

DevOps Continuous Integration and Continuous Deployment Pipeline

Managing Your Source Code with Git

Introduction to DevOps

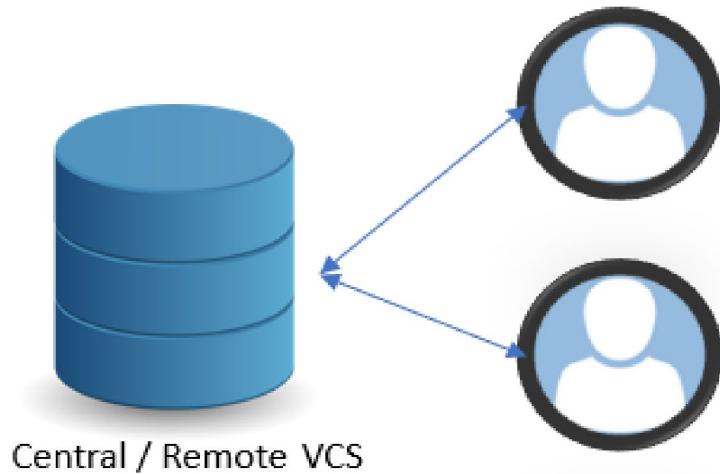
- **Version Control System** (VCS) or more commonly as a **Version Control Manager** (VCM)
- The goals of these VCSes are mainly to do the following:
 - Allow collaboration of developers' code.
 - Retrieve the code.
 - Version the code.
 - Track code changes.
- The implementation of a CI/CD process can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

Overviewing Git and its command lines

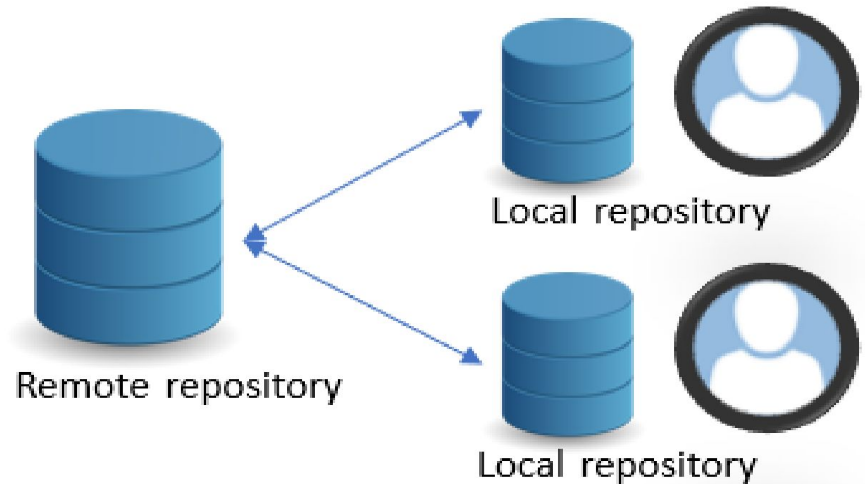
- There are two types of VCS:
 1. Centralized Version Control System
 2. Distributed Version Control System
- The first type to emerge is the Centralized Version Control System, such as SVN, CVS, Subversion, and TFVC (or SourceSafe).
- These systems consist of a remote server that centralizes the code 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).

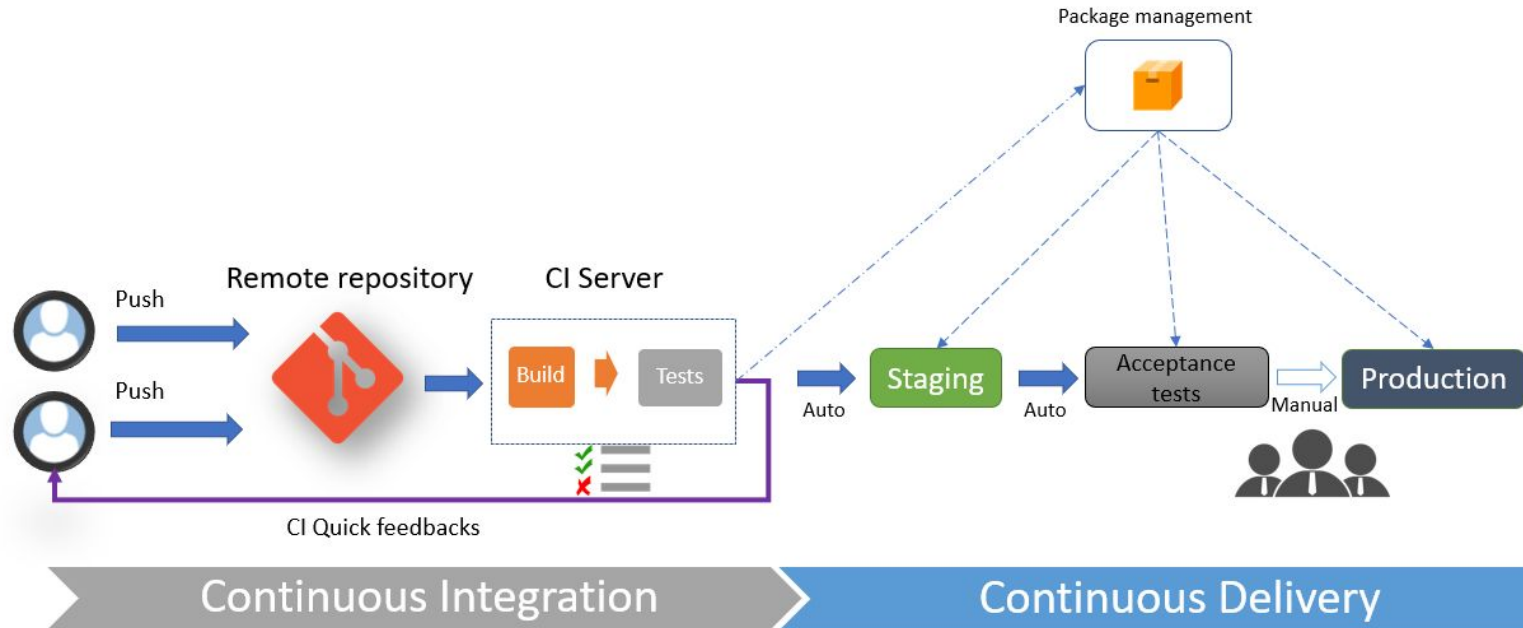
Continuous Integration and Continuous Delivery

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.



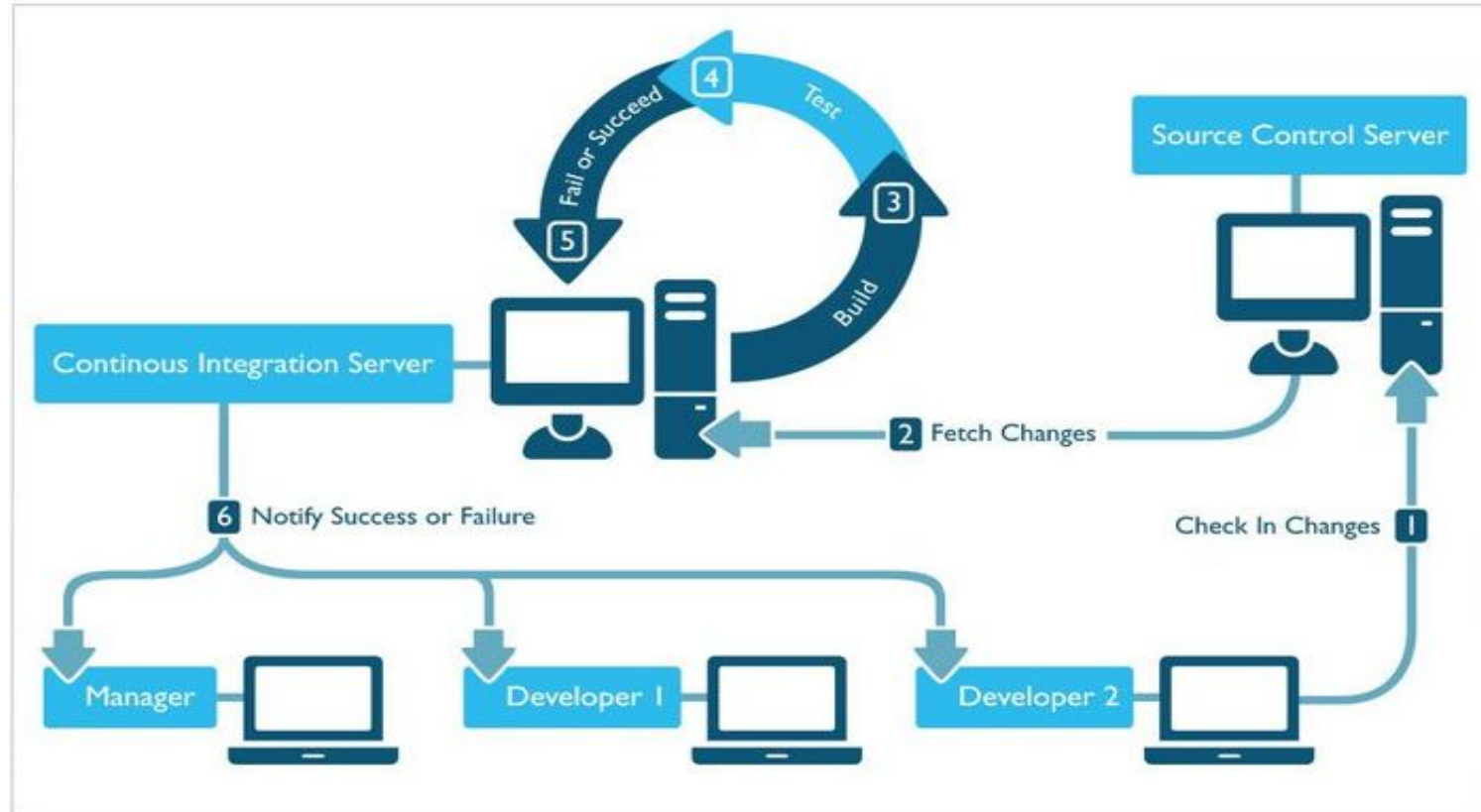
Continuous Integration (CI)

- 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)

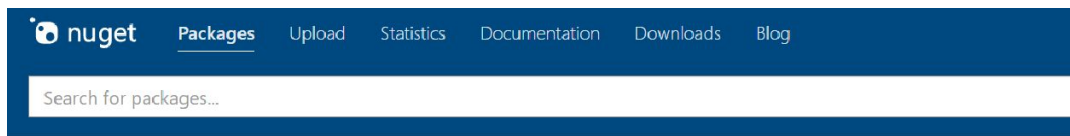
- The different tools for setting up a CI/CD pipeline are as follows:
 1. A source control version
 2. A package manager
 3. A CI server
 4. A configuration manager
- These tools will only be effective in delivering added value to the product if development and operations team work together around them.

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.



There are 155 895 packages



NUnit  by: charliepoole rprouse

↓ 42 874 936 total downloads | ⌚ last updated 18 days ago | 📦 Latest version: 3.12.0 | 🔗 [nunit test testing tdd](#)

NUnit features a fluent assert syntax, parameterized, generic and theory tests and is user-extensible. This package which is referenced by your tests. You will need to install version 3 of the nunit3-console program or a third-party runner



Newtonsoft.Json  by: newtonsoft jamesnk

Json.NET

↓ 228 702 270 total downloads | ⌚ last updated a month ago | 📦 Latest version: 12.0.2 | 🔗 [json](#)

Json.NET is a popular high-performance JSON framework for .NET



EntityFramework  by: EntityFramework aspnet Microsoft

↓ 63 760 274 total downloads | ⌚ last updated 25 days ago | 📦 Latest version: 6.3.0-preview5-19254-05

Entity Framework is Microsoft's recommended data access technology for new applications.

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:

Repositories Manage repositories

+ Create repository

Name ↑	Type
maven-central	proxy
maven-public	group
maven-releases	hosted
maven-snapshots	hosted
nuget-group	group
nuget-hosted	hosted
nuget.org-proxy	proxy

Repositories / Select Recipe

Recipe ↑
apt (hosted)
apt (proxy)
bower (group)
bower (hosted)
bower (proxy)
docker (group)
docker (hosted)
docker (proxy)
gitlfs (hosted)
go (group)
go (proxy)
maven2 (group)
maven2 (hosted)
maven2 (proxy)
npm (group)
npm (hosted)
npm (proxy)

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

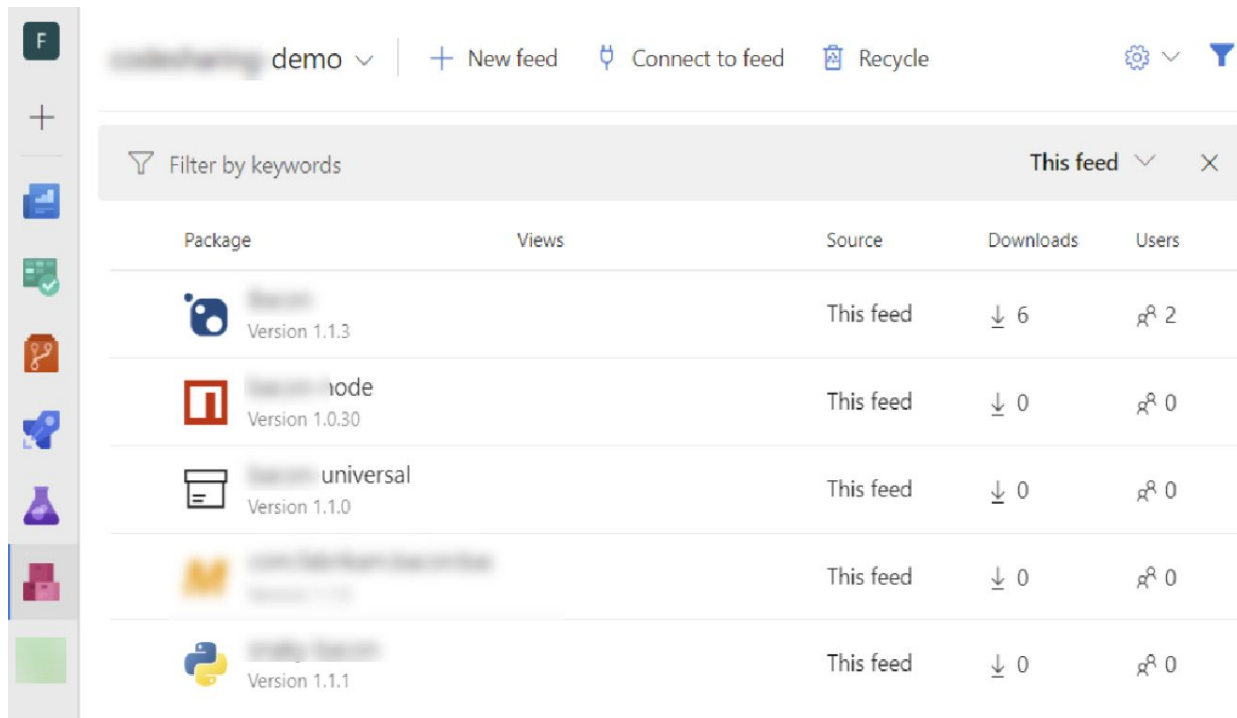
- **Azure Artifacts :**
- **Advantages:**
- It is fully integrated with other Azure DevOps services such as Azure Pipelines, which allows managing CI/CD pipelines.
- In Azure Artifacts, there is also a type of package called universal packages that allows storing all types of files (called a package) 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



The screenshot shows the Azure Artifacts web interface. At the top, there's a header with a dropdown menu set to 'demo', and buttons for '+ New feed', 'Connect to feed', and 'Recycle'. Below the header is a search bar labeled 'Filter by keywords' and a dropdown for 'This feed'. The main content is a table with columns: Package, Views, Source, Downloads, and Users. The table lists five packages: 'demo' (Version 1.1.3), 'demo node' (Version 1.0.30), 'demo universal' (Version 1.1.0), 'demo universal' (Version 1.1.0), and 'demo universal' (Version 1.1.1). Each package has a corresponding icon and a download count of 0, except for the first one which has 6 downloads and 2 users.

Package	Views	Source	Downloads	Users
 demo Version 1.1.3		This feed	↓ 6	👤 2
 demo node Version 1.0.30		This feed	↓ 0	👤 0
 demo universal Version 1.1.0		This feed	↓ 0	👤 0
 demo universal Version 1.1.0		This feed	↓ 0	👤 0
 demo universal Version 1.1.1		This feed	↓ 0	👤 0

Using Jenkins

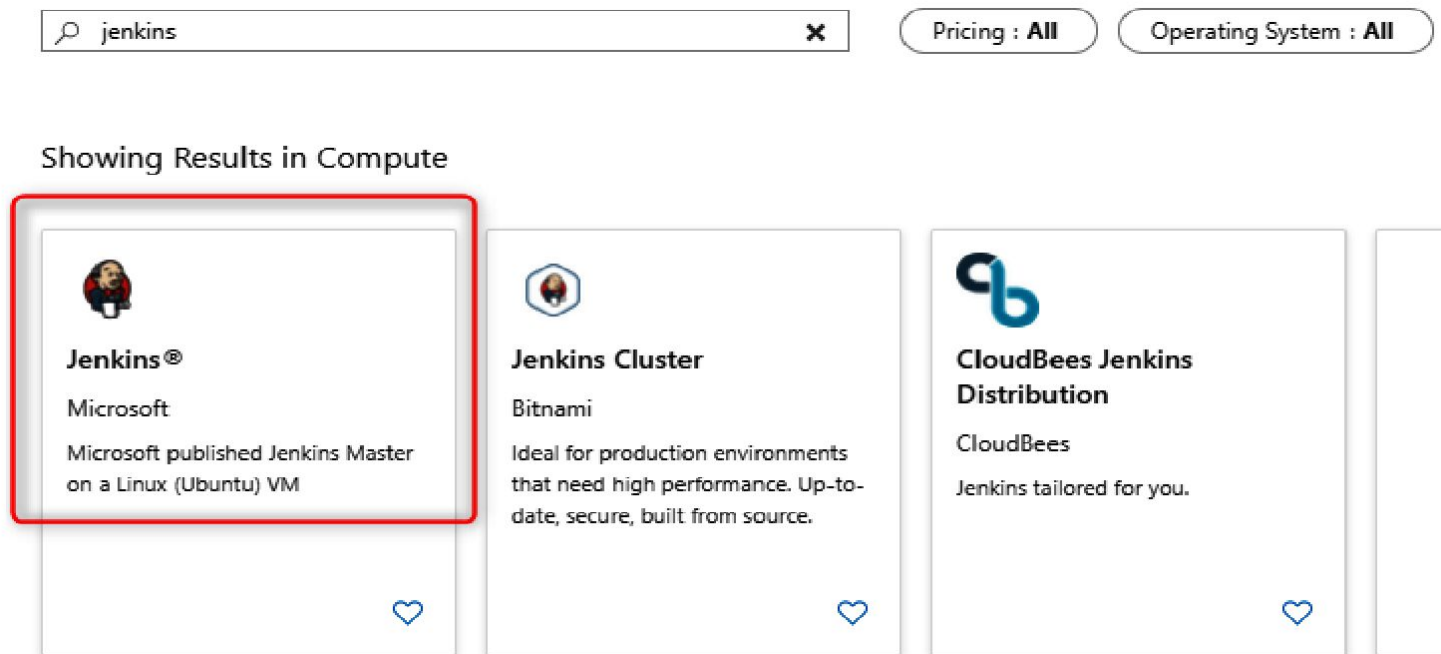
- Jenkins is one of the oldest continuous integration tools, initially released in 2011.
- It is open source and developed in Java.
- Jenkins has become famous due to the large community working on it and its plugins.
- More than 1,500 Jenkins plugins.
- If one of your tasks does not have a plugin, then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

Connect
 Start
 Restart
 Stop
 Capture
 Delete
 Refresh

Resource group [\(change\)](#) : JenkinsDemo

Status : Running
 Location : West Europe
 Subscription [\(change\)](#) : DEMO
 Subscription ID : 1da42ac

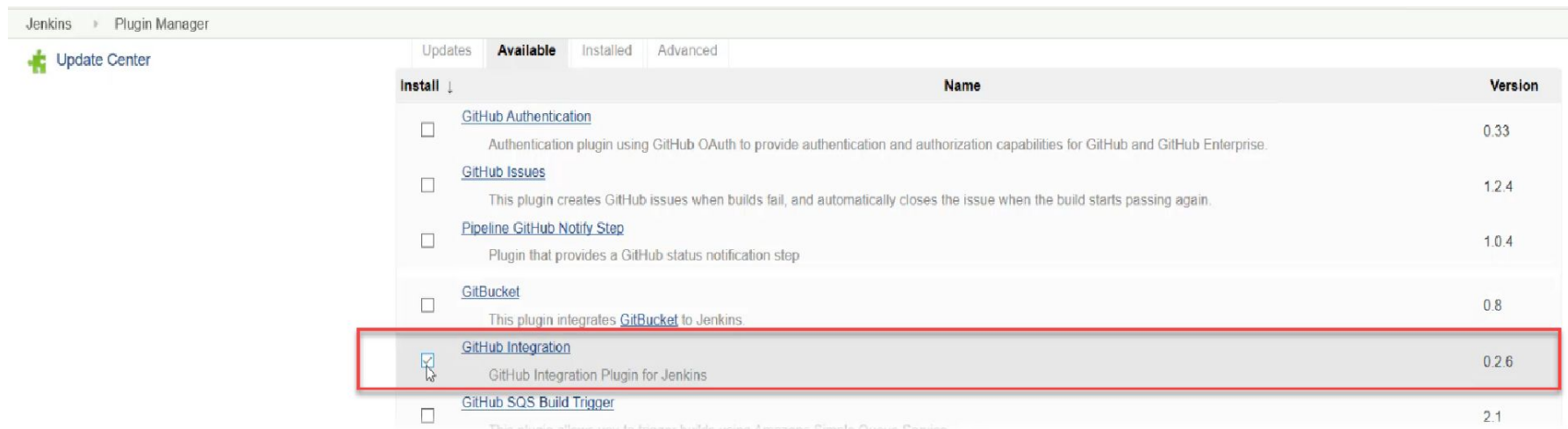
Computer name : jenkins
 Operating system : Linux (ubuntu 16.04)
 Size : Standard DS2 v2 (2 vcpus, 7 GiB memory)
 Ephemeral OS disk : N/A
 Public IP address : 4
 Private IP address : 10.1.0.4
 Virtual network/subnet : jenkins-vnet/jenkins
 DNS name : demojenkins. .cloudapp.azure.com

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.

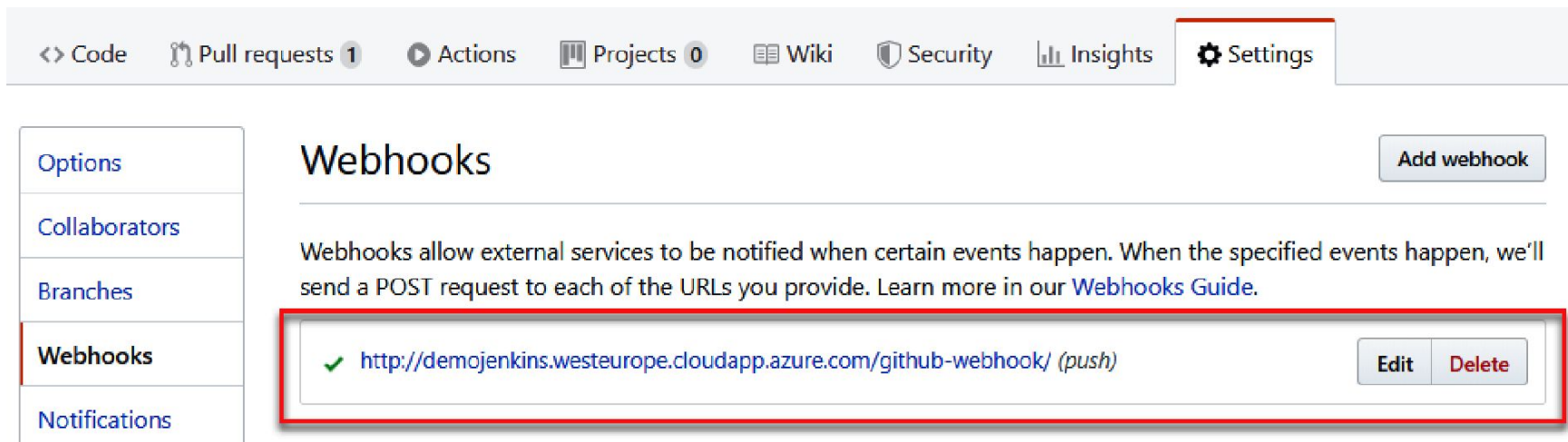
- Configuration of a GitHub webhook for Jenkins:

The screenshot shows the GitHub 'Settings' page for a repository. The 'Settings' tab is selected and highlighted with a red circle labeled '1'. In the left sidebar, the 'Webhooks' option is selected and highlighted with a red circle labeled '2'. The main content area is titled 'Webhooks / Manage webhook'. It contains the following fields and options:

- Payload URL ***: A text input field containing the URL `http://demojenkins.westeurope.cloudapp.azure.com/github-webhook`, which is highlighted with a red rectangle.
- Content type**: A dropdown menu set to `application/x-www-form-urlencoded`.
- Secret**: An empty text input field.
- Which events would you like to trigger this webhook?**: Three radio button options:
 - ☒ Just the push event. (This option is highlighted with a red rectangle.)
 - ☐ Send me everything.
 - ☐ Let me select individual events.
- Active**: A checked checkbox with the text 'We will deliver event details when this hook is triggered.'
- Buttons**: Two buttons at the bottom: 'Update webhook' (green) and 'Delete webhook' (red).

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.



The screenshot displays the GitHub interface for configuring webhooks. At the top, a navigation bar includes links for Code, Pull requests (1), Actions, Projects (0), Wiki, Security, Insights, and Settings (which is highlighted). On the left, a sidebar menu lists Options, Collaborators, Branches, Webhooks (highlighted), and Notifications. The main content area is titled 'Webhooks' and features an 'Add webhook' button. Below the title, a descriptive text explains that webhooks allow external services to be notified of certain events, with a link to the 'Webhooks Guide'. A table lists the configured webhooks, with one entry highlighted by a red border: a green checkmark, the URL 'http://demojenkins.westeurope.cloudapp.azure.com/github-webhook/' followed by '(push)', and 'Edit' and 'Delete' buttons.

Options
Collaborators
Branches
Webhooks
Notifications

Webhooks

Add webhook

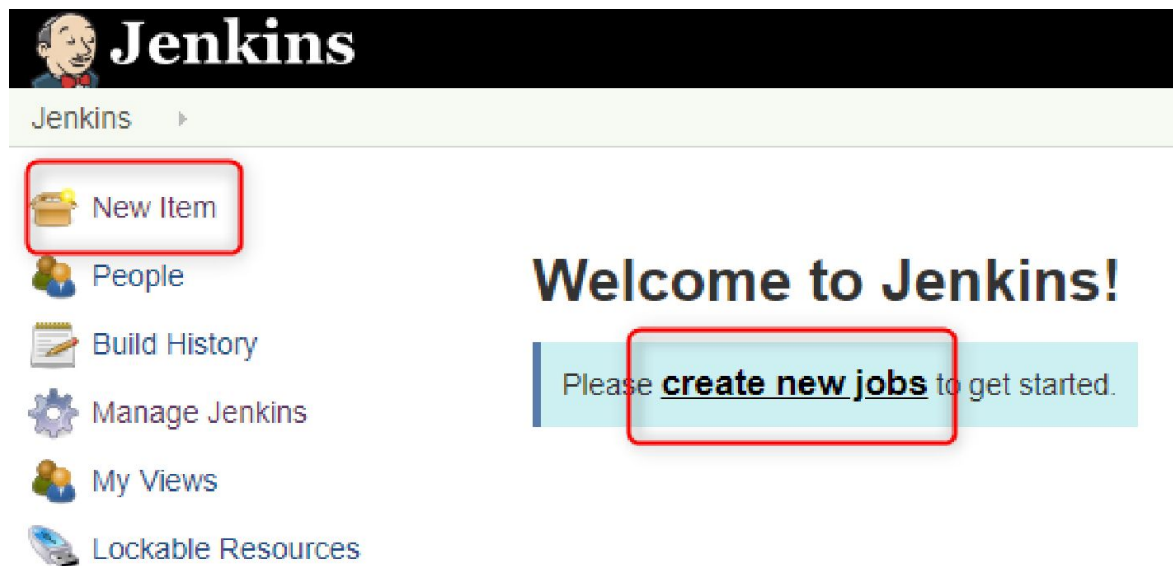
Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

✓	http://demojenkins.westeurope.cloudapp.azure.com/github-webhook/ (push)	Edit	Delete
---	--	------	--------

Configuring a Jenkins CI job

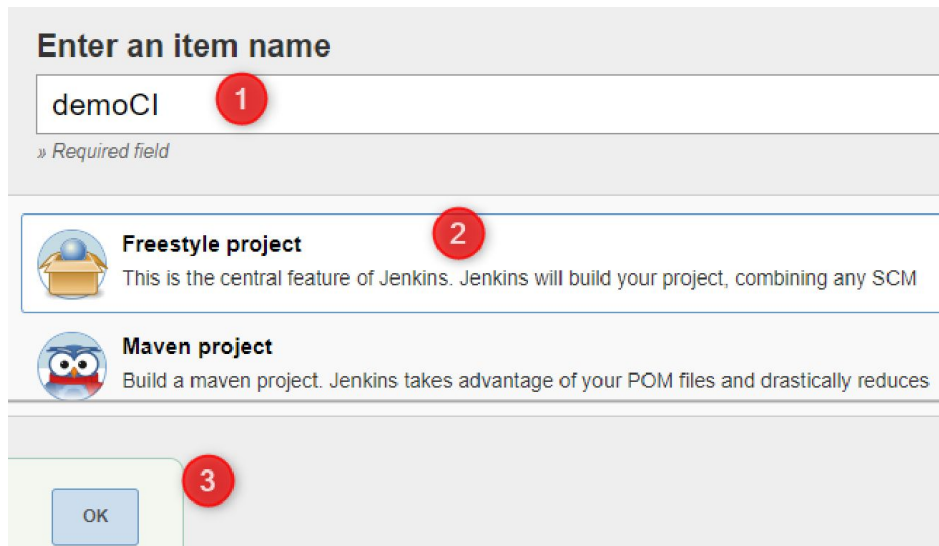
- To configure Jenkins, follow these steps:

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:



The screenshot shows the Jenkins job configuration interface. It includes a text input field for the job name, a list of project templates, and an OK button. Red circles with numbers 1, 2, and 3 highlight the steps: entering the job name, selecting the Freestyle project template, and clicking the OK button.

Enter an item name

demoCI 1

» Required field

Freestyle project 2
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces

OK 3

Configuring a Jenkins CI job

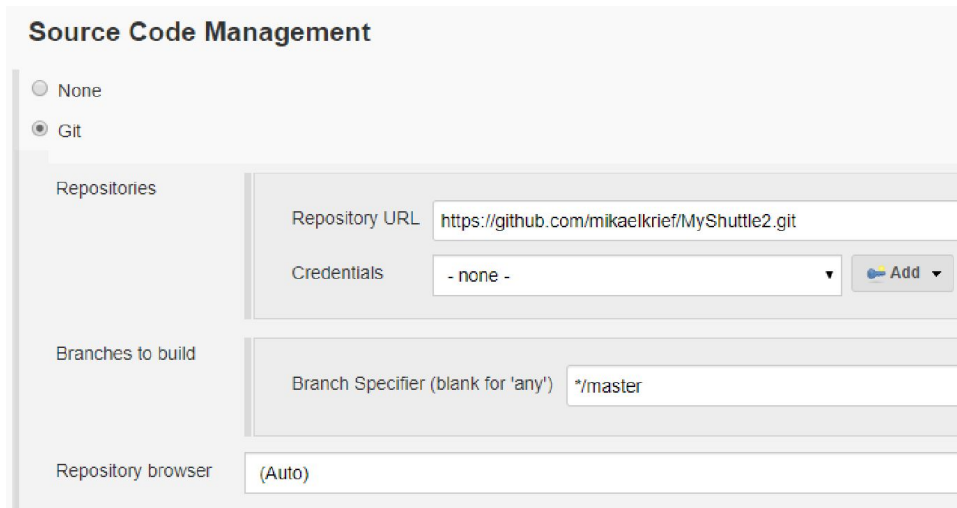
- To configure Jenkins, follow these steps:
3. Then, configure the job with the following parameters:
 - In the GitHub project input, enter the URL of the GitHub repository as follows:



The screenshot shows the Jenkins job configuration page for a GitHub project. At the top, there are two right-pointing chevrons (> >). Below them is a 'Description' field with a text area and a '[Plain text] [Preview](#)' link. Underneath the description field are three checkboxes: 'Enable project-based security' (unchecked), 'Discard old builds' (unchecked), and 'GitHub project' (checked). At the bottom, the 'Project url' field contains the text 'https://github.com/mikaelkrief/MyShuttle2/'.

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:

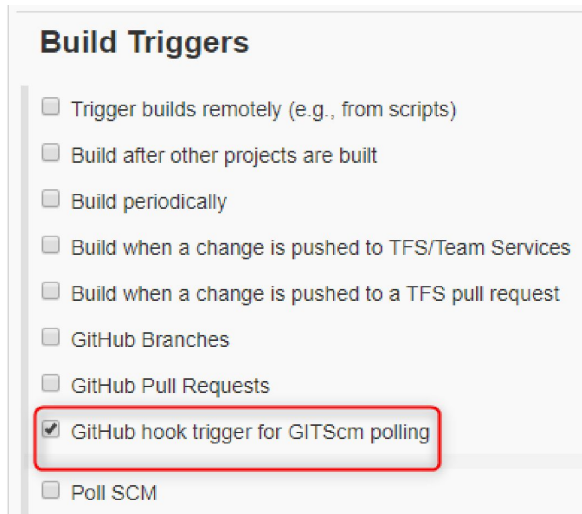


The screenshot shows the 'Source Code Management' configuration page in Jenkins. It features a sidebar with 'None' and 'Git' options, where 'Git' is selected. The main area is divided into three sections: 'Repositories', 'Branches to build', and 'Repository browser'. In the 'Repositories' section, the 'Repository URL' is set to 'https://github.com/mikaelkrief/MyShuttle2.git' and 'Credentials' is set to '- none -'. The 'Branches to build' section has a 'Branch Specifier (blank for \'any\')' set to '*/master'. The 'Repository browser' section is set to '(Auto)'.

Source Code Management	
<input type="radio"/> None	
<input checked="" type="radio"/> Git	
Repositories	<div>Repository URL: <input type="text" value="https://github.com/mikaelkrief/MyShuttle2.git"/></div> <div>Credentials: <input type="text" value="- none -"/> Add</div>
Branches to build	<div>Branch Specifier (blank for 'any'): <input type="text" value="*/master"/></div>
Repository browser	<input type="text" value="(Auto)"/>

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:

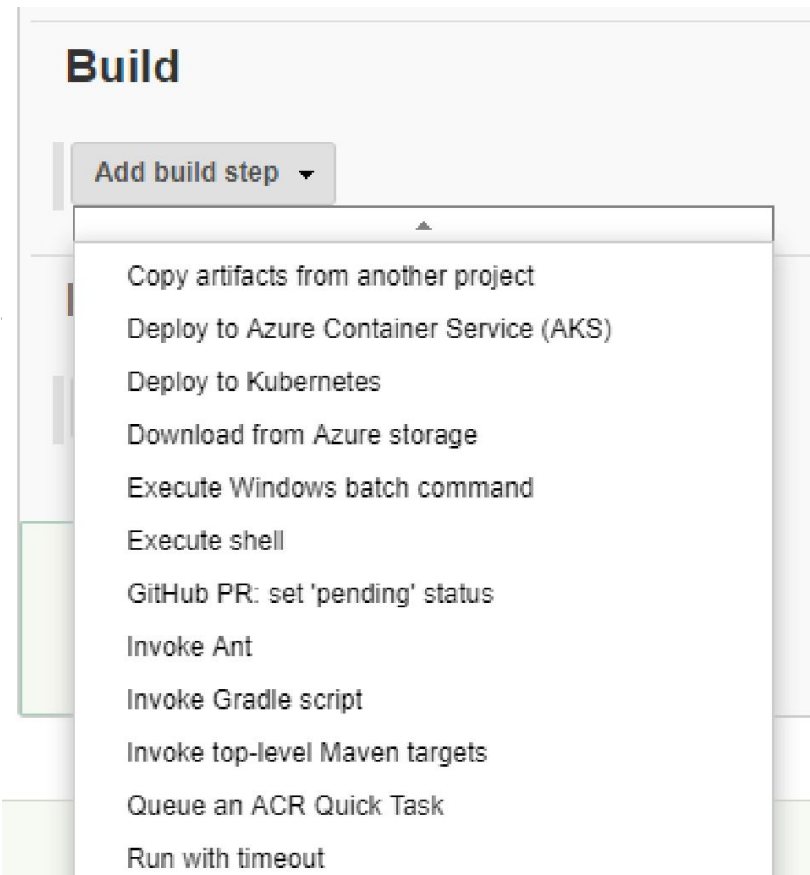


Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ Build when a change is pushed to TFS/Team Services
- ☐ Build when a change is pushed to a TFS pull request
- ☐ GitHub Branches
- ☐ GitHub Pull Requests
- ☒ GitHub hook trigger for GITScm polling
- ☐ Poll SCM

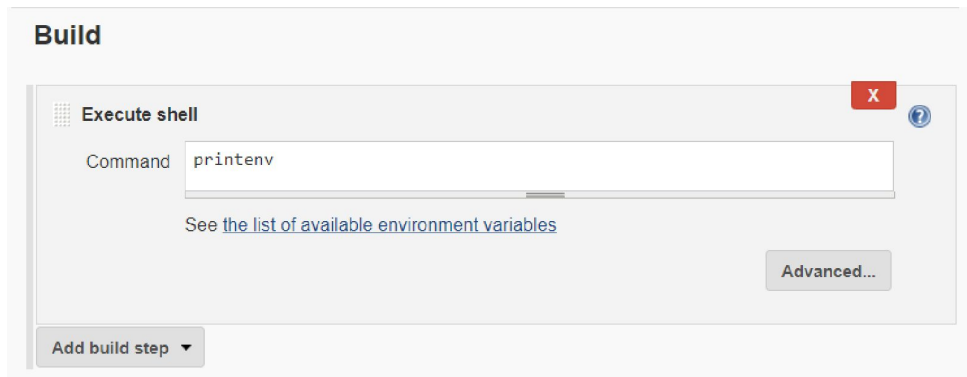
Configuring a Jenkins CI job

- 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

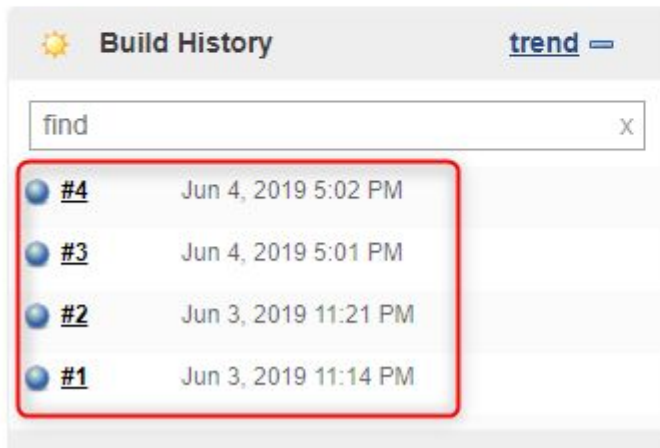
- To test job execution, perform these steps:
 1. Modify the code of GitHub repository, for example, by modifying the Readme.md file.
 2. Then, commit to the master branch directly from the GitHub web interface.
 3. After making this commit, the DemoCI job is queued up and running in Jenkins.
 4. By 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.



The screenshot shows the Jenkins web interface for a job named 'demoCI #4'. The breadcrumb navigation at the top reads 'Jenkins > demoCI > #4'. On the left sidebar, the 'Console Output' link is highlighted. The main content area is titled 'Console Output' with a blue sphere icon. The log text shows the job was started by a GitHub push from 'mikaelkrief'. It details the build process in the workspace, including git commands for parsing, fetching, and checking out a revision. The repository URL is 'https://github.com/mikaelkrief/MyShuttle2.git'.

Jenkins > demoCI > #4

 Back to Project

 Status

 Changes


 **Console Output**


 View as plain text

 Edit Build Information

 Delete build '#4'

 Polling Log

 Git Build Data

 No Tags

Console Output

Started by GitHub push by mikaelkrief
Building in workspace /var/lib/jenkins/workspace/demoCI
No credentials specified

```
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/mikaelkrief/MyShuttle2.git # timeout=10
Fetching upstream changes from https://github.com/mikaelkrief/MyShuttle2.git
> git --version # timeout=10
> git fetch --tags --progress https://github.com/mikaelkrief/MyShuttle2.git +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 475f4117a5e4d4892427e77104875f7f25ab0734 (refs/remotes/origin/master)
```


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:

The screenshot displays the Azure DevOps interface. On the left sidebar, the 'Repos' tab is selected, indicated by a red circle with the number 1. The top navigation bar shows the 'BookDemo' repository selected, indicated by a red circle with the number 2. A dropdown menu is open, showing options for repository management, with 'Import repository' highlighted, indicated by a red circle with the number 3. The main content area shows the 'Contents' tab for the 'BookDemo' repository, displaying a file named 'Readme.md'.

Azure DevOps BookLabs / BookDemo / Repos / Files / BookDemo 2

Search

BookDemo +

Feature1 BookDemo / Type to find

Filter repositories

BookDemo

+ New repository

↑ Import repository 3

⚙️ Manage repositories

Code explorer

Contents History README + New

Name ↑ Last

m+ Readme.md 8/21

Readme.md Mikael Krief

Set up build

Using Azure Pipelines

- Versioning of the code with Git in Azure Repos:

Import a Git repository



Source type

Git



Clone URL *

https://github.com/mikaelkrief/DemoAspNetApp.git

1



Requires authorization

Name *

DemoAspNetApp

2

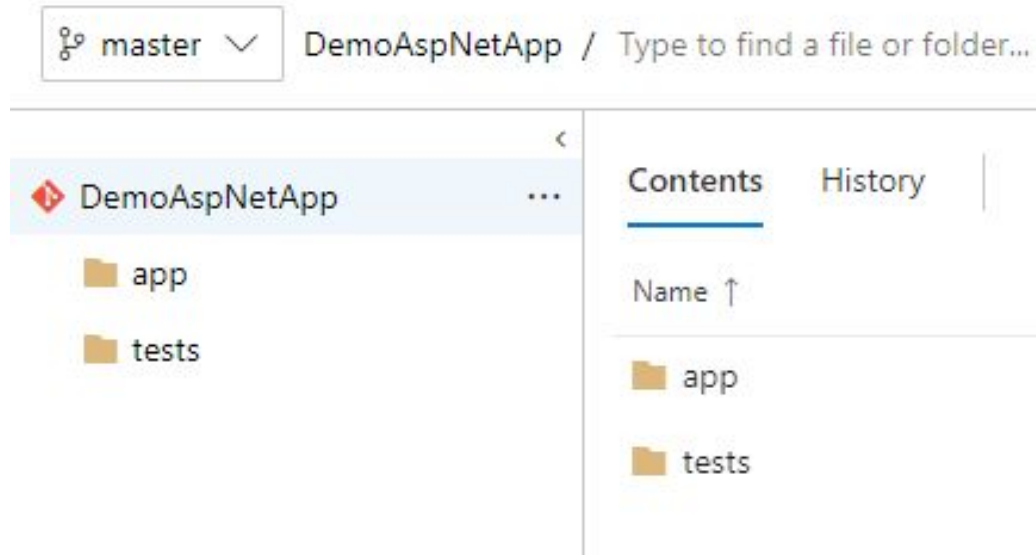
Import

Close

Using Azure Pipelines

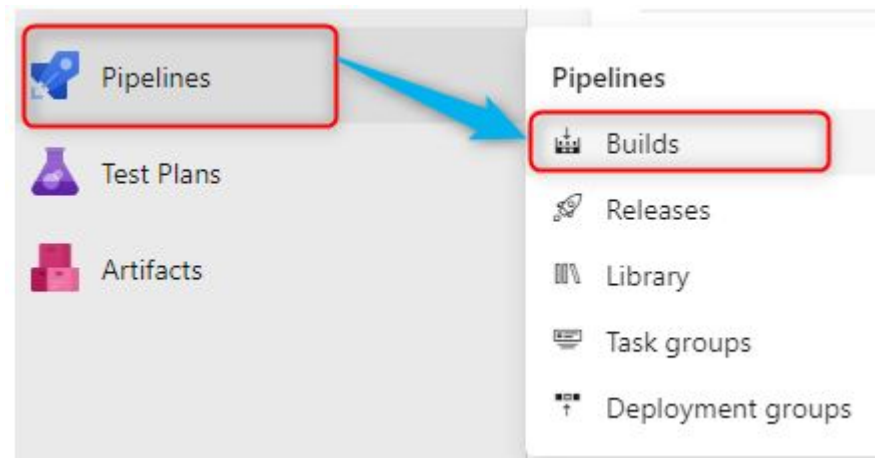
- **Versioning of the code with Git in Azure Repos:**

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.

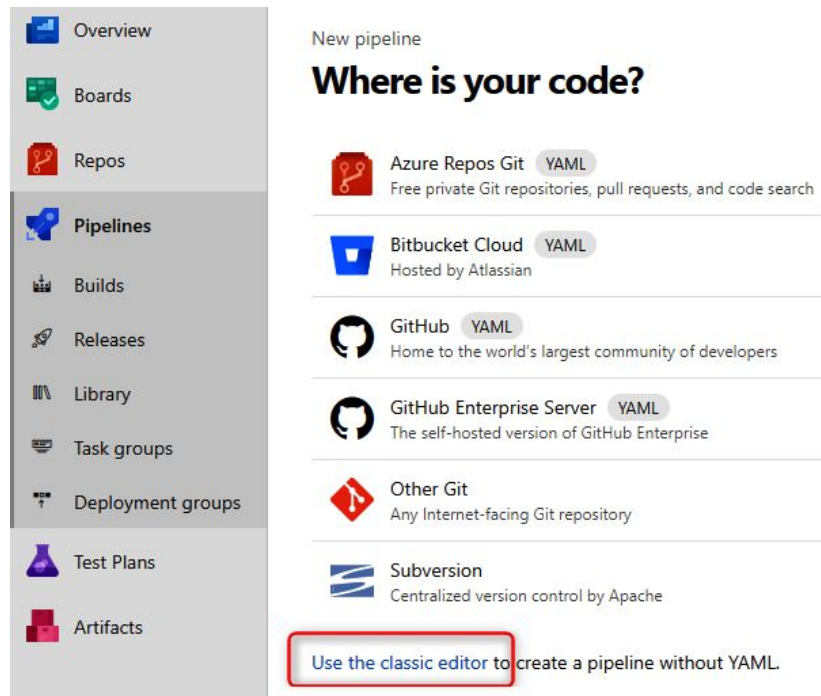


Using Azure Pipelines

- **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.

Select a source



Team project

BookDemo

Repository

DemoAspNetApp

Default branch for manual and scheduled builds

master

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.

Select a template

Or start with an  **Empty job**



Create and test a Python package on multiple Python versions.



Xcode

Build, test, archive, or package an Xcode workspace on macOS.

Others



Ant

Build and test a Java project with Apache Ant.



ASP.NET Core

Build and test an ASP.NET Core web application.

Apply



ASP.NET Core (.NET Framework)

Build an ASP.NET Core web application that targets the full .NET Framework.

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.

🏠 ... > BookDemo-ASP.NET Core-CI

Tasks **Variables** Triggers Options Retention History | Save & queue Discard Summary Queue ...

Pipeline variables

Variable groups

Predefined variables [🔗](#)

Name ↑	Value		Settable at queue time
BuildConfiguration	Release		
BuildPlatform	any cpu		
system.collectionId	76c79aec-9641-44c5-be15-beacfafe67a9		
system.debug	false		
system.definitionId	1		
system.teamProject	BookDemo		

Add

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.

Tasks Variables Triggers Options Retention History | Save & queue Discard Summary

Pipeline
Build pipeline

Get sources
DemoAspNetApp master

Agent job 1
Run on agent

Agent pool * | Pool information | Manage

Azure Pipelines

Name *
DemoCICD-ASP.NET Core-CI

1

2

3

Tasks Variables Triggers Options Retention History | Save & queue

Pipeline
Build pipeline

Get sources
DemoAspNetApp master

Agent job 1
Run on agent

Restore .NET Core

Build .NET Core

Test .NET Core

Publish .NET Core

Publish Artifact
Publish Build Artifacts

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:

The screenshot shows the 'Get sources' configuration page in Azure Pipelines. The left sidebar contains a list of tasks: 'Get sources' (selected and highlighted with a red box), 'Agent job 1', 'Restore', 'Build', 'Test', and 'Publish'. The main area is titled 'Select a source' and shows the 'Azure Repos Git' source selected. Below this, the 'Team project' is set to 'BookDemo', the 'Repository' is 'DemoAspNetApp', and the 'Default branch for manual and scheduled builds' is 'master'.

Tasks Variables Triggers Options Retention History | Save & queue Discard Summary Queue ..

Pipeline
Build pipeline

Get sources
DemoAspNetApp master

Agent job 1
Run on agent

Restore
.NET Core

Build
.NET Core

Test
.NET Core

Publish
.NET Core

Select a source

Azure Repos Git GitHub GitHub Enterprise Server Subversion

Team project
BookDemo

Repository
DemoAspNetApp

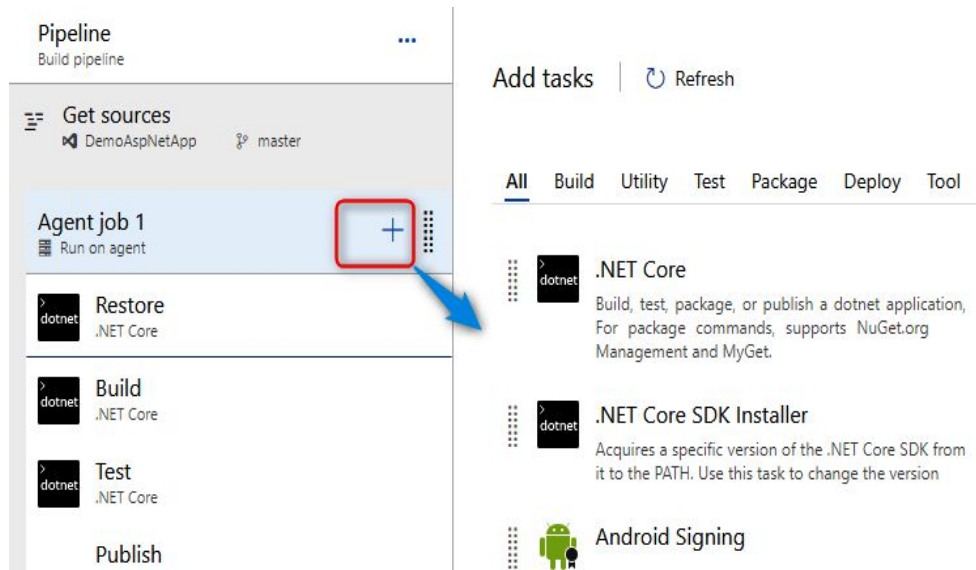
Default branch for manual and scheduled builds
master

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

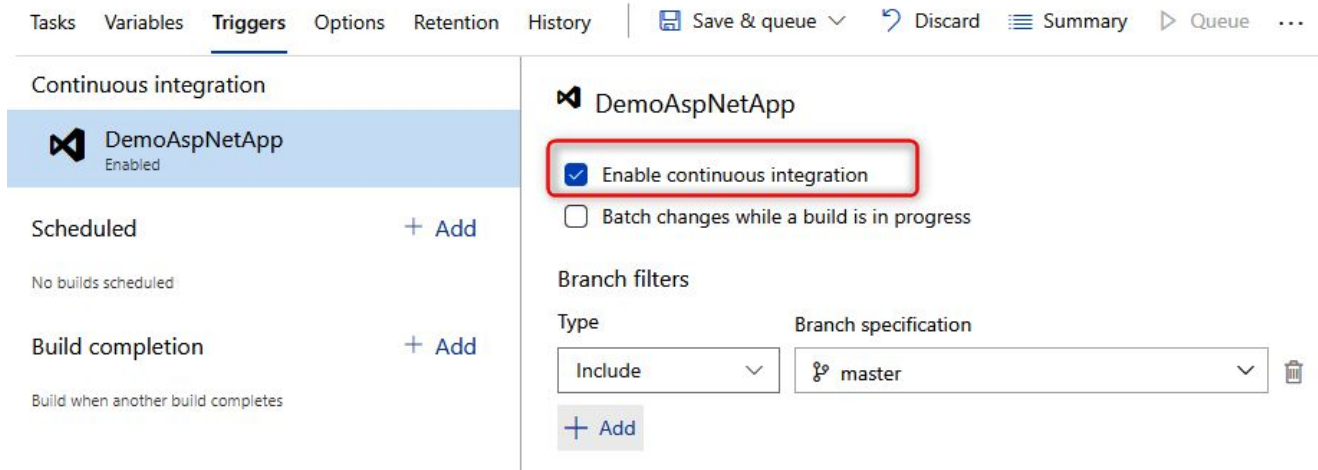
- **Creating the CI pipeline:**
- The five tasks in the CI pipeline:

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, <i>Use package manager</i> , 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:**

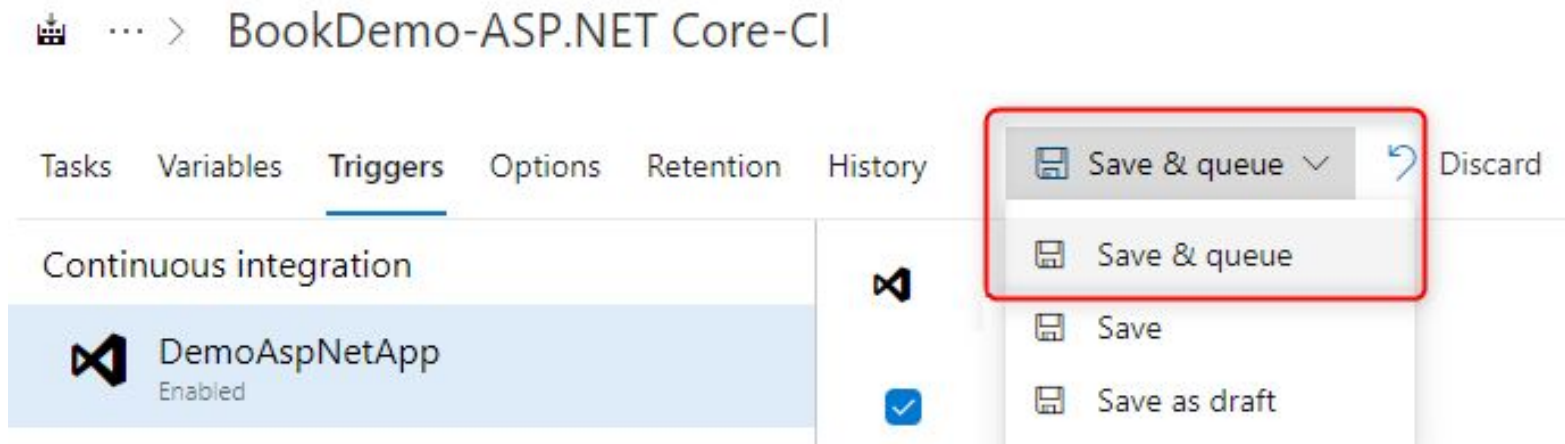
10. The last important configuration of CI pipeline is the configuration of the build trigger in the Triggers tab to enable continuous integration:

The screenshot shows the 'Triggers' tab in the Azure Pipelines interface. The top navigation bar includes 'Tasks', 'Variables', 'Triggers' (selected), 'Options', 'Retention', and 'History'. Action buttons include 'Save & queue', 'Discard', 'Summary', 'Queue', and a menu icon. The left sidebar lists trigger types: 'Continuous integration' (selected), 'Scheduled', and 'Build completion'. Under 'Continuous integration', the 'DemoAspNetApp' trigger is listed as 'Enabled'. The main panel shows the configuration for 'DemoAspNetApp'. A red rectangle highlights the 'Enable continuous integration' checkbox, which is checked. Below it is an unchecked checkbox for 'Batch changes while a build is in progress'. The 'Branch filters' section shows 'Type' set to 'Include' and 'Branch specification' set to 'master'. There is an '+ Add' button at the bottom of the branch filters section.

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:

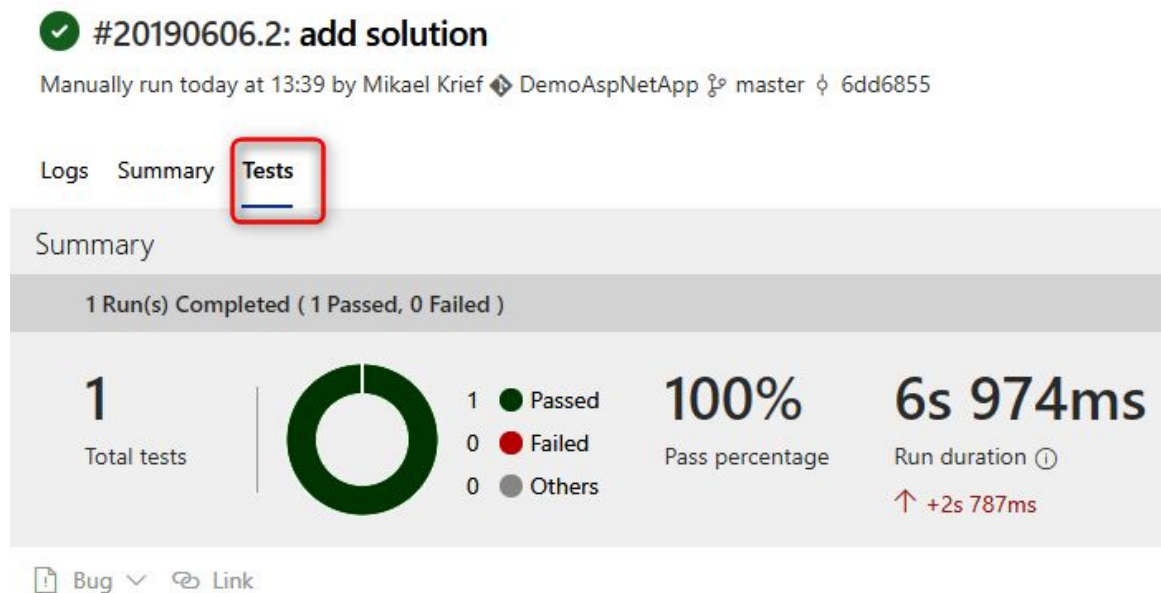
- The following screenshot shows the execution logs:

The screenshot displays the Azure Pipelines interface for a build named **#20190606.2: add solution**. The build was manually run at 13:39 by Mikael Krief on the `DemoAspNetApp` branch, commit `6dd6855`. The **Logs** tab is selected, showing the details for **Agent job 1**, which ran on a **Hosted Ubuntu 1604** agent. The job status is **succeeded**. The log lists the following steps, all of which completed successfully:

- Prepare job · succeeded
- Initialize job · succeeded
- Checkout · succeeded
- Restore · succeeded
- Build · succeeded
- Test · succeeded
- Publish · succeeded
- Publish Artifact · succeeded
- Post-job: Checkout · succeeded
- Finalize Job · succeeded
- Report build status · succeeded

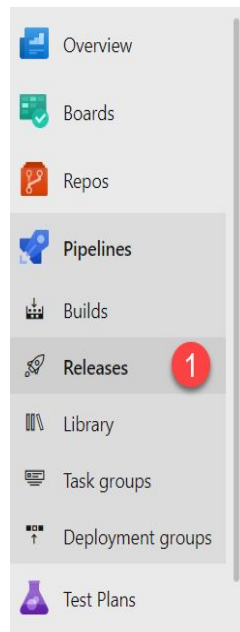
Using Azure Pipelines

- 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:



No release pipelines found

Automate your release process in a few easy steps with a new pipeline

New pipeline

2

Creating the CD Pipeline: The Release

/ BookDemo / Pipelines

All pipelines > New release pipeline

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

+ Add an artifact

Schedule not set

Stages | + Add

Stage 1
Select a template

Select a template

Or start with an Empty job

Search

Featured



Azure App Service deployment

Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.



Deploy a Java app to Azure App Service

Deploy a Java application to an Azure Web App.



Deploy a Node.js app to Azure App Service

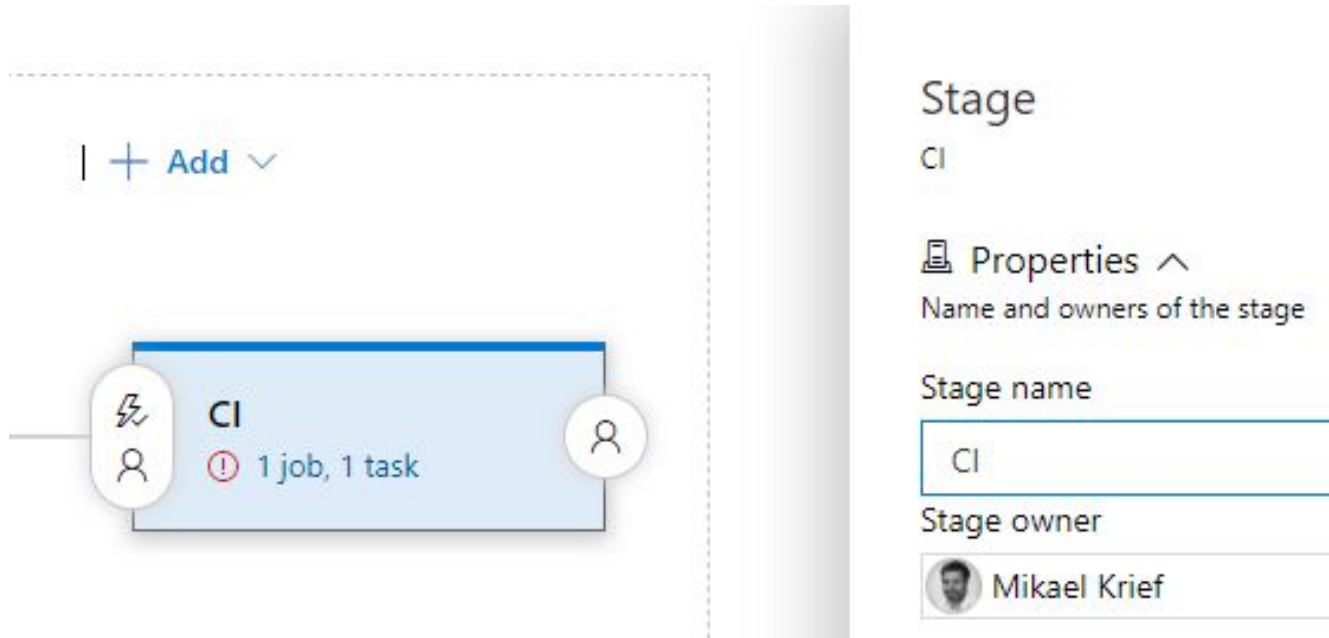
Deploy a Node.js application to an Azure Web App.



Deploy a PHP app to Azure App Service and Azure Database for MySQL

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:



The screenshot displays the Azure DevOps interface for creating a new stage in a pipeline. On the left, a blue stage box is shown with a lightning bolt icon, a person icon, and the text "CI" and "1 job, 1 task". Above the stage box is a button labeled "+ Add". To the right, the "Stage" configuration panel is visible, showing the stage name "CI" and the stage owner "Mikael Krief".

Stage

CI

Properties ^

Name and owners of the stage

Stage name

CI

Stage owner

Mikael Krief

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:

The screenshot shows the 'Add an artifact' configuration page in Azure DevOps. The left pane displays the 'Artifacts' section with a red circle '1' over the 'Add an artifact' button. The right pane shows the configuration for adding an artifact, with a red circle '2' over the 'Source (build pipeline)' dropdown. The 'Project' dropdown is set to 'BookDemo'. The 'Source (build pipeline)' dropdown is set to 'BookDemo-ASP.NET Core-CI'. The 'Default version' dropdown is set to 'Latest'. The 'Source alias' dropdown is set to '_BookDemo-ASP.NET Core-CI'. A red circle '3' is over the 'Add' button at the bottom.

Artifacts | + Add

1 Add an artifact

Schedule not set

2

Add an artifact

Source type

Build Azure Repos ... GitHub TFVC

5 more artifact types v

Project * ⓘ

BookDemo v

Source (build pipeline) * ⓘ

BookDemo-ASP.NET Core-CI v

Default version * ⓘ

Latest v

Source alias * ⓘ

_BookDemo-ASP.NET Core-CI

ⓘ The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of **BookDemo-ASP.NET Core-CI** published the following artifacts: **drop**.

3 Add

Creating the CD Pipeline: The Release

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

All pipelines > New release pipeline

Save Create release ...

Pipeline Tasks Variables Retention Options History

Artifacts | + Add

_BookDemo-ASP.NET Core-CI

Continuous deployment trigger

Build: _BookDemo-ASP.NET Core-CI

Enabled
Creates a release every time a new build is available.

Build branch filters ⓘ

No filters added.

+ Add | v

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

The screenshot shows the 'New release pipeline' configuration interface in Azure DevOps. The 'CI' stage is selected and highlighted with a red box. The configuration panel on the right shows the 'Parameters' section with the following fields:

- Stage name:** CI
- Parameters:**
 - Azure subscription ***: A dropdown menu with a refresh button. Below it, a note states: 'Scoped to subscription 'DEMO'. This field is linked to 1 setting in 'Deploy Azure App Service'.
 - App type:** A dropdown menu with 'Web App on Windows' selected.
 - App service name ***: A dropdown menu with 'demobook-ci' selected.

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:

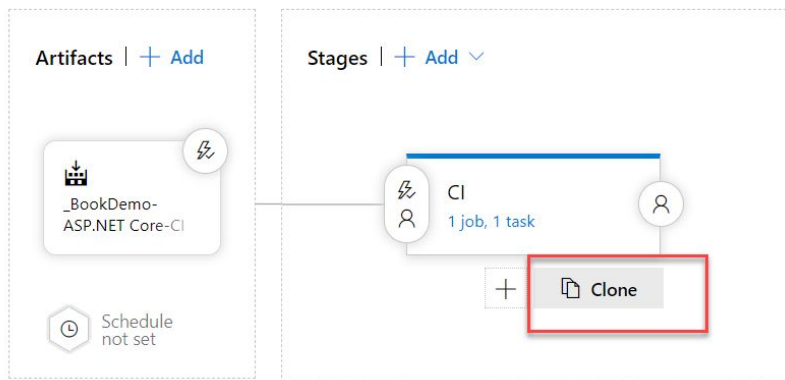
The screenshot shows the Jenkins web interface for a pipeline named 'DeployApp CD'. The breadcrumb navigation at the top reads 'BookLabs / BookDemo / Pipelines'. To the right of the breadcrumb is a dark button labeled 'Edit pipeline name' and a search bar. Below the breadcrumb, the pipeline name 'DeployApp CD' is displayed with a red box around it and a red circle with the number '1' next to it, indicating where to click to edit the name. To the right of the pipeline name are 'Save' and 'Create' buttons, with a red circle and the number '2' next to the 'Save' button, indicating where to click to save the changes. Below the pipeline name, there are tabs for 'Pipeline', 'Tasks', 'Variables', 'Retention', 'Options', and 'History'. The 'Tasks' tab is selected. On the left, under the 'Tasks' tab, there is a section for 'CI' with the description 'Deployment process' and a plus sign to add more tasks. On the right, there is a form for 'Stage name' with the value 'CI' and a 'Parameters' section with a link to 'Unlink all'.

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:

[All pipelines](#) > [DeployApp CD](#)

Pipeline Tasks Variables Retention Options History

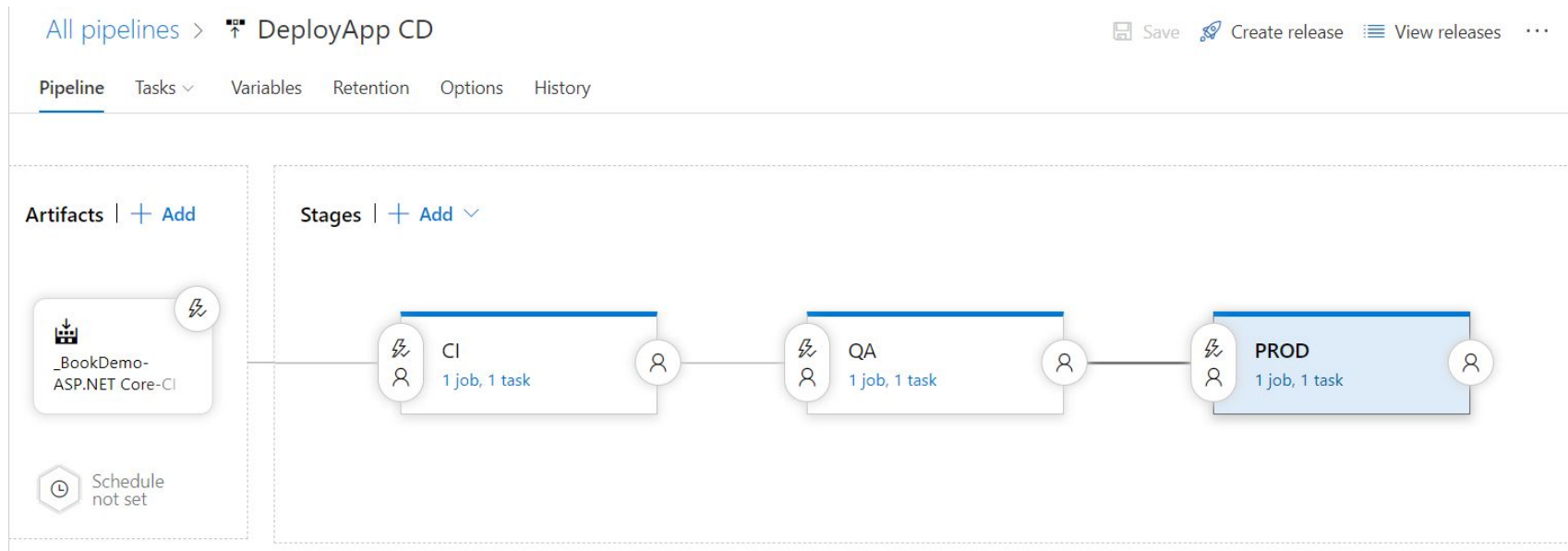


Pipeline Tasks Variables Retention Options History

The screenshot shows the configuration page for the 'QA' stage. The 'Stage name' is set to 'QA'. The 'Parameters' section shows 'Azure subscription' with a value 'Scoped to subscription 'DEMO''. The 'App type' is set to 'Web App on Windows'. The 'App service name' is set to 'demobook-qa'. The 'Clone' button is highlighted with a red box.

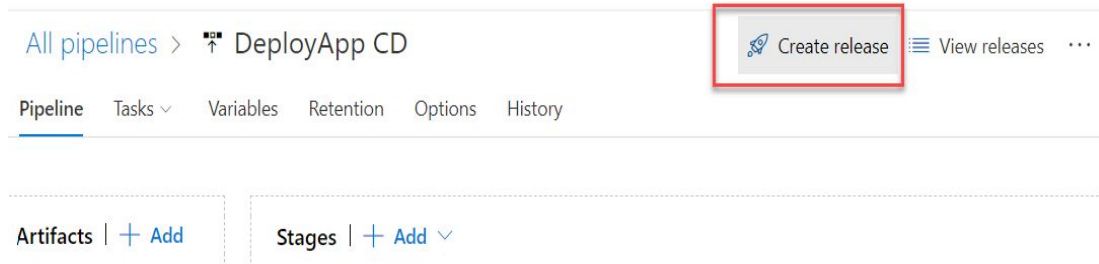
Creating the CD Pipeline: The Release

9. We finally get the release definition as follows:

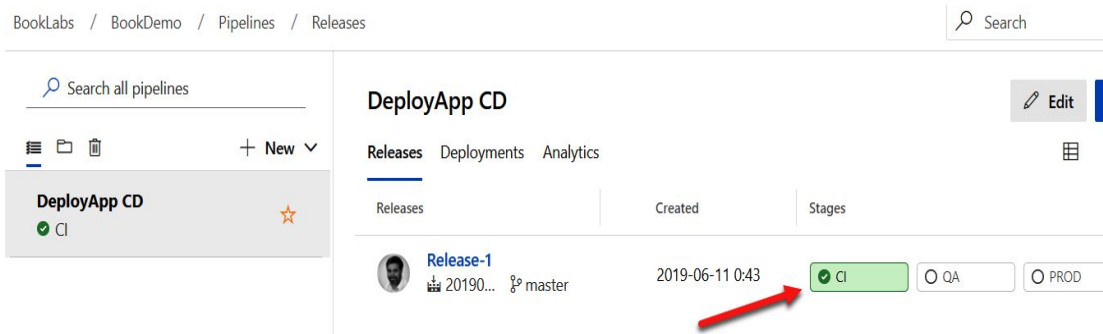


Creating the CD Pipeline: The Release

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.

GitLab.com

GitLab.com offers free unlimited (private) repositories and unlimited collaborators.

- [Explore projects on GitLab.com](#) (no login needed)
- [More information about GitLab.com](#)
- [GitLab.com Support Forum](#)
- [GitLab Homepage](#)

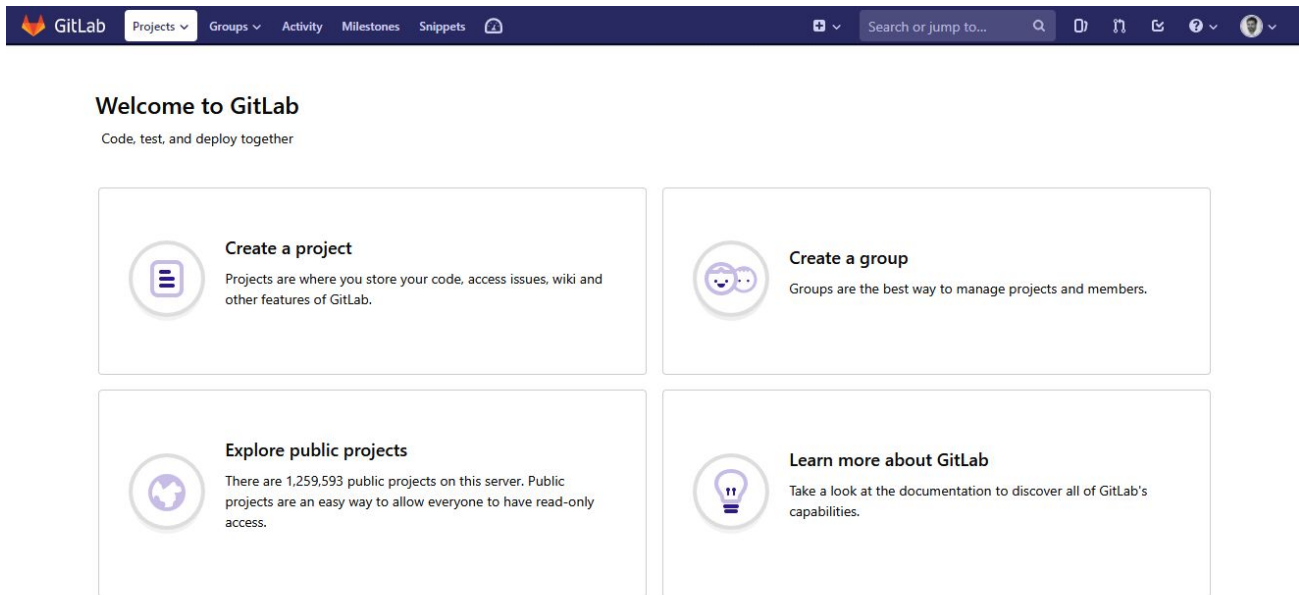
By signing up for and by signing in to this service you accept our:

- [Privacy policy](#)
- [GitLab.com Terms](#).

The screenshot shows the GitLab.com authentication page. At the top, there are two tabs: 'Sign in' (active) and 'Register'. Below the tabs, there are two input fields: 'Username or email' and 'Password', both with eye icons for toggling visibility. A 'Remember me' checkbox is located below the password field, and a 'Forgot your password?' link is to its right. A large green 'Sign in' button is positioned below these fields. Underneath the sign-in section, there is a 'Sign in with' section featuring five buttons for external authentication: Google, Twitter, GitHub, Bitbucket, and Salesforce. A 'Remember me' checkbox is also present at the bottom of this section.

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:

Welcome to GitLab

Code, test, and deploy together



Create a project

Projects are where you store your code, access issues, wiki and other features of GitLab.



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:

Blank project

Create from template

Import project

CI/CD for external repo

Project name

BookDemo

Project URL

https://gitlab.com/mikakrief/


Project slug


bookdemo

Want to house several dependent projects under the same namespace? [Create a group.](#)


Project description (optional)

Description format


Visibility Level 

☒  Private

Project access must be granted explicitly to each user.

☐  Internal

The project can be accessed by any logged in user.

☐  Public

The project can be accessed without any authentication.

☐ **Initialize repository with a README**

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project

Cancel

Creating a New Project and Managing Code Source

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

Blank project

Create from template

Import project


CI/CD for external repo

Learn how to [contribute to the built-in templates](#)

Built-in 17


Instance 0

Group 0

 **Ruby on Rails**
Includes an MVC structure, Gemfile, Rakefile, along with many others, to help you get started.


Preview

Use template

 **Spring**
Includes an MVC structure, mvnw and pom.xml to help you get started.


Preview

Use template

 **NodeJS Express**
Includes an MVC structure to help you get started.


Preview

Use template

 **iOS (Swift)**
A ready-to-go template for use with iOS Swift apps.

Preview

Use template

 **.NET Core**
A .NET Core console application template, customizable for any .NET Core project

Preview

Use template

Creating a New Project and Managing Code Source

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

The screenshot shows the 'Import project' tab selected in a navigation bar. Below the navigation bar, the text 'Import project from' is followed by a grid of buttons for different source repositories: GitLab export, GitHub, Bitbucket Cloud, Bitbucket Server, Google Code, Fogbugz, and Gitea. Below this grid are two more buttons: 'git Repo by URL' and 'Manifest file'.

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

The screenshot shows the 'CI/CD for external repo' tab selected in a navigation bar. Below the navigation bar, the heading 'Run CI/CD pipelines for external repositories' is followed by a paragraph explaining that connecting external repositories enables CI/CD pipelines. Below this is a link for 'More info'. At the bottom, under the heading 'Connect repositories from', there are two buttons: 'GitHub' and 'git Repo by URL'.

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:

The screenshot shows the GitLab interface for a project named 'BookDemo'. The project ID is 12804986. It has 1 commit, 1 branch, 0 tags, and 911 KB of files. The 'Auto DevOps' feature is highlighted, stating it will automatically build, test, and deploy the application based on a predefined CI/CD configuration. Below this, there's a section for the 'Initial commit' by Mikael KRIEF, authored 8 minutes ago. At the bottom, there are buttons to add README, CHANGELOG, CONTRIBUTING, a Kubernetes cluster, and set up CI/CD. A table lists the project's files and their commit history.

Name	Last commit	Last update
app	Initial commit	8 minutes ago
tests	Initial commit	8 minutes ago

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"

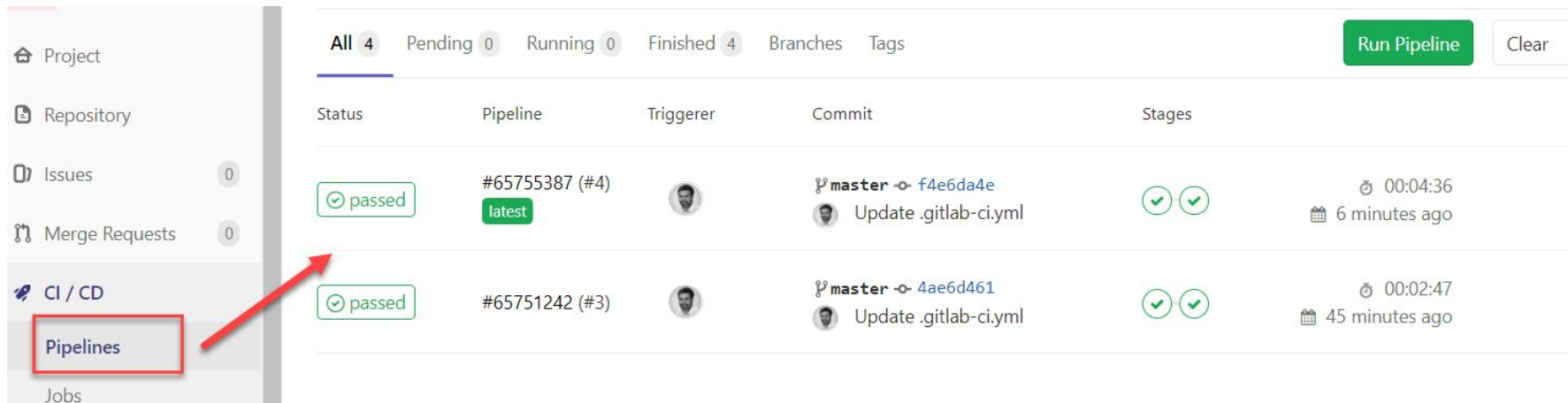
build:
  stage: build
  script:
    - "cd app"
    - "dotnet restore"
    - "dotnet build --configuration $BuildConfiguration"
test:
  stage: test
  script:
    - "cd tests"
    - "dotnet test --configuration $BuildConfiguration"
```

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:



The screenshot displays the GitLab CI Pipelines interface. On the left sidebar, the 'Pipelines' option under the 'CI / CD' section is highlighted with a red box and a red arrow. The main content area shows a table of pipeline executions with filters for 'All' (4), 'Pending' (0), 'Running' (0), and 'Finished' (4). The table lists two pipeline runs, both with a 'passed' status. The first pipeline is #65755387 (#4) for the 'latest' branch, triggered by a user, with commit f4e6da4e and a duration of 00:04:36. The second pipeline is #65751242 (#3) for the 'master' branch, triggered by a user, with commit 4ae6d461 and a duration of 00:02:47. Both pipelines show two successful stages.

Status	Pipeline	Triggerer	Commit	Stages	Duration	Time Ago
passed	#65755387 (#4) latest		master -> f4e6da4e Update .gitlab-ci.yml	✓ ✓	00:04:36	6 minutes ago
passed	#65751242 (#3)		master -> 4ae6d461 Update .gitlab-ci.yml	✓ ✓	00:02:47	45 minutes ago

Accessing the CI Pipeline Execution Details

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

The screenshot shows a GitLab CI/CD pipeline execution page. At the top, a green box with a checkmark and the word "passed" is displayed. To its right, the text "Pipeline #65755387 (#4) triggered 15 minutes ago by Mikael Krief" is shown. Below this, the title "Update .gitlab-ci.yml" is visible. Under the title, a clock icon indicates "2 jobs for master in 4 minutes and 36 seconds". A branch icon and a green box labeled "latest" are shown. Below that, a commit icon, the hash "f4e6da4e", a three-dot menu, and a copy icon are displayed. At the bottom, there are two tabs: "Pipeline" and "Jobs 2". The "Pipeline" tab is selected. Below the tabs, a horizontal flow diagram shows two stages: "Build" and "Test". The "Build" stage has a job named "build" with a green checkmark icon. The "Test" stage has a job named "test" with a green checkmark icon. Both jobs have a circular refresh icon to their right.

passed Pipeline #65755387 (#4) triggered 15 minutes ago by Mikael Krief

Update .gitlab-ci.yml

2 jobs for master in 4 minutes and 36 seconds

latest

f4e6da4e ...

Pipeline Jobs 2

Build Test

build test

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:

Mikael Krief > BookDemo > Jobs > #229390643

 **Job #229390643** triggered 19 minutes ago by  Mikael Krief

```
Running with gitlab-runner 11.11.2 (ac2a293c)
  on docker-auto-scale ed2dce3a
Using Docker executor with image microsoft/dotnet:latest ...
Pulling docker image microsoft/dotnet:latest ...
Using docker image sha256:08663b8eaa01a928bf4b22c6d7892a5306dc76a40d34e9449465d7f8d0c5ec38 for microsoft/dotnet:latest ...
Running on runner-ed2dce3a-project-12804986-concurrent-0 via runner-ed2dce3a-srm-1560285231-e19116a0...
Initialized empty Git repository in /builds/mikakrief/bookdemo/.git/
Fetching changes...
Created fresh repository.
From https://gitlab.com/mikakrief/bookdemo
* [new branch]      master    -> origin/master
Checking out f4e6da4e as master...

Skipping Git submodules setup
$ cd app
$ dotnet restore
Restore completed in 13.32 sec for /builds/mikakrief/bookdemo/app/app.csproj.
```



Thank You

