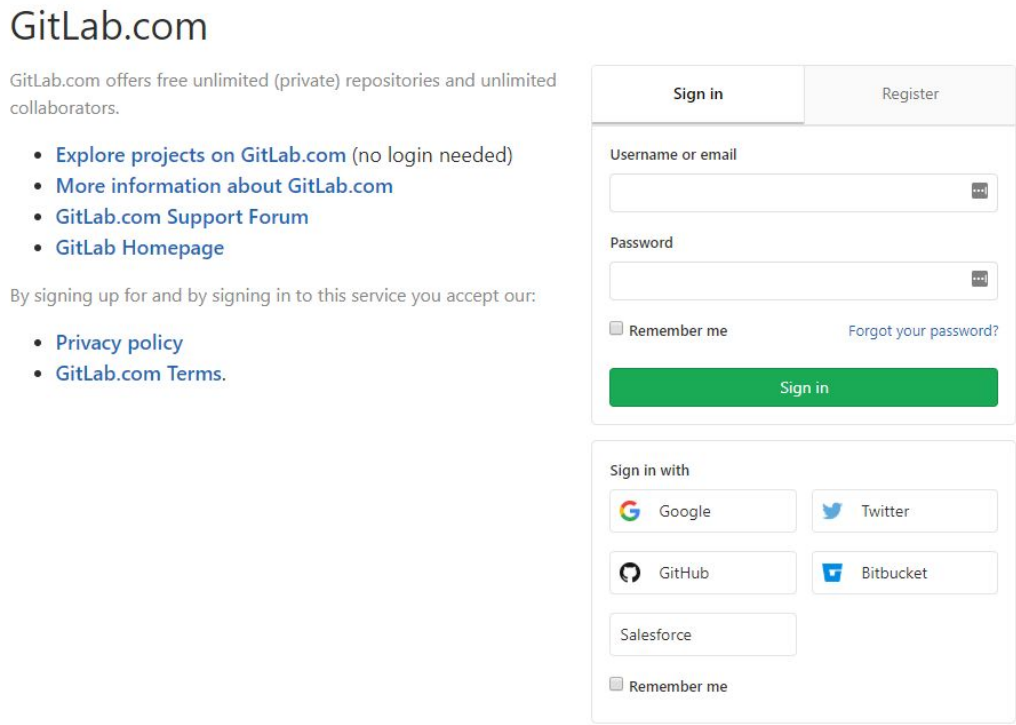
Using GitLab CI

- Following operations can be performed in GitLab CI:

1. Authentication at GitLab
2. Creating a new project and versioning its code in GitLab
3. The creation and execution of a CI pipeline in GitLab CI

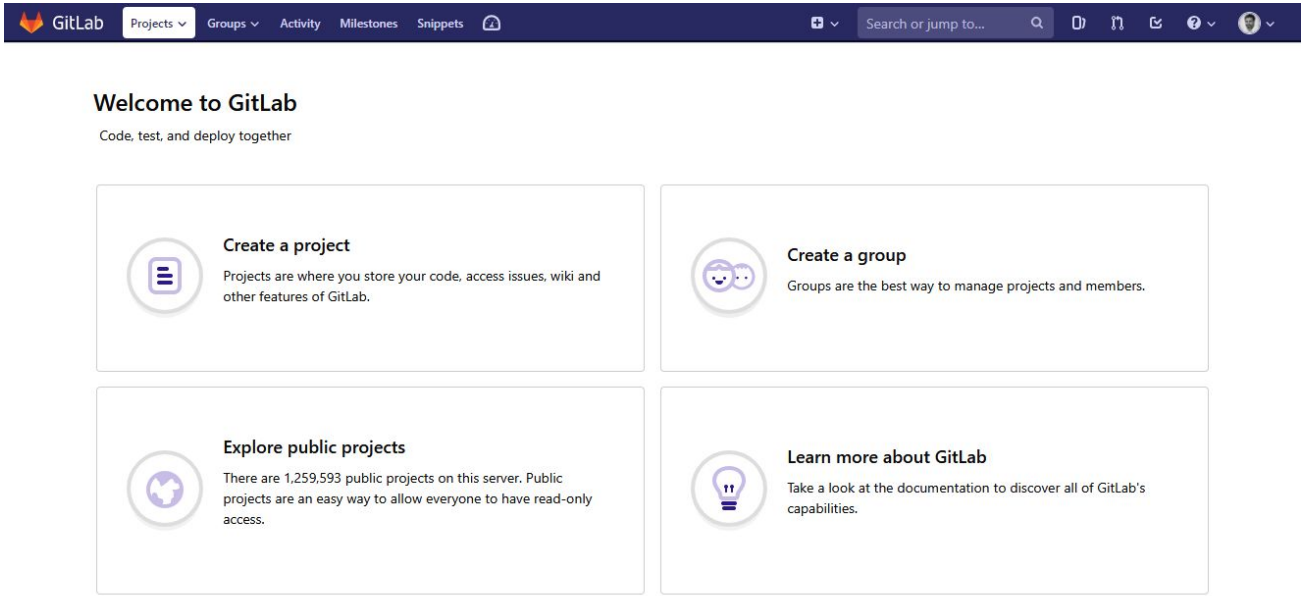
Authentication at GitLab

- Creating a GitLab account is free and can be done either by creating a GitLab account or using external accounts, such as Google, GitHub, Twitter, or Bitbucket.
- To create a GitLab account, in https://gitlab.com/users/sign_in#register- pane and choose the type of authentication.



Authentication at GitLab

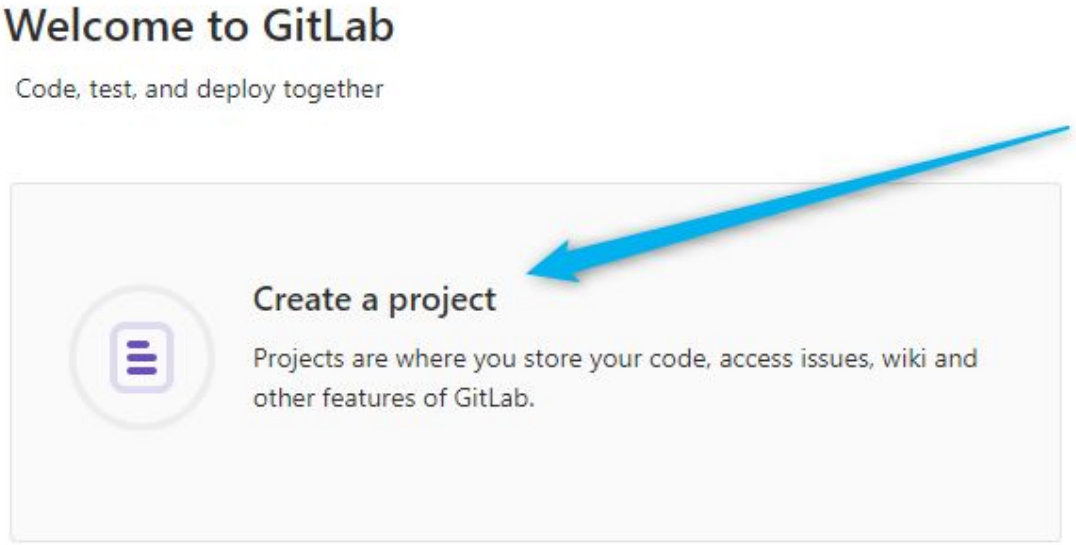
- Once the account has been created and authenticated, home page of the account will be displayed, which offers all the functionalities shown:



Creating a New Project and Managing Code Source

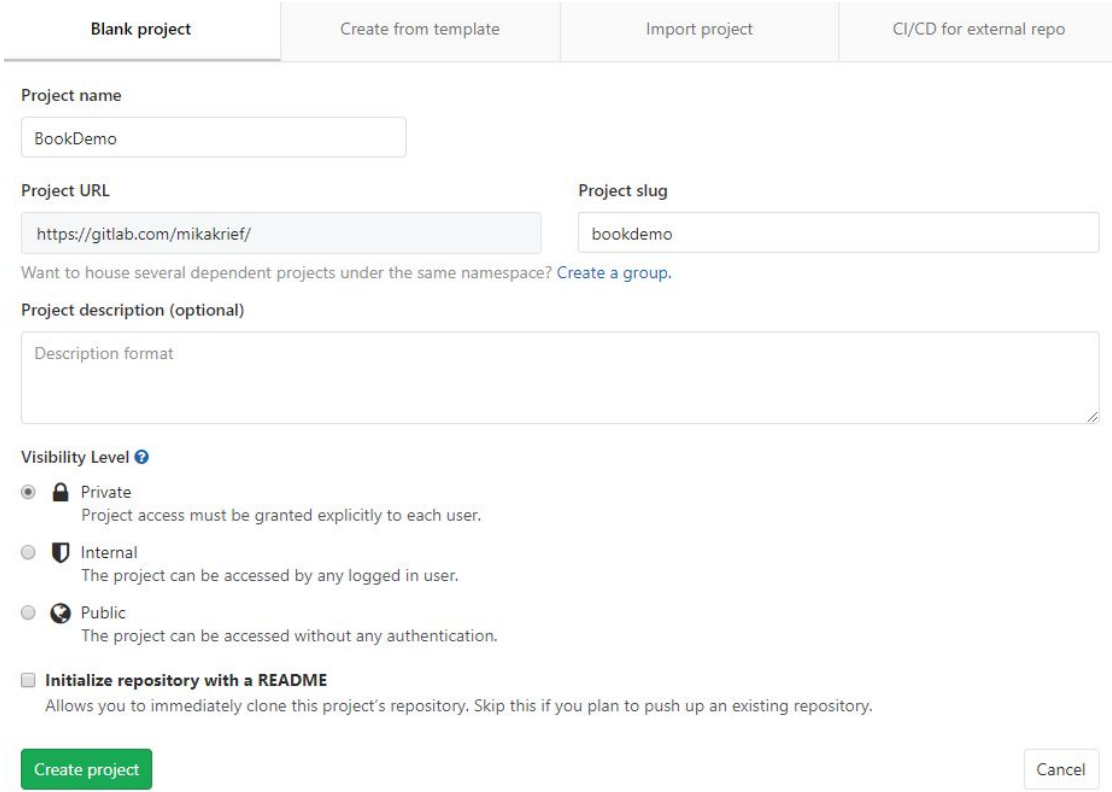
- To create a new project in GitLab, follow these steps:

1. Click on Create a project on the home page:



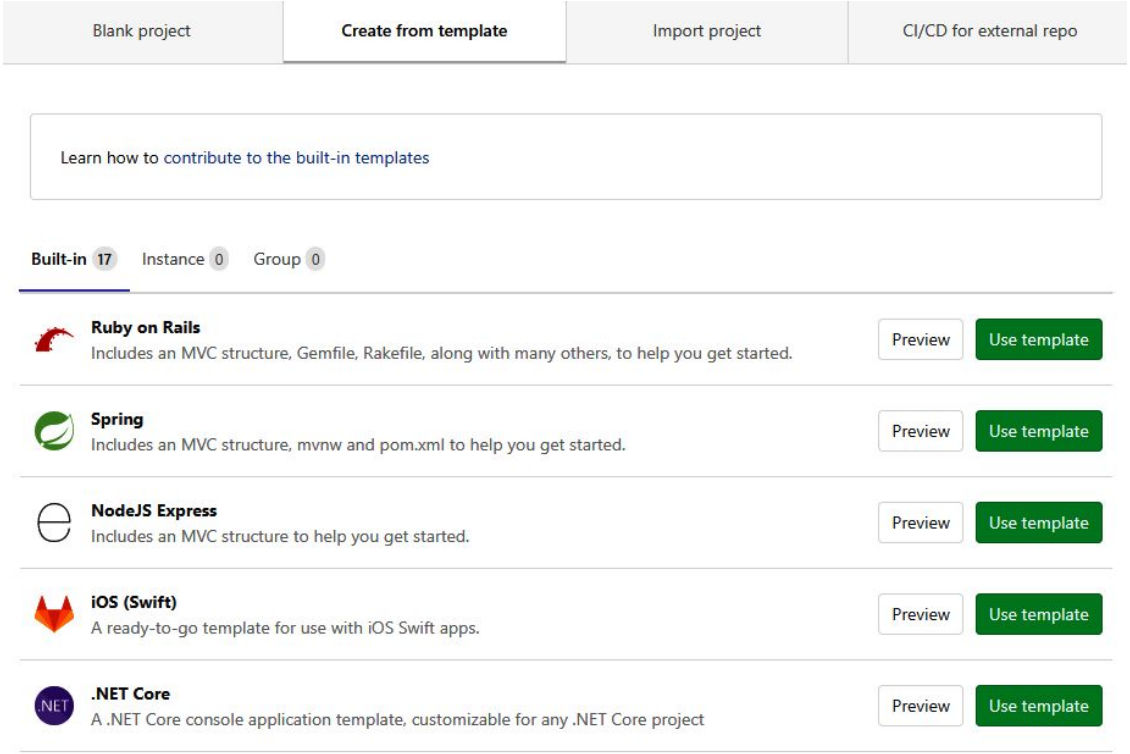
Creating a New Project and Managing Code Source

2. Then, choose a few options: To create an empty project (without code)—the form asks to enter its name:



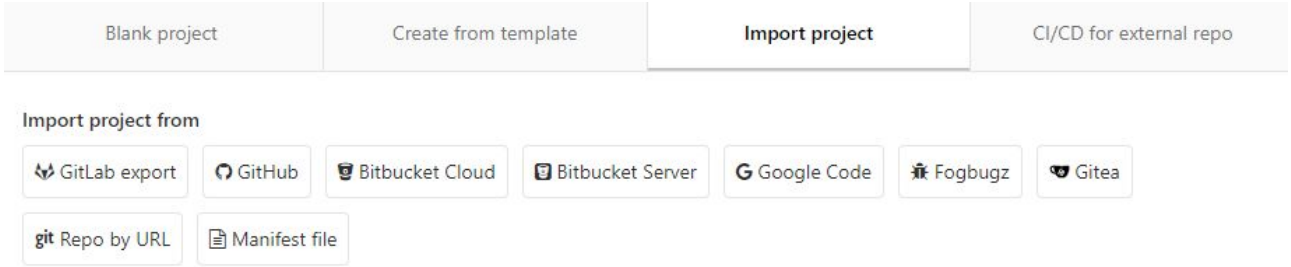
Creating a New Project and Managing Code Source

- To create a new project from a built-in template project, as follows:

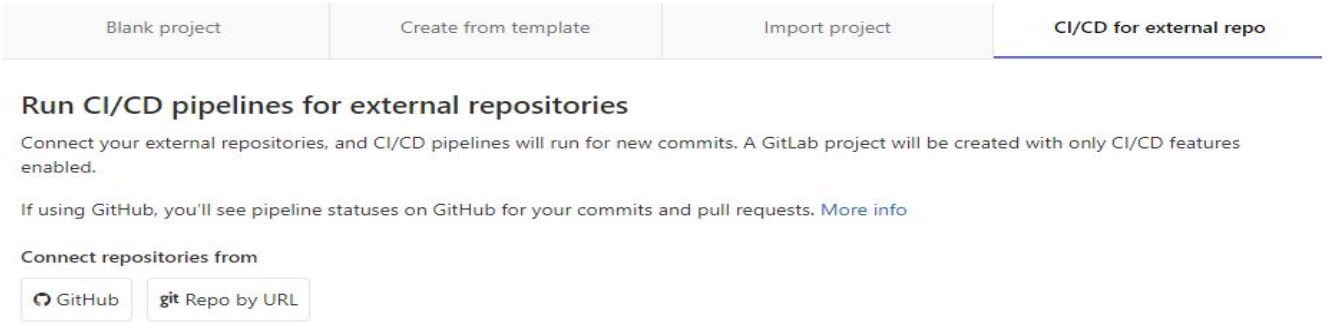


Creating a New Project and Managing Code Source

- To import code from an internal or external repository of another SVC platform:



- The code to import is located in an external SVC repository:



Creating a New Project and Managing Code Source

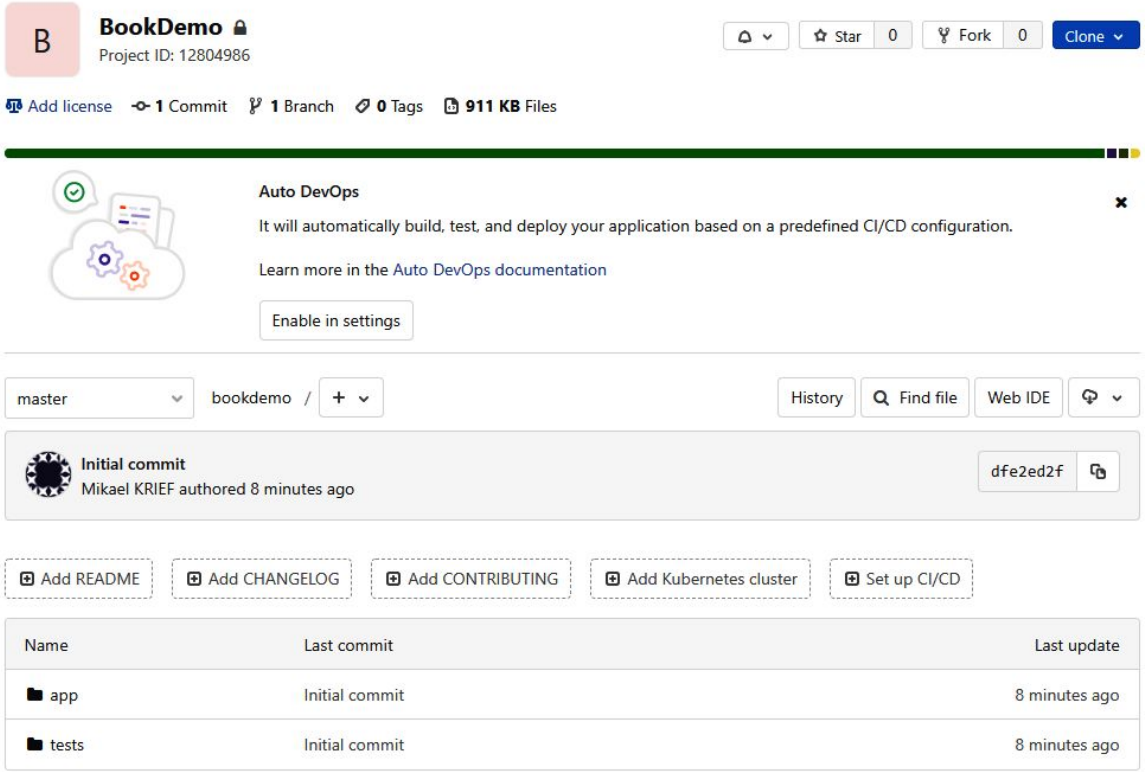
3. Once the project is created, different Git commands can be executed to push the code.
4. To do this, on local disk, create a new gitlabdemo directory and then clone the content.
5. Then, execute the following commands in a terminal to push the code into the repository

    - **git init**
    - **git remote add origin <git repo Url>**
    - **git add .**
    - **git commit -m "Initial commi**

Creating a New Project and Managing Code Source

- Once these commands have been executed, then a remote GitLab repository with lab code will be available.

- Remote GitLab repository:



Creating the CI Pipeline

- In GitLab CI, the creation of a CI pipeline (and CD) is not done via a graphical interface, but with a YAML file at the root of the project.
- This method, which consists of describing the process of a pipeline in a file that is located with the code, can be called Pipeline as Code in the same way as the IaC:

1. To create this pipeline, create, at the root of the application code, a ".gitlab-ci.yml" file with the following content code:

Creating the CI Pipeline

- ".gitlab-ci.yml" file with the content code:

```
image: microsoft/dotnet:latest
stages:
    - build
    - test

variables:
    BuildConfiguration: "Release"

build:
    stage: build
    script:
        - "cd app"
        - "dotnet restore"
        - "dotnet build --configuration $BuildConfiguration"
test:
    stage: test
    script:
        - "cd tests"
        - "dotnet test --configuration $BuildConfiguration"
```

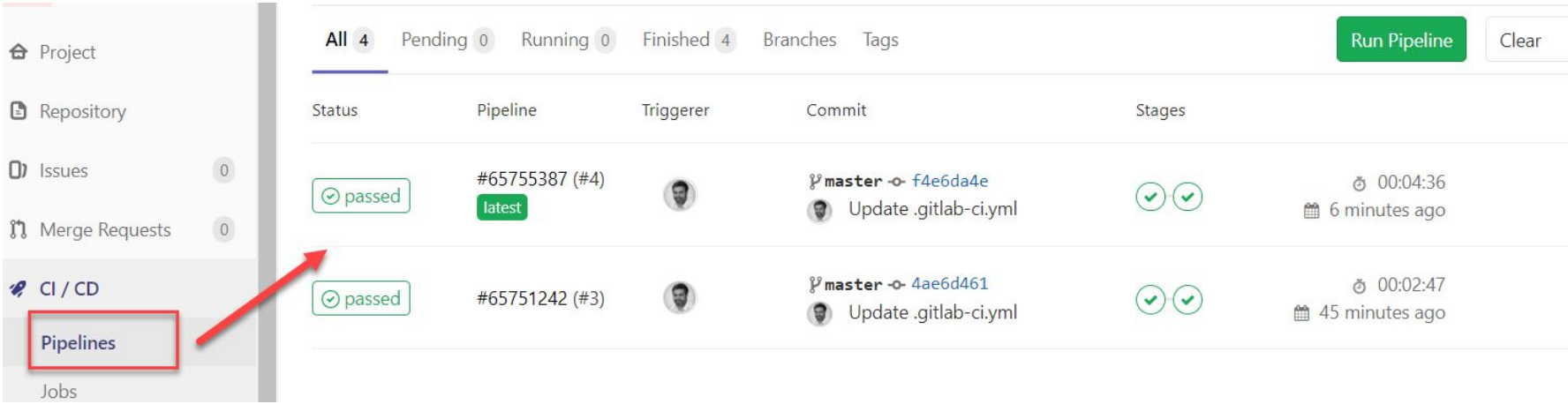
Creating the CI Pipeline

- Then, define two stages: one for the build and one for the test execution, as well as a BuildConfiguration variable that will be used in the scripts.
- Finally, describe each of the stages of the scripts to be executed in their respective directories.
- These .NET core scripts are identical to the ones we saw in the Using Azure Pipelines

2. Then, we will commit and push this file into the remote repository.
3. Just after pushing the code, we can see that the CI process has been triggered.

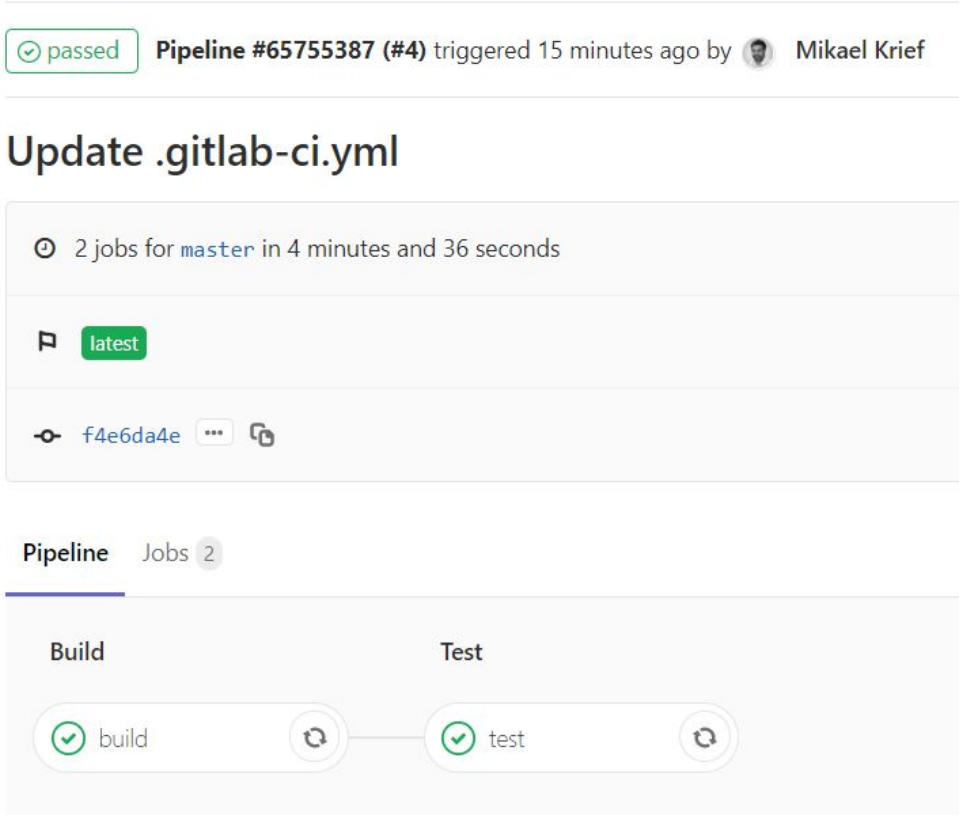
Accessing the CI Pipeline Execution Details

- To access the execution details of the executed CI pipeline, follow these steps:
- 1. In the GitLab CI menu, go to CI/CD | Pipelines, and see the list of pipeline executions, as shown in the following screenshot:



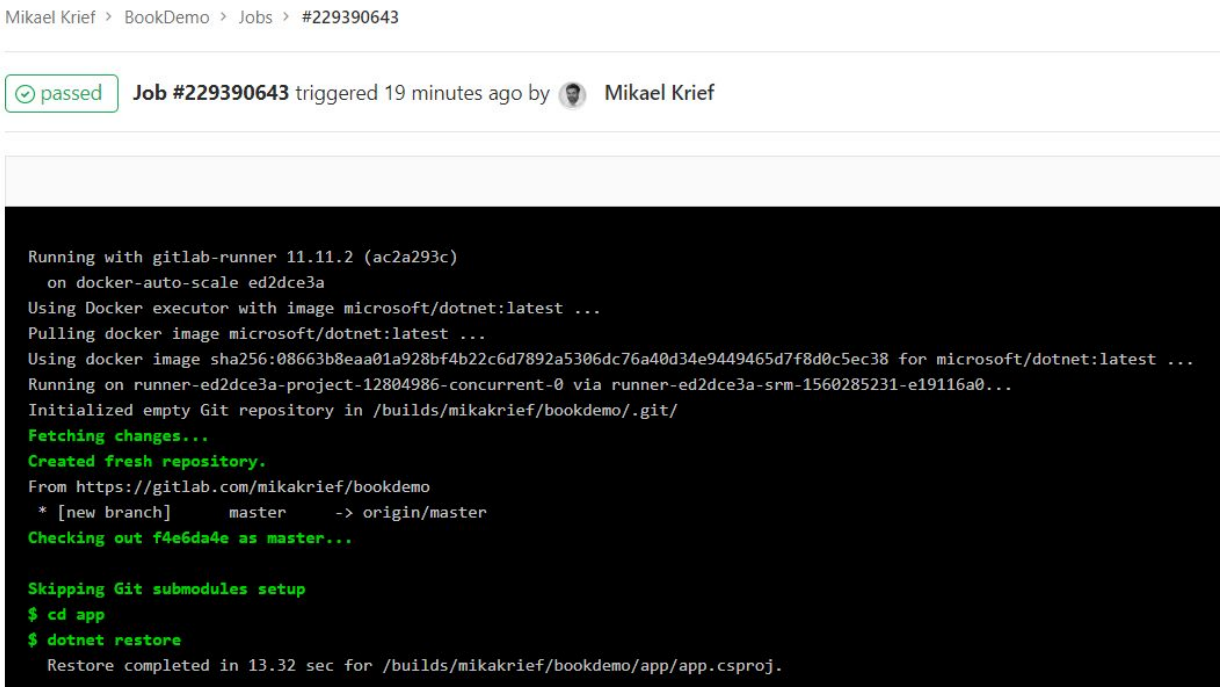
Accessing the CI Pipeline Execution Details

2. To display the details of the pipeline, click on the desired pipeline execution:



Accessing the CI Pipeline Execution Details

3. See the execution status as well as the two stages that was defined in the pipeline YAML file. To view the details of the execution logs for a stage, click on the stage:



Thank You