

Precondiciones:

1. Crear una Cuenta gratuita en Azure DevOps con tu cuenta de correo de Choucair
<https://dev.azure.com/>
2. Una vez finalizado el proceso de crear la cuenta, procedemos a crear un nuevo proyecto privado en Azure DevOps y cargar el código de una automatización que hayas construido y que estes seguro@ ejecuta exitosamente. Nota: Usar un proyecto personal como ejemplo

Create new project

✕


Project name *

RetoAutomatizador


Description

Reto automatizador Choucair Testing S.A

Visibility

 Public ⓘ

Anyone on the internet can view the project. Certain features like TFVC are not supported.

 Private

Only people you give access to will be able to view this project.

Public projects are disabled for your organization. You can turn on public visibility with [organization policies](#).

⌵ Advanced

Cancel

Create

3. Solicitar acceso gratuito a paralelismo de Azure DevOps, para esto llenar los datos que requieren en el siguiente formulario, una vez completado se debe esperar el correo de confirmación de la activación, el cuál tarda entre 12 a 48 horas.

<https://forms.office.com/pages/responsepage.aspx?id=v4j5cvGGr0GRgy180BHbR63mUWPlq7NEsFZhkyH8jChUMIM3QzdDMFZOMkVBWU5BWFM3SDI2QIRBSC4u>

* Obligatorio

1. What is your name? *

Escriba su respuesta

2. What is your email address? *

Escriba su respuesta

3. What is the name of your Azure DevOps Organization? *

(E.g. for <https://myorganization.visualstudio.com> or <https://dev.azure.com/myorganization> link formats - organization name would be 'myorganization')

Escriba su respuesta

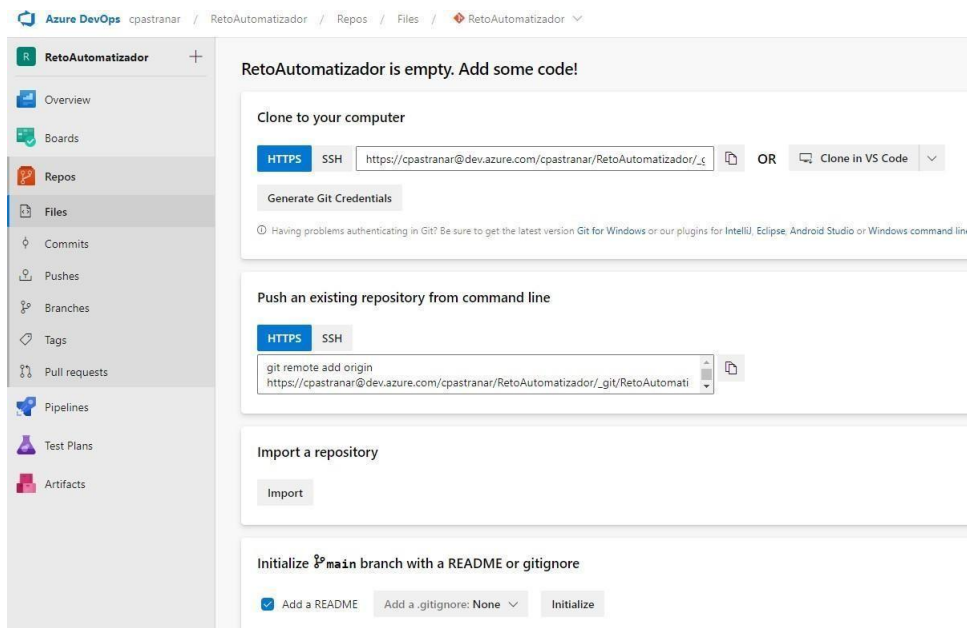
4. Are you requesting a parallelism increase for Public or Private projects? *

☒ Private

☐ Public

Enviar

4. Se debe crear un repositorio que en nuestro caso lo llamaremos “RetoAutomatizador”. Después de crear el proyecto, este se debería visualizar de la siguiente manera:

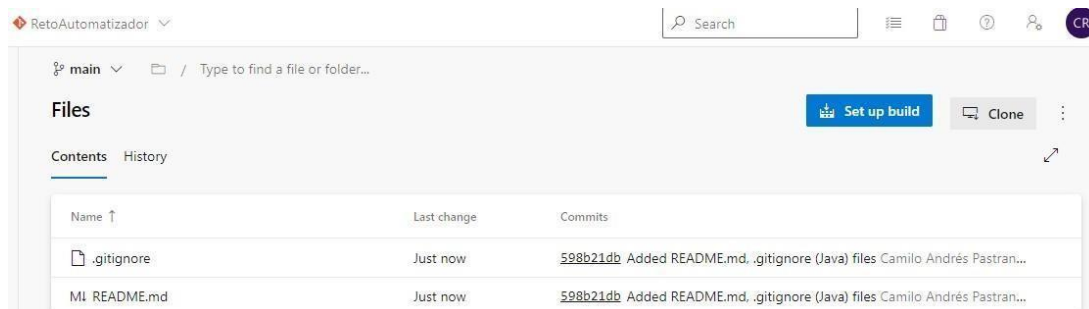


5. Ahora continuaremos con la configuración del README y el archivo gitignore los cuales se pueden visualizar al final de la página, para ello habilitaremos el check de “Add a README”, daremos clic en

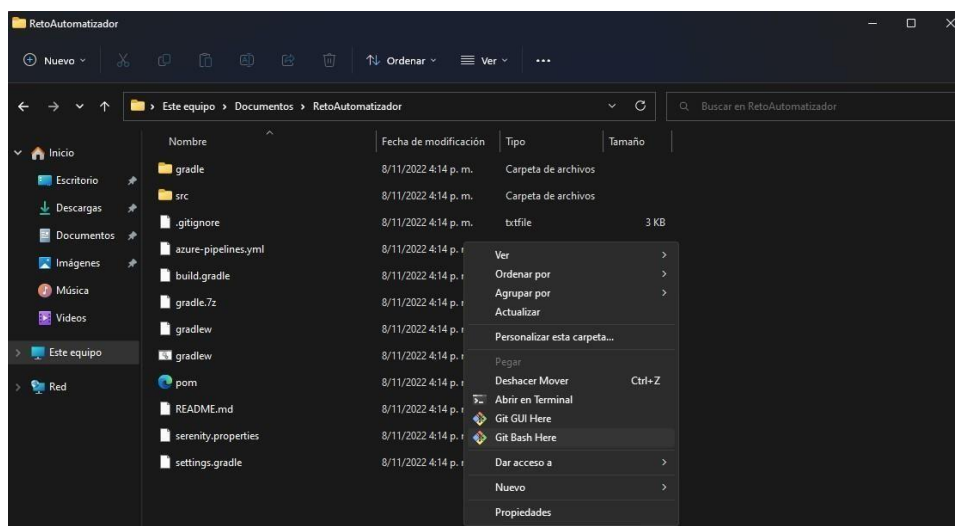


el botón “Add a .gitignore” y en el input buscaremos para palabra “Java” y por último damos clic al botón “Initialize”.

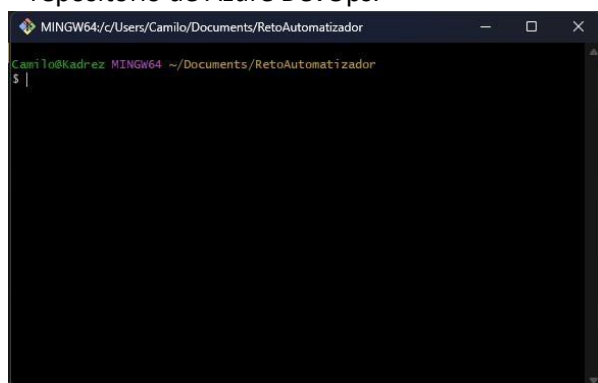
6. Luego de haber realizado el paso anterior, debemos visualizar lo siguiente:



7. A partir de este momento lo que debemos hacer es cargar el proyecto al repositorio de Azure a través de git. Para ello nos situamos en la carpeta de nuestro proyecto, presionamos clic derecho en un espacio vacío y seleccionamos Git Bash Here

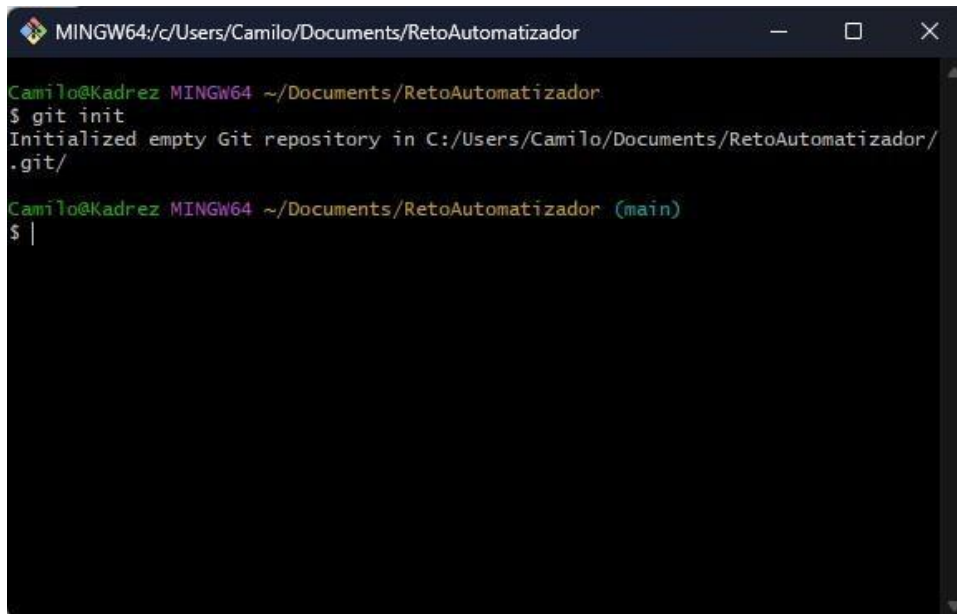


8. Desde esta consola, y a través de los comandos de git podremos subir nuestro proyecto al repositorio de Azure DevOps.



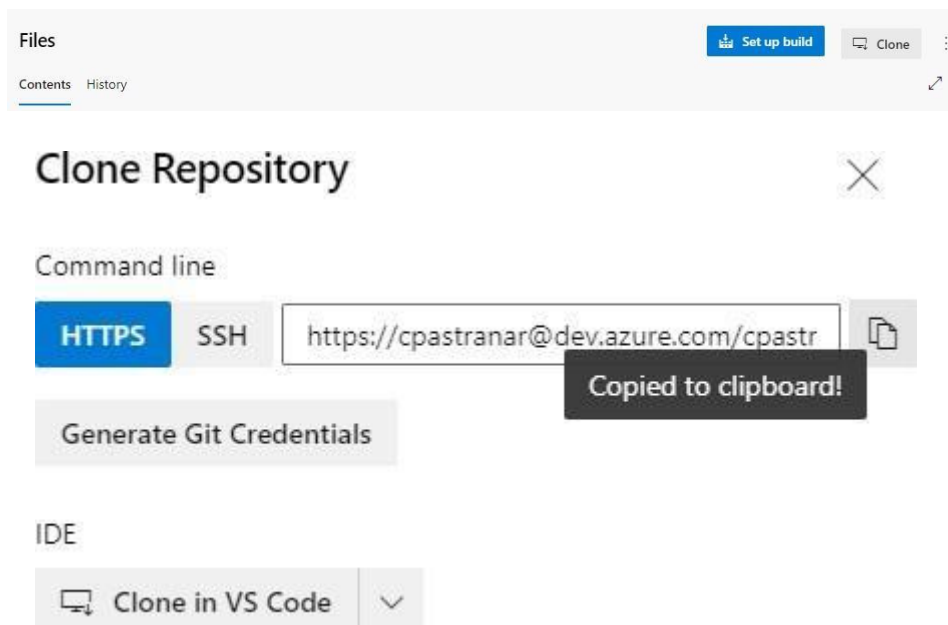
9. El primer comando que utilizaremos será: **“git init”**

Con este comando se crea un nuevo subdirectorio llamado “.git” que contiene todos los archivos necesarios del repositorio es decir un esqueleto de un repositorio Git.



```
MINGW64:/c/Users/Camilo/Documents/RetoAutomatizador
Camilo@Kadrez MINGW64 ~/Documents/RetoAutomatizador
$ git init
Initialized empty Git repository in C:/Users/Camilo/Documents/RetoAutomatizador/.git/
Camilo@Kadrez MINGW64 ~/Documents/RetoAutomatizador (main)
$ |
```

10. Seguidamente utilizaremos el comando: **“git remote add origin <url repositorio remoto>”** que será el encargado de conectarnos con el repositorio remoto de Azure, para esto volvemos a Azure a buscar el link para conectarnos al repositorio.



Vamos a dar clic al botón **“Clone”**, donde veremos la url que necesitamos, seguidamente damos clic en el botón **“Generate Git credentials”** y por último copiamos la url y volvemos a la consola de git.

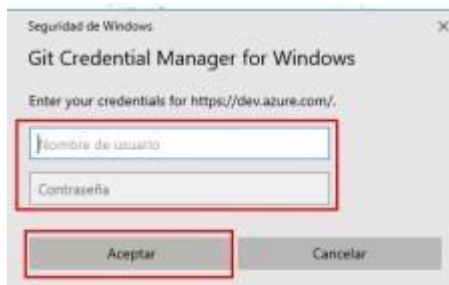


```
Camilo@Kadrez MINGW64 ~/Documents/RetoAutomatizador (main)
$ git remote add origin https://cpastranar@dev.azure.com/cpastranar/RetoAutomatizador/_git/RetoAutomatizador
```

Continuamos con el comando: **"git pull origin <rama>"**, con este comando vamos a bajar los dos archivos que tenemos en el repositorio (README y .gitignore).

```
Camilo@Kadrez MINGW64 ~/Documents/RetoAutomatizador (main)
$ git pull origin main
```

Una vez ejecutado este comando, nos solicitara credenciales las cuales fueron generadas cuando dimos clic al botón **"Generate Git credentials"** en Azure. Ingresamos el usuario y contraseña y damos clic en el botón aceptar.



Una vez ingresadas las credenciales, procedemos a verificar que en nuestro proyecto local se encuentren los dos archivos que estaban en el repositorio (README.md y .gitignore).

Nombre	Fecha de modificación	Tipo	Tamaño
.git	8/11/2022 4:48 p. m.	Carpeta de archivos	
gradle	8/11/2022 4:14 p. m.	Carpeta de archivos	
src	8/11/2022 4:14 p. m.	Carpeta de archivos	
.gitignore	8/11/2022 4:48 p. m.	txtfile	1 KB
azure-pipelines.yml	8/11/2022 4:14 p. m.	Archivo YML	1 KB
build.gradle	8/11/2022 4:14 p. m.	Archivo GRADLE	5 KB
gradle.7z	8/11/2022 4:14 p. m.	Archivo 7Z	53 KB
gradlew	8/11/2022 4:14 p. m.	Archivo	6 KB
gradlew	8/11/2022 4:14 p. m.	Archivo por lotes ...	3 KB
pom	8/11/2022 4:14 p. m.	Microsoft Edge H...	10 KB
README.md	8/11/2022 4:48 p. m.	Archivo MD	1 KB
serenity.properties	8/11/2022 4:14 p. m.	Archivo PROPERTI...	1 KB
settings.gradle	8/11/2022 4:14 p. m.	Archivo GRADLE	1 KB

Ahora procedemos a modificar el archivo “.gitignore” de nuestro proyecto local, lo vamos a abrir en un editor de texto y vamos a adicionar las siguientes líneas al final del mismo. Guardamos los cambios y cerramos el archivo.

```
.gradle
build/
# Ignore Gradle GUI config gradle-app.setting
# Avoid ignoring Gradle wrapper jar file (.jar files are usually ignored) !gradle-wrapper.jar
# Cache of project
.gradletasknameecache
```

También podremos modificar el archivo “README.md” por la información deseada. Una vez guardados los cambios volvemos nuevamente a la consola de git para subir nuestro proyecto automatizado con los últimos cambios realizados, para esto utilizaremos el comando: **“git add -A”** para preparar los archivos que vamos a subir.

```
Camilo@Kadrez MINGW64 ~/Documents/RetoAutomatizador (main)
$ git add -A
```

Luego usamos el comando: **“git commit -m “Descripción deseada””** con el preparamos todo y asignamos la descripción de los cambios que hemos realizado.

```
Camilo@Kadrez MINGW64 ~/Documents/RetoAutomatizador (main)
$ git commit -m "Reto automatizador completado"
```

Y por último utilizamos el comando: **“git push origin main”** para subir los archivos al repositorio de Azure DevOps

```
Camilo@Kadrez MINGW64 ~/Documents/RetoAutomatizador (main)
$ git push origin main
Enumerating objects: 57, done.
Counting objects: 100% (57/57), done.
Delta compression using up to 6 threads
Compressing objects: 100% (49/49), done.
Writing objects: 100% (54/54), 132.95 KiB | 5.78 MiB/s, done.
Total 54 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Analyzing objects... (54/54) (66 ms)
remote: Storing packfile... done (66 ms)
remote: Storing index... done (44 ms)
To https://dev.azure.com/cpastranar/RetoAutomatizador/_git/RetoAutomatizador
598b21d..9f1d772 main -> main
```

Una vez terminado el proceso, nos dirigimos al repositorio de Azure y al dar refresh a nuestra pantalla debemos visualizar todos los archivos ya cargados de nuestro proyecto.

The screenshot shows the Azure DevOps web interface for a repository named 'RetoAutomatizador'. The interface includes a search bar at the top right, a breadcrumb navigation path 'main' with a dropdown arrow, and a file explorer bar. Below the file explorer, there are tabs for 'Files' (selected) and 'History'. A 'Set up build' button is visible in the top right corner of the file view area. The main content is a table listing files and folders with columns for 'Name', 'Last change', and 'Commits'. The table shows a directory structure with folders 'gradle' and 'src', and various files including '.gitignore', 'azure-pipelines.yml', 'build.gradle', 'gradle.7z', 'gradlew', 'gradlew.bat', 'pom.xml', 'README.md', 'serenity.properties', and 'settings.gradle'. All files show a 'Just now' last change and a commit hash '9f1d7728' with the message 'Reto automatizador completado cpastranar'. At the bottom, there is a section for 'Reto Automatizador' with the description 'Reto técnico automatizador Choucair Testing S.A'.

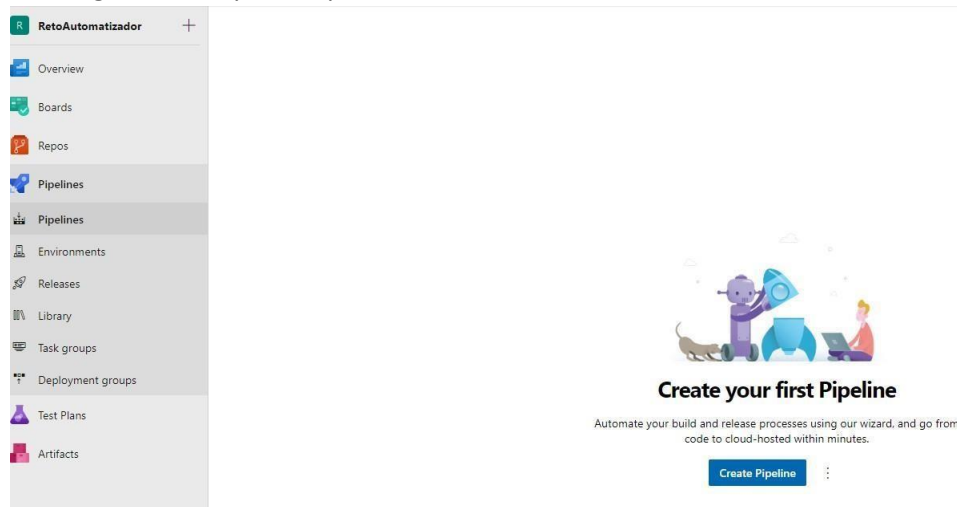
Name ↑	Last change	Commits
gradle	Just now	9f1d7728 Reto automatizador completado cpastranar
src	Just now	9f1d7728 Reto automatizador completado cpastranar
.gitignore	Just now	9f1d7728 Reto automatizador completado cpastranar
azure-pipelines.yml	Just now	9f1d7728 Reto automatizador completado cpastranar
build.gradle	Just now	9f1d7728 Reto automatizador completado cpastranar
gradle.7z	Just now	9f1d7728 Reto automatizador completado cpastranar
gradlew	Just now	9f1d7728 Reto automatizador completado cpastranar
gradlew.bat	Just now	9f1d7728 Reto automatizador completado cpastranar
</> pom.xml	Just now	9f1d7728 Reto automatizador completado cpastranar
MI README.md	Just now	9f1d7728 Reto automatizador completado cpastranar
serenity.properties	Just now	9f1d7728 Reto automatizador completado cpastranar
settings.gradle	Just now	9f1d7728 Reto automatizador completado cpastranar

Reto Automatizador
Reto técnico automatizador Choucair Testing S.A

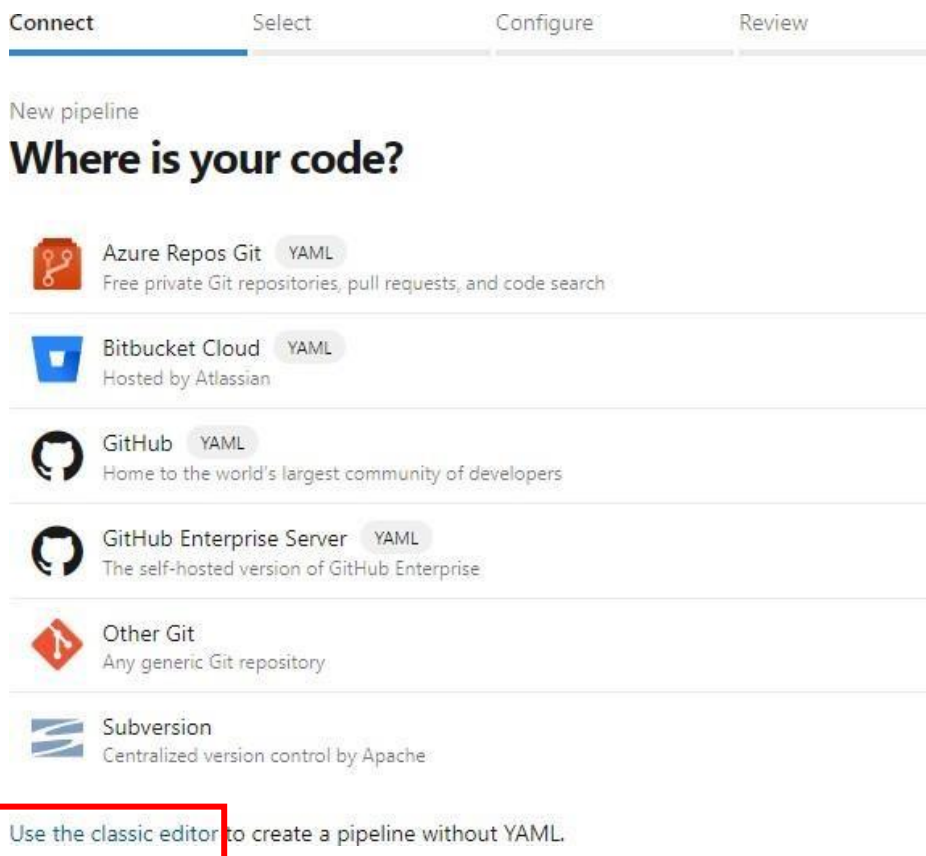
Con esto tendríamos lo necesario para crear el pipeline de CI/CD.

CREACIÓN PIPELINE CI/CD

Nos dirigimos a la opción Pipelines:

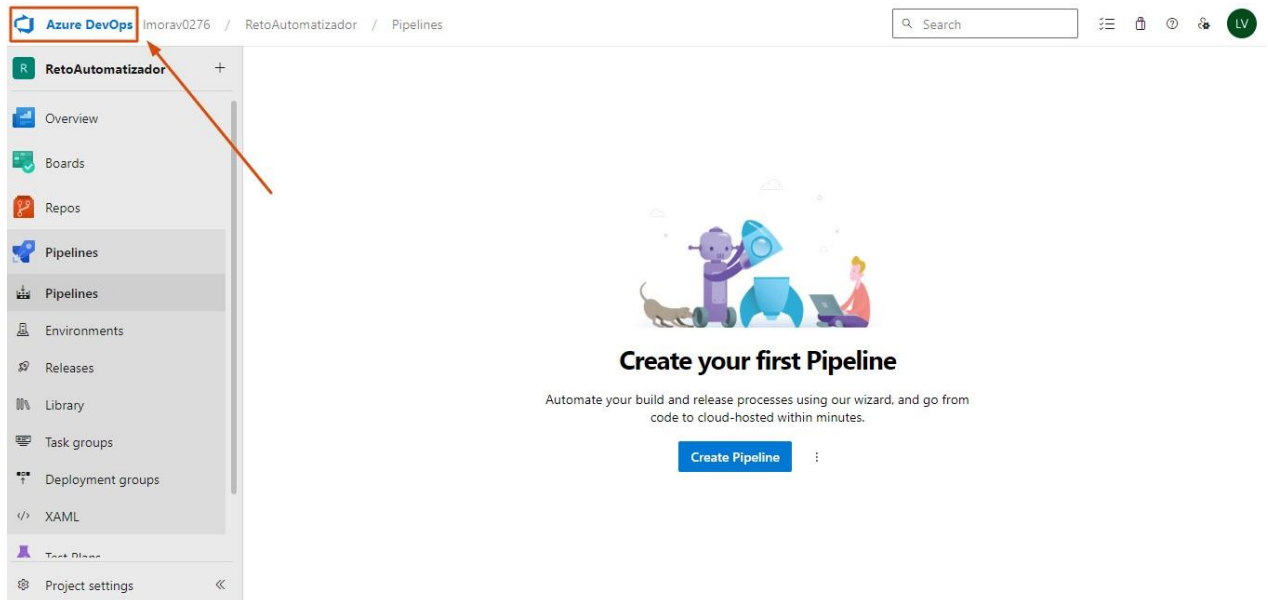


Pulsamos el botón **“Create Pipeline”** y procedemos a dar clic en **“Use the classic editor”**

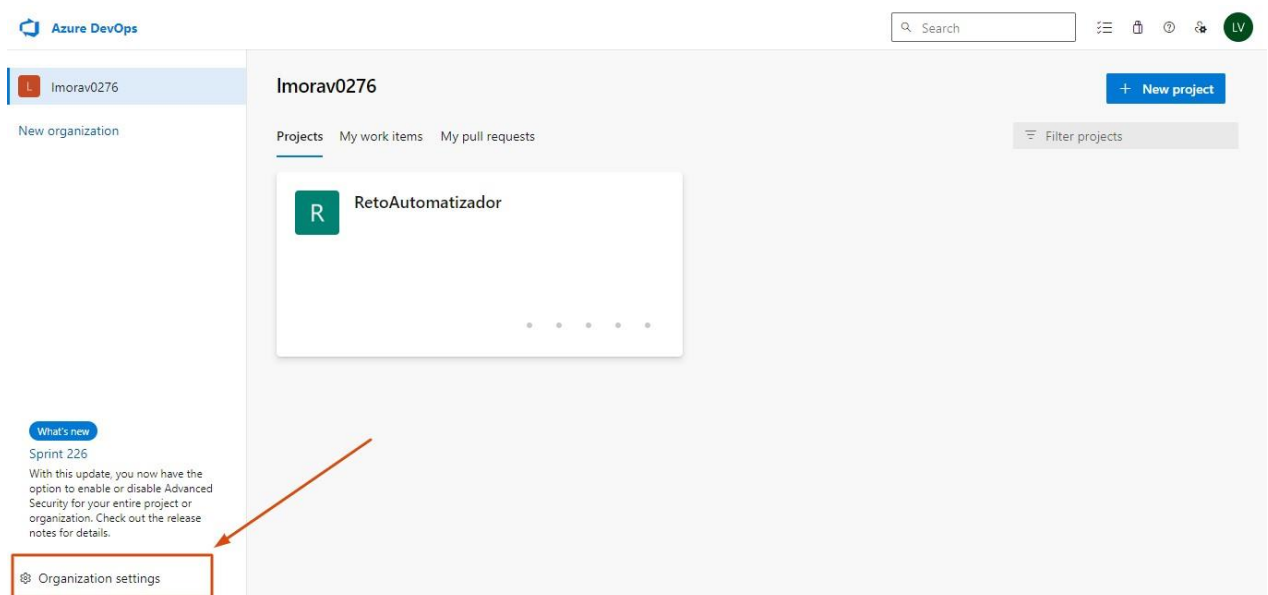


En caso de no tener la opción **“Use the classic editor”** habilitada realizamos la siguiente configuración de lo contrario omitir estos pasos.

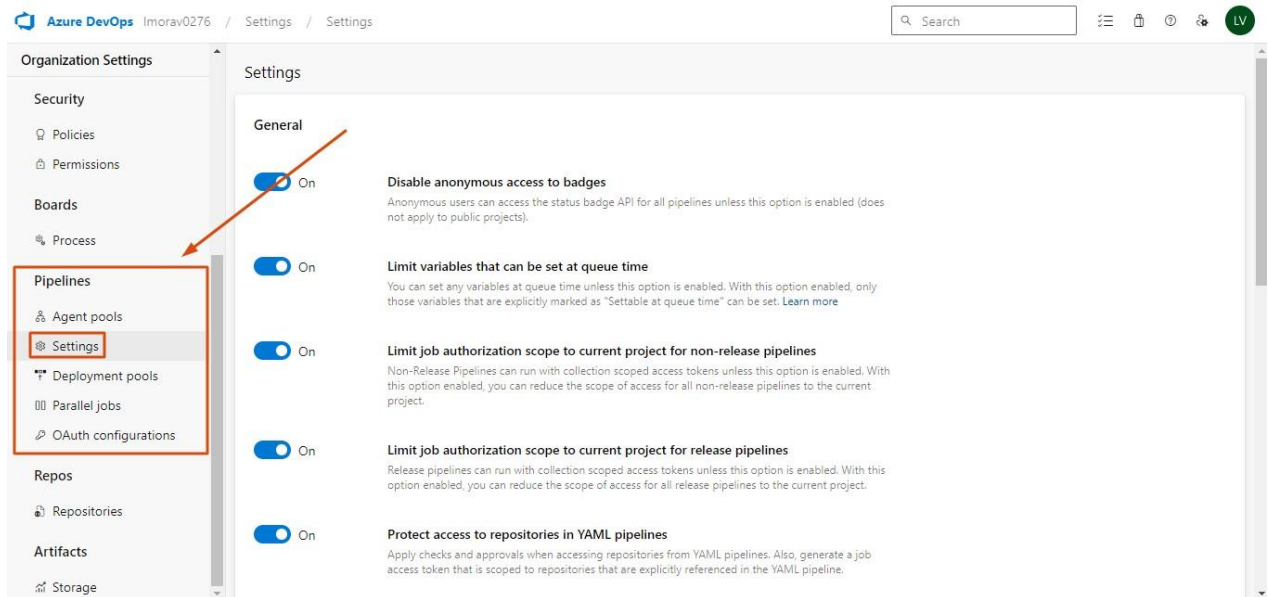
Damos clic en el logo Azure DevOps



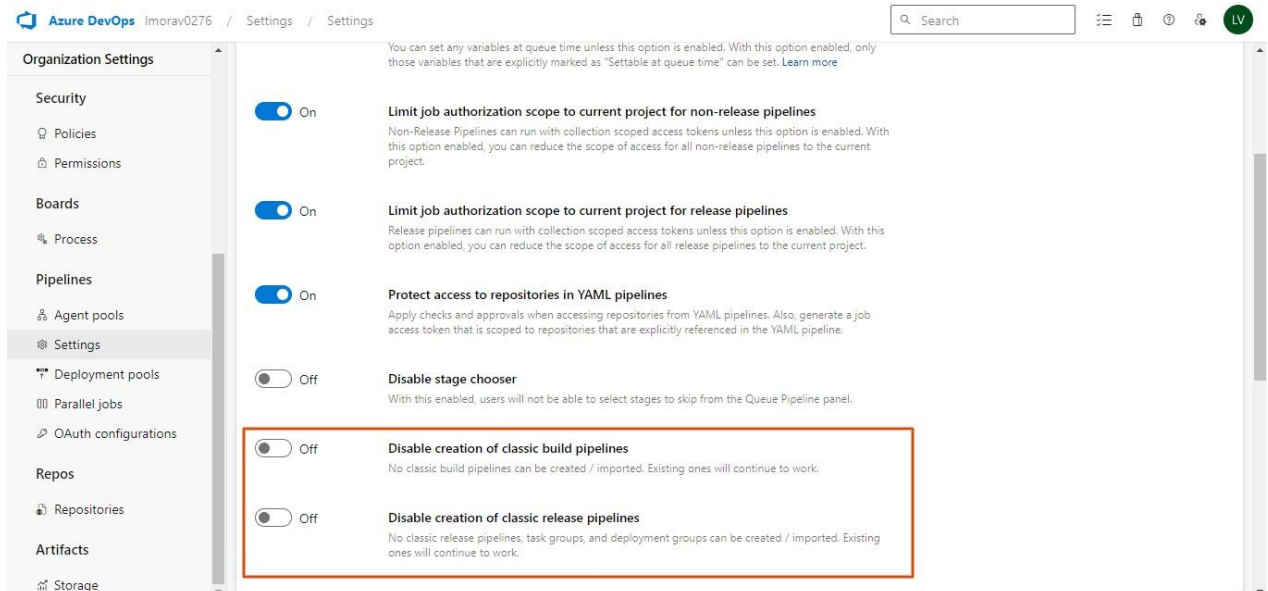
Nos dirigimos al apartado **“Organization Settings”**.



En el módulo de “PIPELINES” Seleccionamos el submódulo “Settings”.



Y deshabilitamos las siguientes dos opciones:





Una vez habilitada esta configuración continuamos con la creación del pipeline:


Connect Select Configure Review


New pipeline


Where is your code?


 **Azure Repos Git** YAML
Free private Git repositories, pull requests, and code search

 **Bitbucket Cloud** YAML
Hosted by Atlassian

 **GitHub** YAML
Home to the world's largest community of developers

 **GitHub Enterprise Server** YAML
The self-hosted version of GitHub Enterprise


 **Other Git**
Any generic Git repository


 **Subversion**
Centralized version control by Apache


[Use the classic editor](#) to create a pipeline without YAML.


Verificamos que estén correctamente seleccionados nuestro proyecto, repositorio y rama por defecto. Luego clic en **“Continue”**


Select a source


 Azure Repos Git

 GitHub


 GitHub Enterprise Server

 Subversion


 Bitbucket Cloud

 Other Git


Team project

 RetoAutomatizador

Repository

 RetoAutomatizador

Default branch for manual and scheduled builds

 main

Continue

Seleccionamos **“Empty Job”** para continuar

Select a template
Or start with an  Empty job

Search

Para este caso el nombre de nuestro pipeline de CI/CD será RetoAutomatizador

Name *

RetoAutomatizador

Agent pool ⓘ | Pool information | Manage ⓘ

Azure Pipelines

Agent Specification *


windows-2019

Nos dirigimos a la pestaña **“Triggers”** y marcamos la opción **“Enable continuous integration”**, verificamos que todas las ramas de nuestro repositorio se encuentren seleccionadas y volvemos a la pestaña **“Tasks”**

🏠 ... > RetoAutomatizador


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

Continuous integration

 RetoAutomatizador
Enabled

Scheduled + Add
No builds scheduled

Build completion + Add
Build when another build completes

 RetoAutomatizador

☒ Enable continuous integration
☐ Batch changes while a build is in progress

Branch filters

Type Branch specification

Include ▼ 📁 main

+ Add

Path filters

+ Add

Esta configuración se debe habilitar para el Pipeline de CI donde se habilitará la ejecución automática cuando se realicen “commits” en las distintas ramas.

1. develop
2. release
3. master

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

Continuous integration

Scheduled + Add
No builds scheduled

Build completion + Add
Build when another build completes

☒ Enable continuous integration
☐ Batch changes while a build is in progress

Branch filters

Type Branch specification

Include ▼ 📁 master

Include ▼ 📁 develop

Include ▼ 📁 release

+ Add

Path filters

+ Add

En el apartado de Clean en Get Sources debemos seleccionar la opción **“true”** y luego en clean options **“All build directories”**

The screenshot shows the 'Get sources' configuration panel in Azure Pipelines. The 'Clean' dropdown is set to 'true' and the 'Clean options' dropdown is set to 'All build directories'. The 'Team project' is 'RetoAutomatizador', the 'Repository' is 'RetoAutomatizador', and the 'Default branch' is 'main'.

En el apartado de Agent job 1, seleccionamos check box de Allow Scripts to access the oauth token

The screenshot shows the 'Agent job 1' configuration panel in Azure Pipelines. The 'Allow scripts to access the OAuth token' checkbox is checked and highlighted with a green box. The 'Display name' is 'Agent job 1', the 'Agent pool' is 'Azure Pipelines', and the 'Agent Specification' is 'windows-latest'. The 'Demands' table shows 'java' with condition 'exists'. The 'Execution plan' is 'None'. The 'Timeout' is '0' and the 'Job cancel timeout' is '0'. The 'Dependencies' are 'Select dependencies'. The 'Additional options' are 'Run this job' and 'Only when all previous jobs have succeeded'.

Nos dirigimos al apartado Agent job 1 y clic en el botón de “+” > Marketplace y buscamos SonarCloud e instalamos.

The screenshot displays the Azure Pipelines interface. On the left, the 'Pipeline' sidebar shows 'Get sources' and 'Agent job 1' with a red '+' button next to it. The main area is titled 'Add tasks' and includes a 'Refresh' button and a search bar containing 'sonarcloud'. Below the search bar, three tasks are listed: 'Publish Quality Gate Result', 'Prepare Analysis Configuration', and 'Run Code Analysis'. A 'Marketplace' section is also visible, featuring a green box around the 'SonarCloud' task, which is described as 'Detect bugs, vulnerabilities and code smells across project branches and pull requests.' Other tasks in the marketplace include 'SonarCloud build breaker' and 'Use .NET Core SonarCloud'.

Pipeline
Build pipeline

Get sources
Run on agent

Agent job 1
Run on agent

Add tasks | Refresh

Search

All Build Utility Test Package Deploy Tool Marketplace

Refresh

sonarcloud

Publish Quality Gate Result
Publish SonarCloud's Quality Gate result on the Azure Pipelines build result. To be used after the actual analysis.

Prepare Analysis Configuration
Prepare SonarCloud analysis configuration

Run Code Analysis
Run scanner and upload the results to the SonarCloud server.

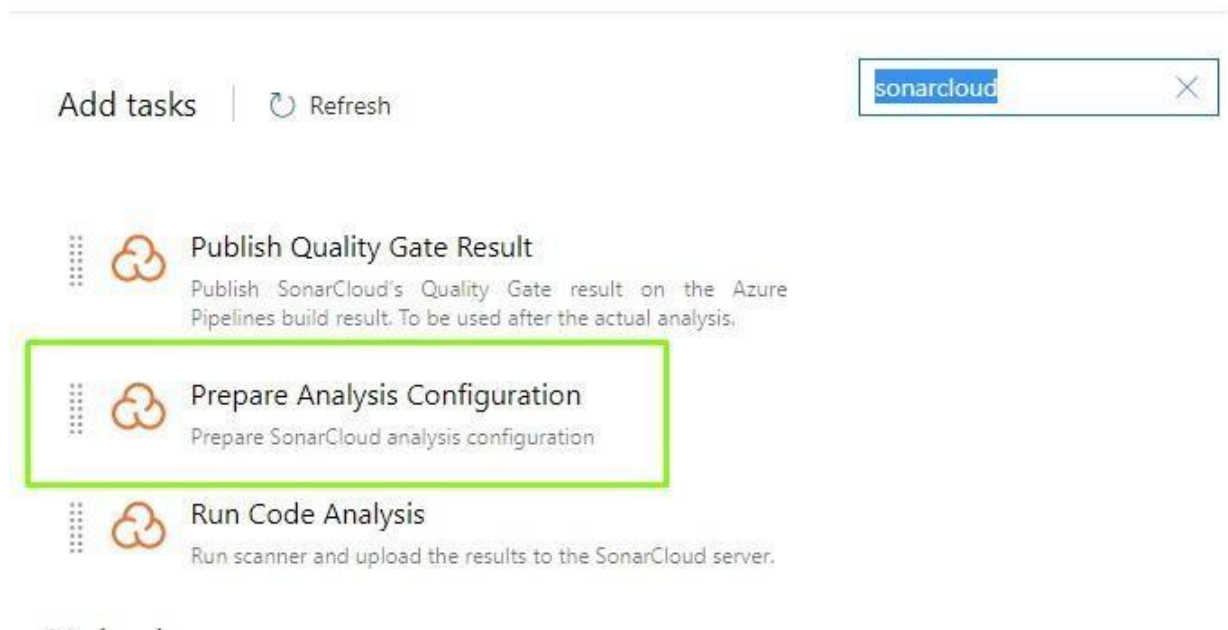
Marketplace ^

SonarCloud
Detect bugs, vulnerabilities and code smells across project branches and pull requests.

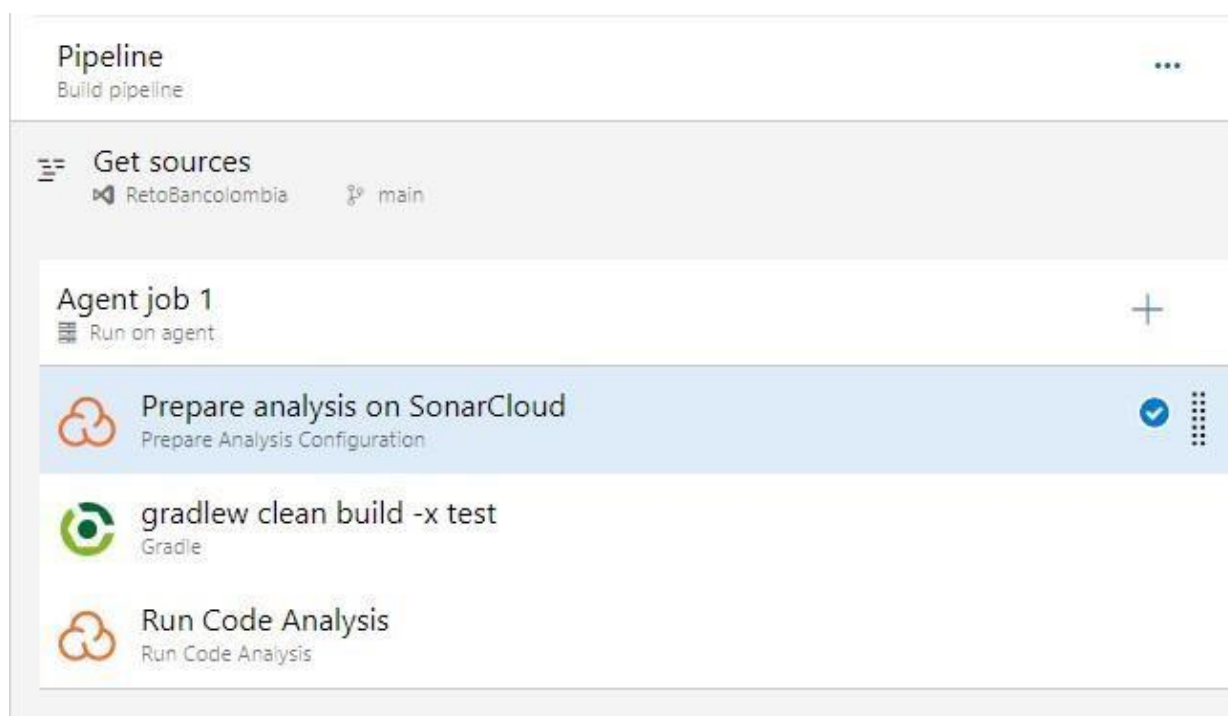
Quality Gate
SonarCloud build breaker
Task for breaking your build when it fails the SonarCloud quality gate.

SP Tools
Use .NET Core SonarCloud
Allows to run SonarCloud for .NET Core projects on on-premise build agents behind corporate proxy.

Continuando se agrega la nueva task (Prepare Analysis Configuration), buscamos y agregamos una a una como se muestra a continuación:



Debemos agregar las siguientes task:



Debemos configurar el servicio de SonarQube, para ello clic en la tarea **“Prepare analysis on SonarCloud”** y luego en SonarCloud Server Endpoint seleccionaremos **“New”**

Prepare Analysis Configuration ⓘ [Link settings](#) [View YAML](#) [Remove](#)

Task version 1.* ▼

Display name *
Prepare analysis on SonarCloud

SonarCloud Service Endpoint * ⓘ | [Manage](#) [↗](#)
▼ ↻ **+ New**

ⓘ This setting is required.

Organization * ⓘ
▼ ↻

ⓘ This setting is required.

Choose the way to run the analysis * ⓘ
☐ Integrate with MSBuild ☐ Integrate with Maven or Gradle ☒ Use standalone scanner

Mode * ⓘ
☐ Store configuration with my source code (sonar-project.properties) ☒ Manually provide configuration

Project Key * ⓘ
\$(Build.Repository.Name)

Project Name ⓘ
\$(Build.Repository.Name)

Project Version ⓘ
1.0

Sources directory root * ⓘ
... ⋮

Advanced ▼

Control Options ▼

Output Variables ▼

Token: <token a utilizar> Service connection name: sonartestch

Pasos para obtener el token de sonarCloud

1. Ingresa a sonarCloud <https://sonarcloud.io/explore/projects>
2. Ingresar con la cuenta GitHub
3. Crear una nueva organización
4. Crear un nuevo Proyecto
5. Ingresar al proyecto y seleccionar el apartado manual
6. Escogemos la opción Gradle
7. Copiamos el token

What option best describes your build?

1 Name of the environment variable: SONAR_TOKEN

2 Value of the environment variable: 58320ff13433aac5e11af7babf05db8b1382e451

Update your `build.gradle` file with the `org.sonarqube` plugin and it's configuration:

Copy

 Copy

Edit service connection

Authentication

SonarCloud Token

XXXXXXXXXX

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

Verify

Details

Service connection name

sonartestch1

Description (optional)

Security

☒ Grant access permission to all pipelines

Learn more

Troubleshoot

Cancel

Verify and save

Teniendo todos los datos diligenciados damos clic en **“Verify and Save”**

Continuamos diligenciando los datos de nuestra tarea, seleccionando el SonarQube Service Endpoint que acabamos de generar y los demás campos.

Project Key and Name: `$(Build.Repository.Name)`

Prepare Analysis Configuration ⓘ

[Link settings](#)

[View YAML](#)

[Remove](#)

Task version

Display name *

Prepare analysis on SonarQube

SonarQube Server Endpoint * ⓘ | [Manage](#)

sonartestch



[+ New](#)

Choose the way to run the analysis * ⓘ



Integrate with MSBuild



Integrate with Maven or Gradle



Use standalone scanner

Mode * ⓘ



Store configuration with my source code (sonar-project.properties)



Manually provide configuration

Project Key * ⓘ

`$(Build.Repository.Name)`

Project Name ⓘ

`$(Build.Repository.Name)`

Project Version ⓘ

1.0

Damos clic en la sección Advanced y en el campo “Additional Properties” ingresamos lo siguiente.

Advanced ^

Additional Properties ⓘ

```
sonar.sources=$(Build.SourcesDirectory)/src/main/java
sonar.tests=$(Build.SourcesDirectory)/src/test/java
sonar.java.binaries=$(Build.SourcesDirectory)/build/classes
sonar.exclusions=**/*BeforeSuite.java,**/*DataToFeature.java,**/*LectorExcel.java,**/*RunnerPersonalizado.java,**/*.gradle
```

```
sonar.sources=$(Build.SourcesDirectory)/src/main/java
```

```
sonar.tests=$(Build.SourcesDirectory)/src/test/java
```

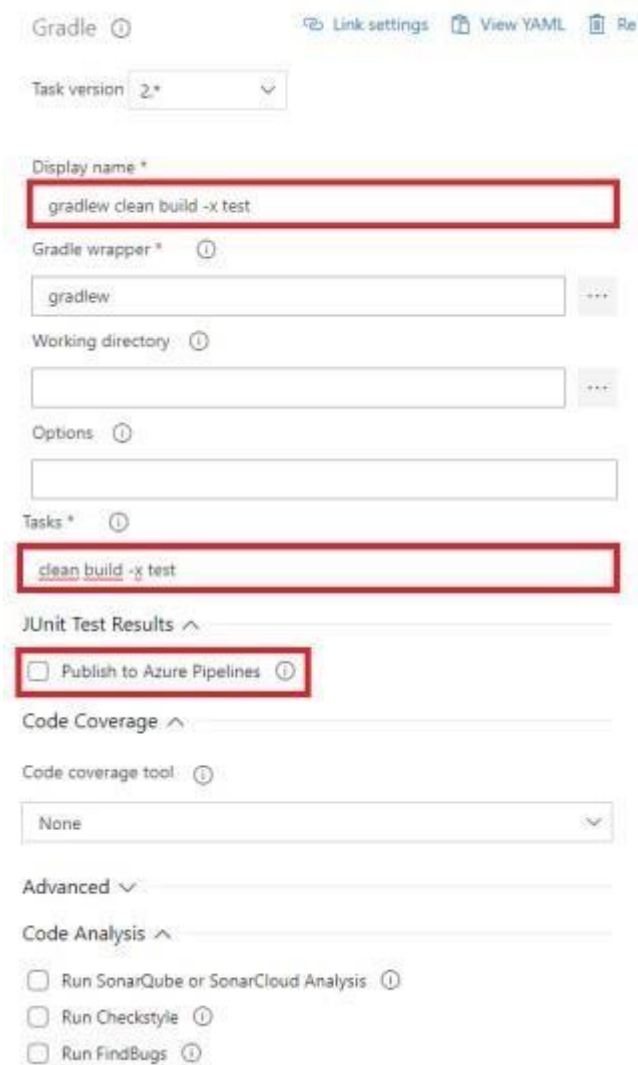
```
sonar.java.binaries=$(Build.SourcesDirectory)/build/classes
```

```
sonar.exclusions=**/*BeforeSuite.java,**/*DataToFeature.java,**/*LectorExcel.java,**/*RunnerPersonalizado.java,**/*.gradle
```

Continuamos seleccionando nuestra tarea de gradle y diligenciamos los campos de la siguiente manera:

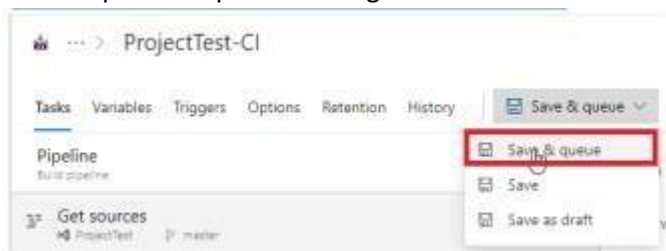
Display name: gradlew clean build -x test

Task: clean build -x test



The screenshot shows the configuration page for a Gradle task in Azure Pipelines. The 'Task version' is set to '2*'. The 'Display name' field is highlighted with a red box and contains the text 'gradlew clean build -x test'. The 'Gradle wrapper' field contains 'gradlew'. The 'Working directory' field is empty. The 'Options' field is empty. The 'Tasks' field is highlighted with a red box and contains the text 'clean build -x test'. The 'JUnit Test Results' section has a checkbox for 'Publish to Azure Pipelines' which is highlighted with a red box. The 'Code Coverage' section has a dropdown menu set to 'None'. The 'Advanced' section is expanded, showing 'Code Analysis' options: 'Run SonarQube or SonarCloud Analysis', 'Run Checkstyle', and 'Run FindBugs', all of which are unchecked.

Terminado de configurar nuestro Pipeline de CI, procedemos a guardar y correr el mismo para verificar que no se presente ningún error.



Agregamos comentarios sobre la modificación realizada, grabamos y ejecutamos.

Run pipeline

Select parameters below and manually run the pipeline

Save comment

Modificación CI

Agent pool

Azure Pipelines

Agent Specification *

windows-2019

Branch/tag

master

Select the branch, commit or commit tag

Variables

+

Demands

+

☐ Enable system diagnostics

Cancel

Save and run

Podemos ir observando la ejecución, dando clic en el job.

#10 version 1.0

on ProjectTest-CI

Cancel

Summary

Manually run by Nelson Ariza

ProjectTest: master: 664b1d

20 Just now

Outputs

Tests

Changes

1 commit

Work items

Artifacts

Jobs

Name	Status	Duration
Agent job 1	Queued	

Jobs in run #10

ProjectTest-CI

Agent job 1

24s

Agent job 1

Job information

- 1 Pool: Azure Pipelines
- 2 Image: windows-2019
- 3 Agent: Hosted Agent
- 4 Started: Just now
- 5
- 6 The agent request is already running or has already completed.
- 7 Job preparation parameters
- 8 1 queue time variable used

- Initialize job 3s
- Checkout 5s
- Prepare analysis on SonarQube 1s
- gradlew clean build -x test 11s
- Run Code Analysis
- Post-job: Checkout

Configuración Pipeline CD

Para nuestra tarea de **“Break build on quality gate failure”** debemos seleccionar nuevamente el SonarCloud Service Endpoint generado anteriormente.

SonarQube build breaker ⓘ [Link settings](#) [View YAML](#) [Remove](#)

Task version 8.* ▼

Display name *
Break build on quality gate failure

SonarQube Service Endpoint * ⓘ [Manage](#)
sonartestch ▼ [Refresh](#) [+ New](#)

Control Options ▼

Output Variables ▼

En nuestra tarea **“Copy files”** configuramos sus parámetros de la siguiente manera:

Display name: Copy files to: \$(build.artifactstagingdirectory)

Source folder: \$(system.defaultworkingdirectory)

Contents: **/*

!build

Target folder: \$(build.artifactstagingdirectory)

Copy files ⓘ [Link settings](#) [View YAML](#) [Remove](#)

Task version 2.* ▼

Display name *
Copy Files to: \$(build.artifactstagingdirectory)

Source Folder ⓘ
\$(system.defaultworkingdirectory) ...

Contents * ⓘ
**/*
!build

Target Folder * ⓘ
\$(build.artifactstagingdirectory)

Finalmente nos dirigimos a nuestra tarea “**Publish Artifact**” y para esta configuramos los valores:

Display name: Publish Artifact

Path to publish: \$(Build.ArtifactStagingDirectory)

Artifact name: RetoAutomatizadorArtifact

Publish build artifacts ⓘ

[Link settings](#)

[View YAML](#)

[Remove](#)

Task version 1.* ▼

Display name *

Publish Artifact

Path to publish * ⓘ

\$(Build.ArtifactStagingDirectory)

Artifact name * ⓘ

RetoAutomatizadorArtifact

Artifact publish location * ⓘ

Azure Pipelines ▼

Advanced ▼

Control Options ▼

Output Variables ▼

Una vez finalizado, procedemos a guardar y ejecutar nuestro pipeline.


Tasks

Variables

Triggers

Options

History

 Save & queue ✓

Con esto terminaríamos la creación de nuestro pipeline CI/CD y para hacer seguimiento a su ejecución nos dirigimos nuevamente a pipelines y seleccionamos el creado anteriormente.

Recently run pipelines

Pipeline



RetoAutomatizador

Nos dirigimos al apartado de jobs y seleccionamos nuestro Agent job 1

Jobs

Name

Status

Duration

✓ Agent job 1

Success

🕒 5m 43s

A continuación, podremos ver el log de la ejecución de cada una de las tareas

←

Jobs in run #29

RetoAutomatizador

Jobs

✓	Agent job 1	5m 43s
✓	Initialize job	6s
✓	Checkout RetoAutomat...	4s
✓	Prepare analysis on Son...	2s
✓	gradlew clean buil...	3m 22s
✓	Run Code Analysis	1m 13s
✓	Break build on quality ...	17s
✓	Copy Files to: D:\a\1\...	<1s
✓	Publish Artifact	34s
✓	Post-job: Checkout Re...	<1s
✓	Finalize Job	<1s
✓	Report build status	<1s

✓ Agent job 1

1

Pool: [Azure Pipelines](#)

2

Image: windows-2019

3

Agent: Hosted Agent

4

Started: Today at 14:25

5

Duration: 5m 43s

6

7

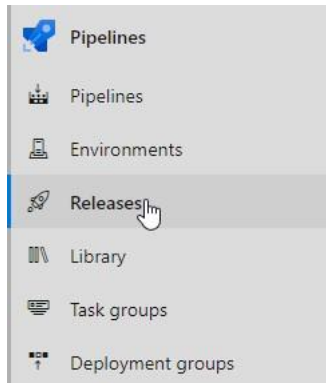
▶ Job preparation parameters

8

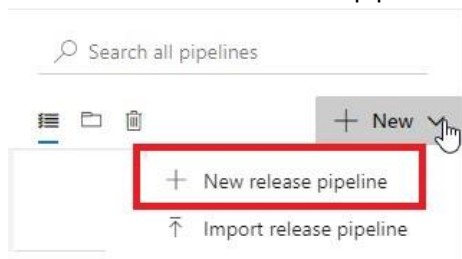
📦 1 artifact produced

Creación Pipeline RM

1. Damos clic en la opción de menú: Releases



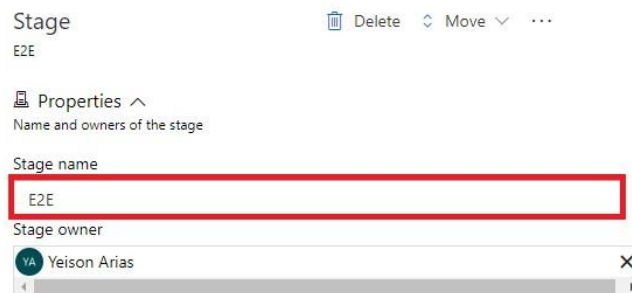
2. Creamos un nuevo "Release pipeline"



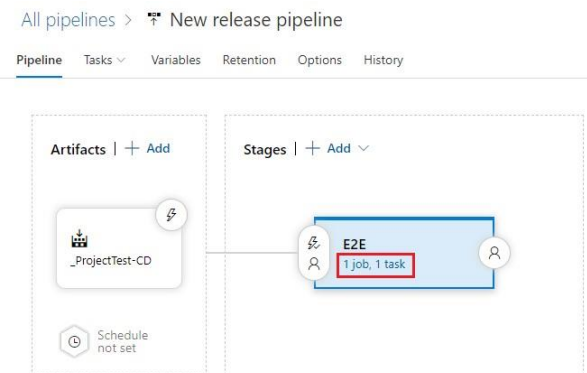
3. Seleccionamos "Empty Job"



4. Le damos nombre a nuestro Stage "E2E"



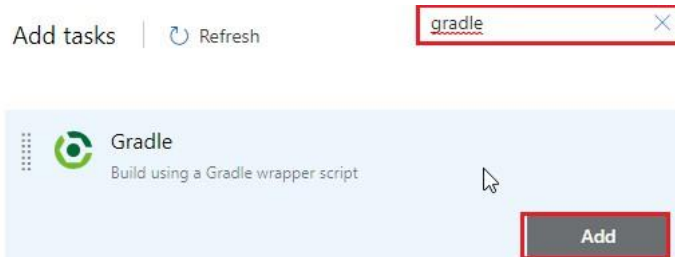
5. Damos clic en Job, task



6. Agregamos una nueva tarea "Task" a nuestro "Agent job"



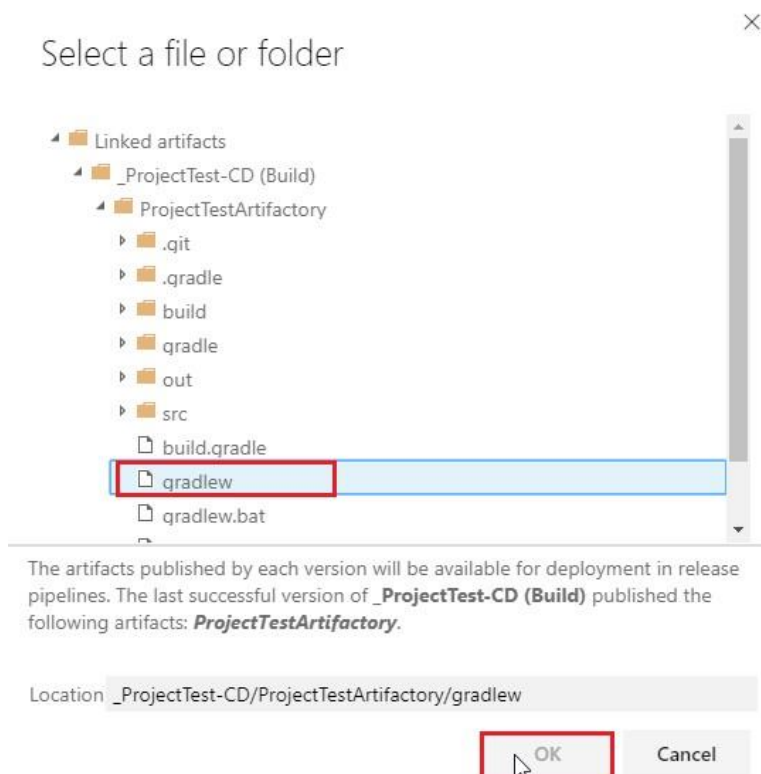
Llamada "gradle"



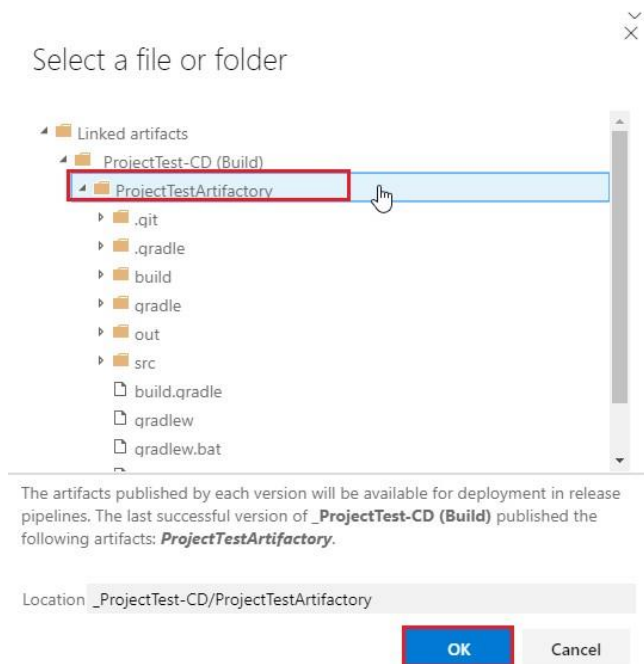
7. Parametrizamos la tarea de la siguiente manera:

Display name: **gradlew**

Para el campo "Gradle wrapper" damos clic en el botón para seleccionar el archivo "gradlew" y clic en el botón "OK"

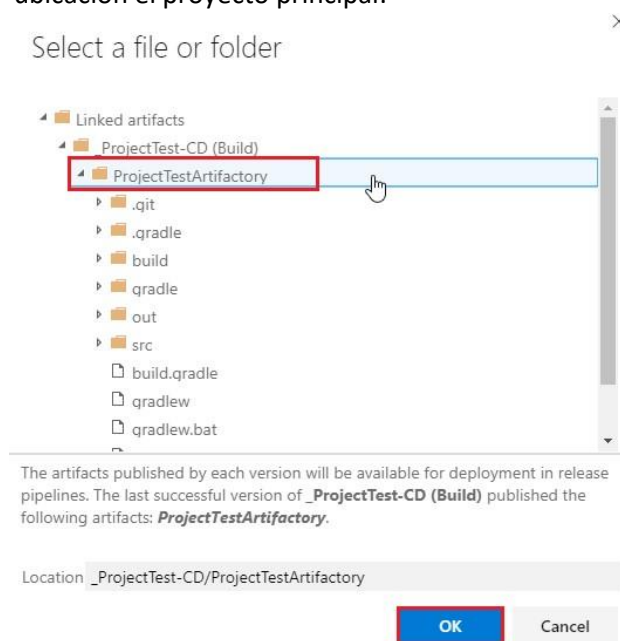


8. Para el campo “Working directory”, de igual manera damos clic en el botón para especificar ruta y seleccionamos el directorio raíz de nuestro repositorio de código y clic en el botón “OK”.



9. En los campos:
Task: **clean test --info aggregate**

10. Para el campo “Test results files” damos clic para seleccionar inicialmente como ubicación el proyecto principal:



Y a la ruta que queda agregada:

```
$(System.DefaultWorkingDirectory)/_ProjectTest-  
CD/ProjectTestArtifactory
```

Le complementamos con el siguiente segmento:

/target/site/serenity/SERENITY*.xml

Quedando como ruta definitiva: **\$(System.DefaultWorkingDirectory)/_ProjectTest-
CD/ProjectTestArtifactory/target/site/serenity/SERENITY-*.xml**

Test results files *



```
$(System.DefaultWorkingDirectory)/_ProjectTest-  
CD/ProjectTestArtifactory/target/site/serenity/SERENITY-*.xml
```



11. En el campo “Test run title” colocamos: “pruebas de aceptación” y finalizamos dando clic en SAVE



Create release ...

```
$(System.DefaultWorkingDirectory)/_ProjectTest-  
CD/ProjectTestArtifactory/gradlew
```



Working directory ⓘ

```
$(System.DefaultWorkingDirectory)/_ProjectTest-  
CD/ProjectTestArtifact
```



Options

Tasks *

```
clean test --info aggregate
```

JUnit Test Results ^

☒ Publish to Azure Pipelines ⓘ

Test results files * ⓘ

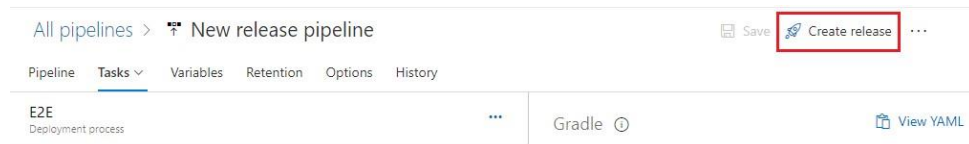
```
$(System.DefaultWorkingDirectory)/_ProjectTest-  
CD/ProjectTestArtifactory/target/site/serenity/SERENITY-*.xml
```



Test run title ⓘ

```
pruebas de aceptacion
```

12. Para ejecutar el pipeline de RM. Damos clic en el botón “Create Release”



Damos clic en el Stage “E2E” y luego clic en el botón “Crear”

Create a new release

New release pipeline

Pipeline ^

Click on a stage to change its trigger from automated to manual.



Stages for a trigger change from automated to manual. ⓘ

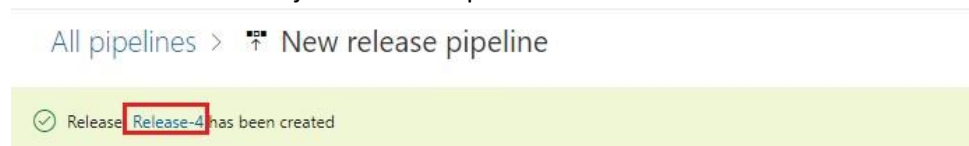
Artifacts ^

Select the version for the artifact sources for this release

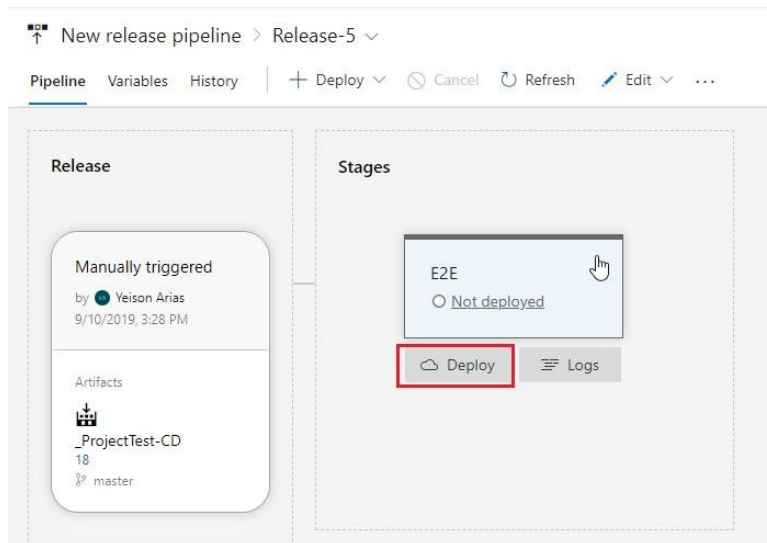
Source alias	Version
_ProjectTest-CD	18

Release description

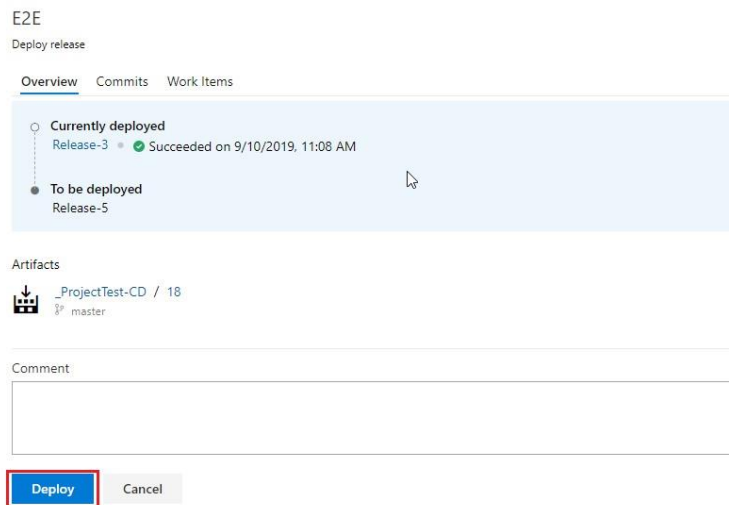
Para continuar con la ejecución del Pipeline damos clic en el link



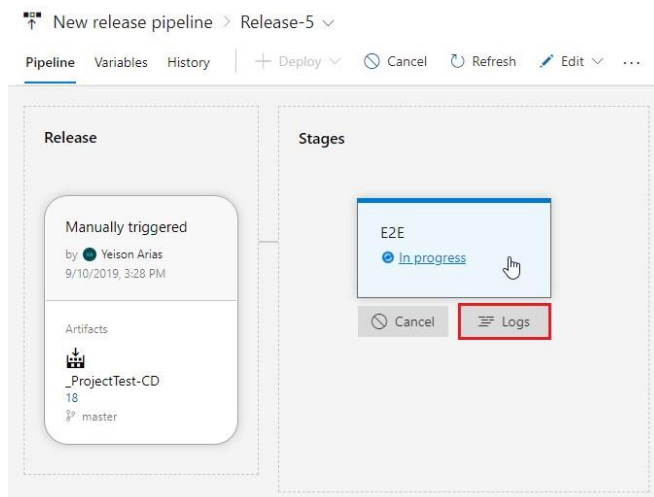
Luego colocamos el cursor sobre el Stage “E2E” y damos clic en “Deploy”



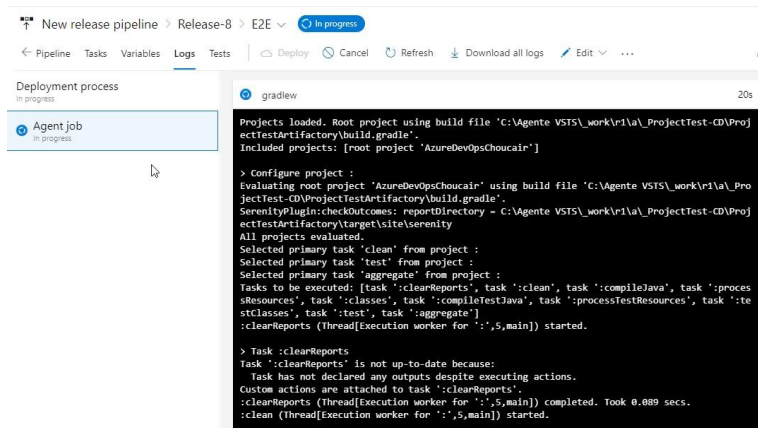
Nuevamente clic en el botón “Deploy”



Para hacer seguimiento damos clic en “Logs”



Y visualizamos el Log de la ejecución.



Aquí hemos terminado con la creación de los pipelines (CI, CD, RM), practícalo tantas veces sea necesario.