La partie CI

Plan de contenu

[**A. Création et déploiement manuel du projet** 2](#_heading=h.gjdgxs)

[A.1. Configuration du projet 2](#_heading=h.30j0zll)

[A.2. Construction du projet et d’une page web simple ASP.NET 3](#_heading=h.1fob9te)

[A.3. Construction du projet et d’une page web simple ASP.NET 4](#_heading=h.3znysh7)

[A.4. Gestion de version avec GIT 5](#_heading=h.2et92p0)

[A.5. Création d’un dépôt sur AZURE DEVOPS 6](#_heading=h.tyjcwt)

[**B. Intégration continue sur AZURE DEVOPS** 7](#_heading=h.4d34og8)

[B.1. Configuration du pipeline (classic editor pipeline) 7](#_heading=h.2s8eyo1)

[B.2. Création de la pipeline CI en utilisant l’outils classic editor pipeline (méthode 1) 7](#_heading=h.17dp8vu)

[B.3. Création de la pipeline CI (Méthode # 2 - YAML) 9](#_heading=h.3rdcrjn)

[B.4. Création de la pipeline CI (Méthode # 3 – via le classic editor pipeline) 10](#_heading=h.26in1rg)

[SECTION FACULTATIVE : les variables 10](#_heading=h.lnxbz9)

**A. Création et déploiement manuel du projet**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PARTIE # A.1 – Configuration du projet \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

A.1. Configuration du projet

Téléchargez le Framework dotnet .NET5

cd C:\Users\Haythem\Desktop

dotnet --version

dotnet --list-sdks

dotnet --list-runtimes

dotnet --help

md DevopsDemo

cd DevopsDemo

dotnet new globaljson

ls

cat .\global.json

dotnet --list-sdks

nano .\global.json

***Dépannage & troubleshooting :***

<https://dev.to/polarbit/how-to-change-default-dotnet-sdk-version-43ph>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PARTIE # A.2 – Construction du projet et de la page web \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

A.2. Construction du projet et d’une page web simple ASP.NET

cd  **C:\ ou C:\**

**md DevopsDemo**

cd **DevopsDemo**

dotnet new sln -o HelloWorldApp

cd HelloWorldApp

dir

dotnet new mvc -n HelloWorldApp.web

dir

code .

cd HelloWorldApp.web

dir

dotnet build

dotnet build --configuration debug

dotnet build --configuration release

Aller à build/bin et vérifier la présence de deux dossiers Debug et Release

cd **C:\DevopsDemo**\HelloWorldApp\HelloWorldApp.web\bin\release\net7.0

dotnet HelloWorldApp.web.dll

Vérifier <http://localhost:5000/>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PARTIE # A.3 – Déploiement de la page web manuellement \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

A.3. Construction du projet et d’une page web simple ASP.NET

cd **C:\DevopsDemo\**HelloWorldApp\HelloWorldApp.web

dotnet new sln -o HelloWorldApp

dotnet publish -o /out

dotnet publish -c release -o /out

cd C:\out\

dotnet HelloWorldApp.web.dll

Vérifier <http://localhost:5000/>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PARTIE # A.4 – Gestion de version avec GIT \*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

A.4. Gestion de version avec GIT

cd **C:\DevopsDemo\**HelloWorldApp

git init

git status

git config --global user.name "hrhouma"

git config --global user.email "rehoumahaythem@gmail.com"

git config --local user.name "hrhouma"

git config --local user.email "rehoumahaythem@gmail.com"

code .

git add .

git status

git commit -m "first commit"

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PARTIE # A.5 – Création d’un dépôt sur AZURE DEVOPS \*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<https://status.dev.azure.com/>

<https://aex.dev.azure.com/>

A.5. Création d’un dépôt sur AZURE DEVOPS

1 ) Allez à AZURE DEVOPS

2) Créer un profil ici ==> **https://dev.azure.com/**

3) Créer une organisation dans AZURE DEVOPS

4) Créer un projet HelloWorldApp sur AZURE DEVOPS

5) Cliquez sur Repos

6) Nous allons nous préparer à faire le push du dépôt local (LOCAL) au dépôt distant (AZURE DEVOPS ) sur Azure.

git remote help

cat .git/config

git remote remove origin

7) Génération des identifiants (***Credentials***) sur Azure DEVOPS:

Dans Azure, cherchez l'option « *generate credentials* ».

Attention le *password* sera affiché uniquement une seule fois.

Username: hrehouma

Password: vv2qfumwpirb3lcol6bynlkachvsnoigmic5afbx2r5dbxg7e5tq

git remote add origin https://hrehouma0084@dev.azure.com/hrehouma0084/HelloWorldApp/\_git/HelloWorldApp

git push -u origin --all

8) SECTION FACULTATIVE : confirmation de la sauvegarde des informations sur votre ordinateur local via le gestionnaire d'identifiants

*Remarque : Pour vérifier la présence de ces informations, consultez le Gestionnaire d'identifiants où elles ont été préalablement enregistrées.*

* Adresse Internet ou réseau : git:https://hrehouma0084@dev.azure.com
* Nom d'utilisateur: hrehouma0084
* Mot de passe : vv2qfumwpirb3lcol6bynlkachvsnoigmic5afbx2r5dbxg7e5tq
* Persistance : Cette configuration garantit que vos identifiants sont conservés sur votre machine pour des connexions futures, éliminant le besoin de les saisir à nouveau.

**B. Intégration continue sur AZURE DEVOPS**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PARTIE # B.1 – Configuration de la pipeline \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

B.1. Configuration du pipeline (classic editor pipeline)

**IMPORTANT :** pour travailler avec le « ***classic editor pipeline*** », il faut l'activer au niveau organisationnel.

**Étape B.1:**

Organisation settings => pipelines => settings =>Disable creation of classic build pipelines + Disable creation of classic release pipelines => off.

**Étape B.2:**

Project settings => pipelines => settings =>Disable creation of classic build pipelines + Disable creation of classic release pipelines => off.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\* PARTIE # B.2 – Création de la pipeline CI (Méthode # 1 - classic editor pipeline) \*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

B.2. Création de la pipeline CI en utilisant l’outils classic editor pipeline (méthode 1)

**Étape B.3:**

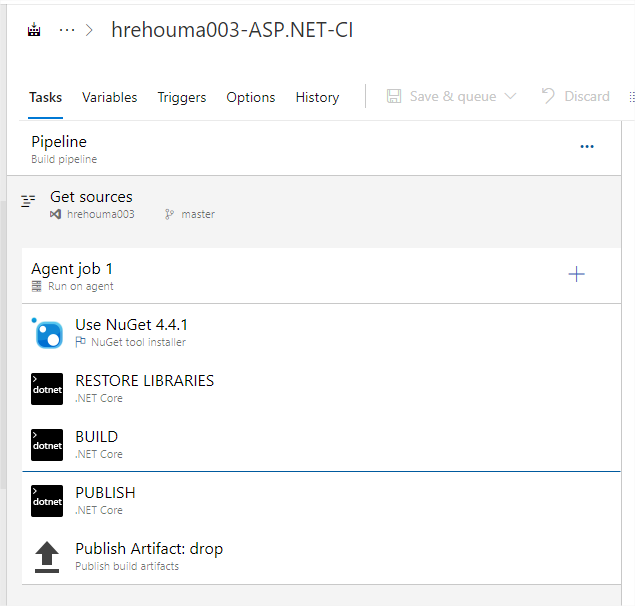
Pipelines => new pipeline => Use the classic editor => Azure Repos Git => ASP.NET Core (SANS LE .NET Framework).

**Étape B.4:**

RESTORE => Build => Test => Publish => Publish Artifact.

TRÈS IMPORTANT :

* *Il faut supprimer les 4 composants proposés après avoir choisi ASP.NET Core.*
* *Vous pouvez laisser Use NuGet 4.4.1 si vous le désirez.*
* *Choisir 3 composant .NET CORE*
* *Choisir un composant à la fin Publish Artifact: drop*

**

*PREMIER COMPOSANT (1/4)*

RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

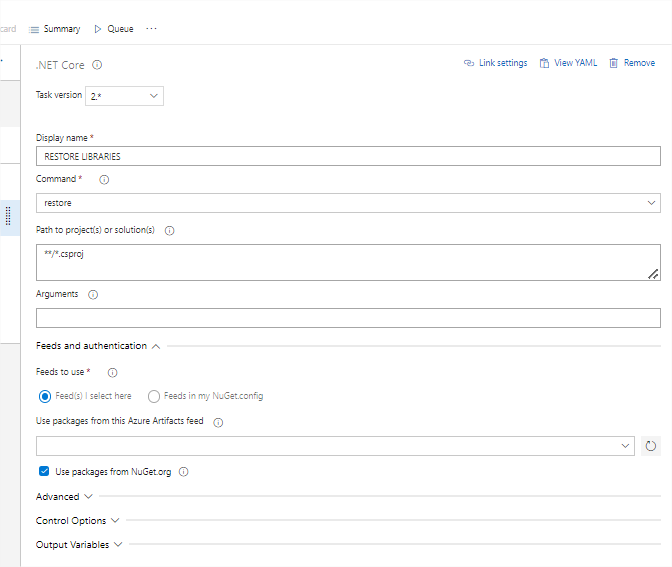
\*\*\* RESTORE

Display name : Restore Libraries

Command : restore

Path to project(s) : \*\*/\*.csproj  *(si ça ne fonctionne pas, il faut le changer d’abord dans Get Sources ou Agent Job)*

Arguments : NO



*DEUXIÈME COMPOSANT (2/4)*

RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

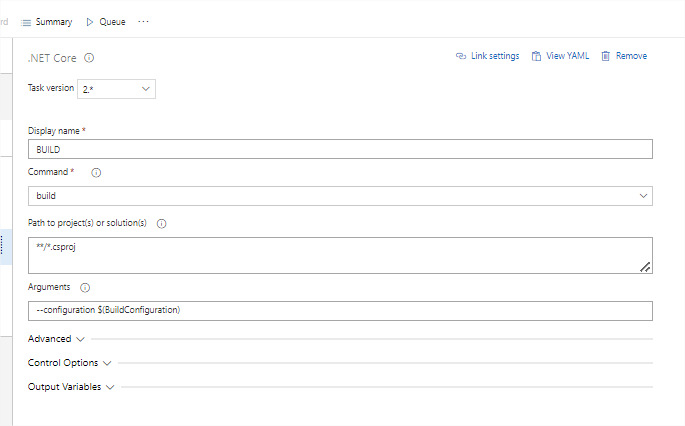
\*\*\* RESTORE => BUILD

Display name : Build

Command : build

Path to project(s) : \*\*/\*.csproj

Arguments : --configuration $(BuildConfiguration)



*TROISIÈME COMPOSANT (3/4)*

RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

\*\*\* RESTORE => BUILD => PUBLISH

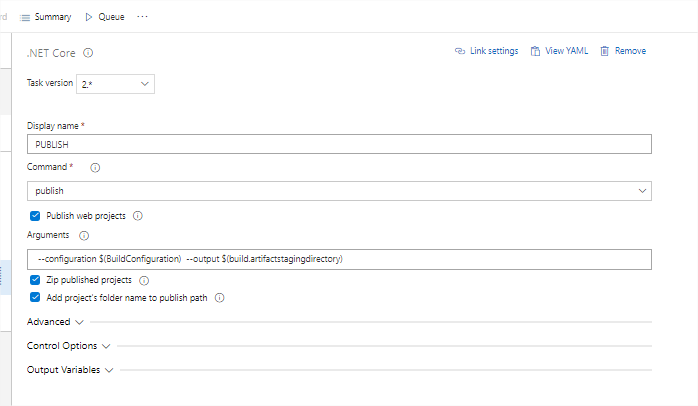
Display name : Publish

Command: publish

Path to project(s) : \*\*/\*.csproj

Arguments : --configuration $(BuildConfiguration) --output $(build.artifactstagingdirectory)

**il faut cocher zip published projects + add project's folder name to publish path**



*QUATRIÈME COMPOSANT (4/4)*

***Attention : ce type de composant est différent !***

RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

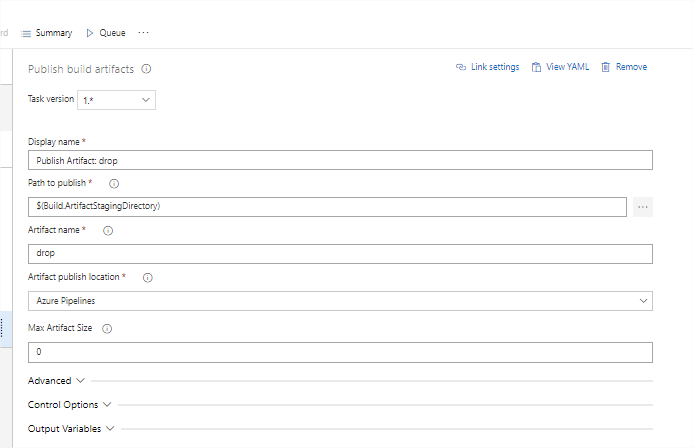
\*\*\* RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

Display name : Publish Artifact: drop

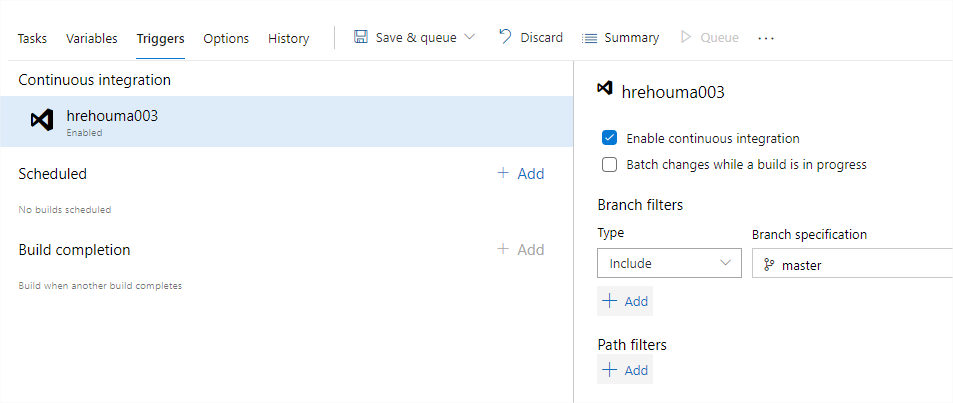
Path to publish: $(Build.ArtifactStagingDirectory)

Artifact name : drop

Artifact publish location : Azure pipelines

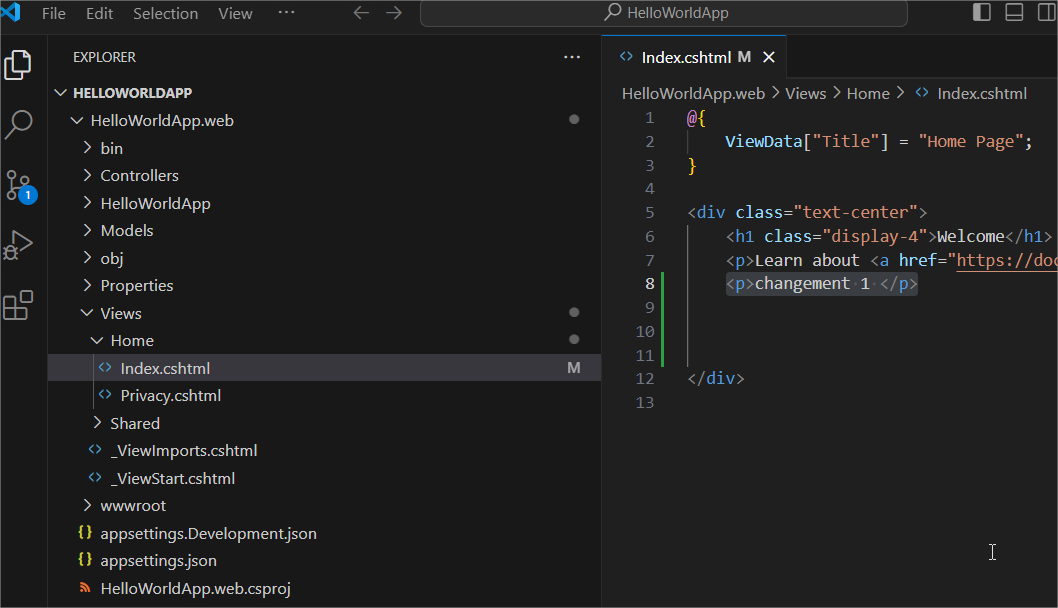


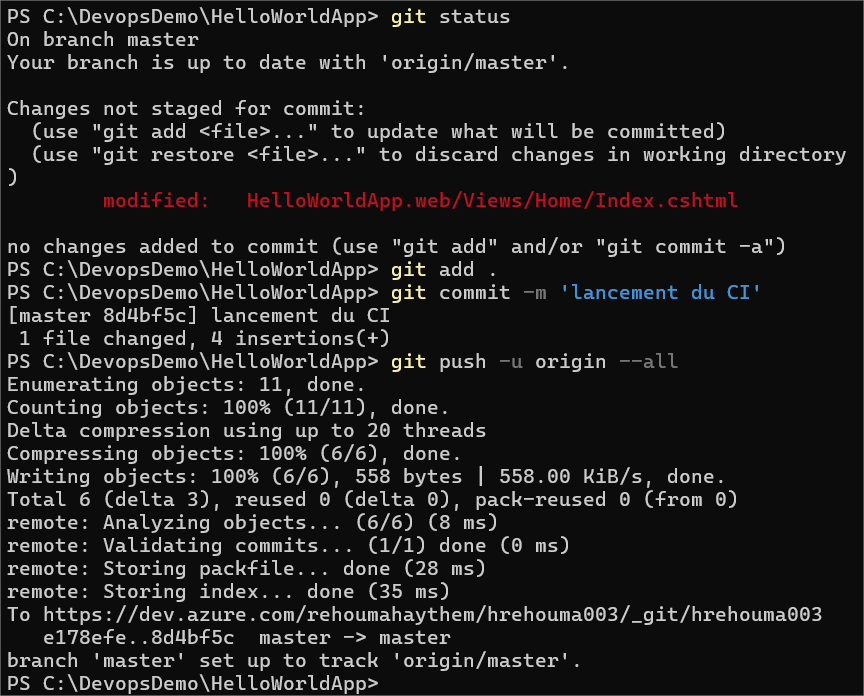
*TRIGGER (AUTOMATISER)*



*Faites ce changement :*

**<p> changement 1 </p>**



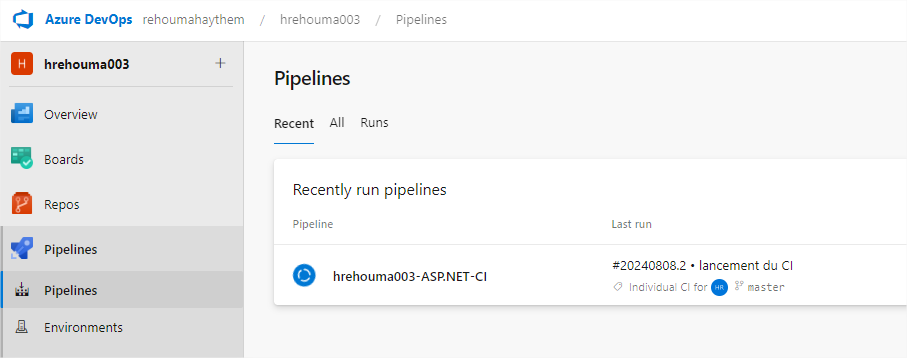


**git status**

**git add .**

**git commit -m 'lancement du CI'**

**git push -u origin --all**



***Voir l’annexe 1 pour les explications.***

*MÉTHODE 2*

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\* PARTIE # B.3 – Création de la pipeline CI (Méthode # 2 - YAML) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

B.3. Création de la pipeline CI (Méthode # 2 - YAML)

trigger:

- main

pool:

vmImage: 'ubuntu-22.04'

steps:

- task: UseDotNet@2

inputs:

packageType: 'sdk'

version: '7.x'

installationPath: $(Agent.ToolsDirectory)/dotnet

- task: DotNetCoreCLI@2

displayName: 'Restore Libraries'

inputs:

command: 'restore'

projects: '\*\*/\*.csproj'

- task: DotNetCoreCLI@2

displayName: 'Build'

inputs:

command: 'build'

projects: '\*\*/\*.csproj'

arguments: '--configuration $(BuildConfiguration)'

- task: DotNetCoreCLI@2

displayName: 'Publish'

inputs:

command: 'publish'

publishWebProjects: true

projects: '\*\*/\*.csproj'

arguments: '--configuration $(BuildConfiguration) --output $(Build.ArtifactStagingDirectory)'

zipAfterPublish: true

modifyOutputPath: true

- task: PublishBuildArtifacts@1

displayName: 'Publish Artifact'

inputs:

PathtoPublish: '$(Build.ArtifactStagingDirectory)'

ArtifactName: 'drop'

publishLocation: 'Container'

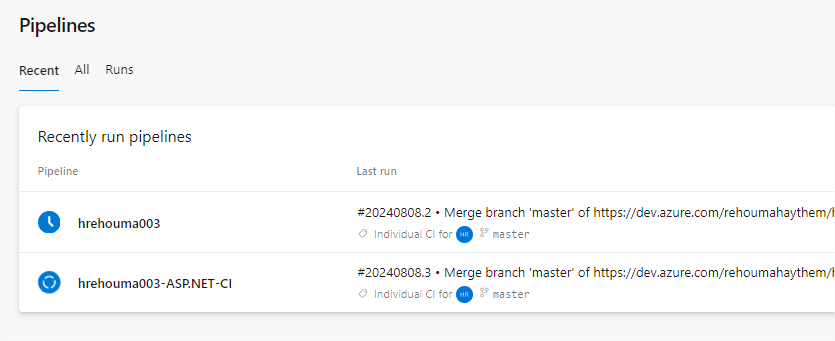
TESTEZ : Effectuer un changement dans votre page web et ensuite faire un

**git status**

**git add .**

**git commit -m 'lancement du CI'**

**git push -u origin --all**



***Voire l’annexe 2 pour les explications.***

ANNEXE

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\* PARTIE # B.4 – Création de la pipeline CI (Méthode # 3 - classic editor pipeline) \*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

B.4. Création de la pipeline CI (Méthode # 3 – via le classic editor pipeline)

Utilisation de composants « ***.NET task*** » à la place de l’utilisation de la pile « ***ASP.NET Core*** »

* .NET task
* ***Ou*** Utilisez .NET Core SDK 7.x
* Version ==> 7.x
* Build solution \*\*/.\*.sln

* RESTORE Arguments ==> -p:Configuration=Release
* Build Arguments ==> --configuration $(BuildConfiguration) --output $(Build.ArtifactStagingDirectory)
* Test Disable
* Publish Artifact Path to publish ==> $(build.artifactstagingdirectory)

SECTION FACULTATIVE : les variables

Exemples de variables testées

--configuration $(BuildConfiguration)

-p:Configuration=Release

--output $(Build.ArtifactStagingDirectory)

--configuration $(BuildConfiguration) --output $(build.artifactstagingdirectory)

-p:Configuration=Release -p:OutputPath=$(Build.ArtifactStagingDirectory)

Autres essais :

-p:OutputPath=$(build.artifactstagingdirectory)

# ANNEXE 1 - Aperçu détaillé des étapes et des arguments dans un pipeline CI/CD pour un projet .NET :

Chaque étape de ce pipeline représente un processus typique dans le développement .NET/ C#/ WEB /DATA ETC … :

1) Restaurer les dépendances,

2) Compiler le projet,

3) Préparer le projet pour le déploiement,

4) et Stocker les artefacts pour le déploiement, avec des arguments spécifiques pour contrôler les comportements de chaque commande.

**1. RESTORE**

RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

\*\*\* RESTORE

Display name : Restore Libraries

Command : restore

Path to project(s) : \*\*/\*.csproj

Arguments : NO

- Nom : "Restore Libraries"

- Commande : restore

- Chemin : \*\*/\*.csproj

- Arguments : Aucun (NO)

- Cette étape restaure les paquets NuGet nécessaires pour le projet sans options supplémentaires.

**2. RESTORE => BUILD**

RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

\*\*\* RESTORE => BUILD

Display name : Build

Command : build

Path to project(s) : \*\*/\*.csproj

Arguments : --configuration $(BuildConfiguration)

- Nom "Build"

- Commande : build

- Chemin : \* \*\*/\*.csproj

- Arguments --configuration $(BuildConfiguration)

- --configuration spécifie la configuration de build (comme Debug ou Release).

- $(BuildConfiguration) est une variable contenant la valeur spécifique de la configuration de build à utiliser.

**3. RESTORE => BUILD => PUBLISH**

RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

\*\*\* RESTORE => BUILD => PUBLISH

Display name : Publish

Command: publish

Path to project(s) : \*\*/\*.csproj

Arguments : --configuration $(BuildConfiguration) --output $(build.artifactstagingdirectory)

il faut cocher zip published projects + add project's folder name to publish path

- Nom "Publish"

- Commande publish

- Chemin\* \*\*/\*.csproj

- Arguments --configuration $(BuildConfiguration) --output $(build.artifactstagingdirectory)

- --configuration $(BuildConfiguration) définit la configuration de build.

- --output $(build.artifactstagingdirectory) détermine le répertoire de sortie pour les artefacts publiés.

- Options supplémentaires

- "zip published projects" pour compresser les projets publiés en format zip.

- "add project's folder name to publish path" pour inclure le nom du dossier du projet dans le chemin de publication.

**4. RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT**

RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

\*\*\* RESTORE => BUILD => PUBLISH => PUBLISH ARTIFACT

Display name : Restore Libraries

Command: restore

Path to project(s) : \*\*/\*.csproj

Arguments : NO

- Nom "Restore Libraries"

- Commande restore

- Chemin\* \*\*/\*.csproj

- Arguments Aucun (NO)

- Cette étape est une répétition de la première, sans arguments supplémentaires.

**5. RÉSUMÉ**

Pour résumer, le processus de build dans le contexte de développement .NET/C#/Web/Data peut paraître complexe, mais en le décomposant étape par étape, il devient plus facile à comprendre. Voici une explication détaillée de chaque étape mentionnée :

**1. Restaurer les dépendances (RESTORE)**

-Objectif : Avant de pouvoir compiler un projet, il est essentiel de s'assurer que toutes les bibliothèques et dépendances externes requises par le projet sont disponibles. Dans le monde .NET, ces dépendances sont souvent gérées par NuGet, un gestionnaire de paquets pour .NET.

-Comment ça fonctionne

-Commande : restore

-Chemin : \*\*/\*.csproj (ceci indique que la commande doit chercher tous les fichiers .csproj dans le répertoire courant et ses sous-répertoires).

-Arguments : Aucun. Cela signifie que la commande restore s'exécutera avec ses paramètres par défaut, se concentrant simplement sur la récupération et l'installation des paquets NuGet sans modifications spécifiques.

**2. Compiler le projet (BUILD)**

-Objectif : La compilation est le processus de transformation du code source écrit dans un langage de programmation (comme C#) en un format exécutable ou en une bibliothèque. C'est une étape cruciale pour vérifier que le code est syntaxiquement correct et ne contient pas d'erreurs de compilation.

-Comment ça fonctionne

-Commande : build

-Chemin : \*\*/\*.csproj

-Arguments : --configuration $(BuildConfiguration) permet de spécifier si le projet doit être compilé en mode Debug (pour le développement) ou Release (pour la production). $(BuildConfiguration) est une variable qui sera remplacée par la valeur appropriée au moment de l'exécution.

**3. Préparer le projet pour le déploiement (PUBLISH)**

-Objectif : Après la compilation, le projet doit être préparé pour le déploiement. Cela implique de rassembler tous les fichiers nécessaires pour exécuter l'application, y compris les fichiers compilés, les ressources, les configurations, etc., et de les placer dans un répertoire spécifique.

-Comment ça fonctionne

-Commande : publish

-Chemin : \*\*/\*.csproj

-Arguments : --configuration $(BuildConfiguration) --output $(build.artifactstagingdirectory) spécifie non seulement la configuration de build, mais aussi le répertoire de sortie où les fichiers publiés doivent être placés. Les options supplémentaires comme "zip published projects" et "add project's folder name to publish path" permettent de compresser le résultat en un fichier zip et d'organiser les artefacts de manière plus structurée.

**4. Stocker les artefacts pour le déploiement (PUBLISH ARTIFACT)**

-Objectif : Cette étape consiste à stocker les artefacts (les fichiers résultant de l'étape de publication) dans un emplacement spécifique pour le déploiement. Ces artefacts peuvent ensuite être déployés sur un serveur ou dans un environnement cloud.

-Comment ça fonctionne : Cette étape devrait concerner la collecte et le stockage des artefacts générés lors de l'étape de publication. Cela inclut généralement la commande pour publier ou copier les artefacts dans un système de stockage d'artefacts ou un répertoire de déploiement.

En résumé, ce pipeline de build dans un environnement .NET/C# consiste à restaurer les dépendances nécessaires, compiler le code source en un exécutable ou une bibliothèque, préparer ces fichiers pour le déploiement en les organisant et les compressant au besoin, et enfin stocker les artefacts générés dans un emplacement approprié pour le déploiement ultérieur.

# ANNEXE 2 - Aperçu détaillé des étapes et des arguments dans un pipeline CI/CD – Version YAML:

trigger:

- main

pool:

vmImage: 'ubuntu-22.04'

steps:

- task: UseDotNet@2

inputs:

packageType: 'sdk'

version: '7.x'

installationPath: $(Agent.ToolsDirectory)/dotnet

- task: DotNetCoreCLI@2

displayName: 'Restore Libraries'

inputs:

command: 'restore'

projects: '\*\*/\*.csproj'

- task: DotNetCoreCLI@2

displayName: 'Build'

inputs:

command: 'build'

projects: '\*\*/\*.csproj'

arguments: '--configuration $(BuildConfiguration)'

- task: DotNetCoreCLI@2

displayName: 'Publish'

inputs:

command: 'publish'

publishWebProjects: true

projects: '\*\*/\*.csproj'

arguments: '--configuration $(BuildConfiguration) --output $(Build.ArtifactStagingDirectory)'

zipAfterPublish: true

modifyOutputPath: true

- task: PublishBuildArtifacts@1

displayName: 'Publish Artifact'

inputs:

PathtoPublish: '$(Build.ArtifactStagingDirectory)'

ArtifactName: 'drop'

publishLocation: 'Container'

Ce pipeline de build YAML pour Azure DevOps automatise le processus de compilation et de déploiement d'une application .NET Core sur une machine virtuelle Ubuntu 22.04. Voici un résumé concis de chaque étape :

**1.Trigger:** La pipeline se déclenche automatiquement à chaque push sur la branche main.

**2.Environnement:** Exécute les tâches sur une machine virtuelle utilisant l'image ubuntu-22.04.

**3.Étape 1 - Installation du SDK .NET:** Utilise la version 7.x du SDK .NET Core pour compiler l'application, installée dans le répertoire spécifié.

**4.Étape 2 - Restauration des dépendances:** Exécute la commande restore pour récupérer les dépendances nécessaires du projet à partir des fichiers .csproj.

**5.Étape 3 - Compilation:** Compile les projets avec la commande build, en utilisant une configuration spécifiée (Debug ou Release).

**6.Étape 4 - Publication:** Publie les projets web, plaçant les artefacts dans un répertoire spécifié et les compresse en fichier zip après la publication.

**7.Étape 5 - Publication des artefacts**: Publie les artefacts compilés et empaquetés (nommés drop) dans un conteneur pour un déploiement ultérieur.

Chaque tâche est finement contrôlée par des paramètres spécifiques, optimisant le processus de build pour le développement .NET Core sur Azure DevOps.