

Laboratório de Desenvolvimento de Software

Célio Carvalho
cdf@estg.ipp.pt

P. PORTO

CONTEXTUALIZAÇÃO

- **CI/CD** = *Continuous Integration & Continuous Delivery* (ou *Continuous Deployment*)
- Um *pipeline* CI/CD permite que os processos de *Integration* (integração) e *Delivery* (entrega) sejam automatizados até ao nível pretendido pela equipa de desenvolvimento. Por exemplo, é possível definir um *pipeline* para que, após o *merge* para um determinado *branch* específico (e.g. *production*), a aplicação seja compilada, testada e publicada, entrando em produção de seguida. O nível de automatismo é definido pela própria equipa.
- **Continuous Integration** ou **Integração Contínua**, refere-se à capacidade do sistema em integrar continuamente o código que é submetido num repositório central (e.g. servidor GitLab). Após integração são, normalmente, iniciadas tarefas de testes automáticos e análise de qualidade de código. O objetivo é detetar problemas o mais cedo possível (e.g. durante o *merge* de uma nova *feature*).
- **Continuous Delivery** ou **Entrega Contínua**, ocorre após a integração contínua, e refere-se à capacidade do sistema em entregar / publicar peças de software entretanto concluídas, integradas, e testadas. Normalmente a entrega é feita num sistema de pré-produção para que se realizem os testes de aceitação de utilizador (*User Acceptance Tests*).

VANTAGENS CI/CD

- Acelera o processo de desenvolvimento e entrega de software.
- Melhora a qualidade do código final, através da execução dos testes automáticos a cada integração de código.
- Redução de tarefas manuais de compilação, testes, e publicação, diminuindo também os erros associados a estas tarefas.
- Como é obtido feedback imediato aquando da realização dos testes, os programador pode corrigir os problemas imediatamente. Como tem presente o trabalho realizado, poderá corrigir os problemas de forma mais célere e eficiente.
- A entrega contínua, permite que possam ser disponibilizadas atualizações a qualquer momento, sem depender da DevOps ou de um calendário de publicações tão rígido.
- Resposta mais rápida aos pedidos dos clientes.
- Permite disponibilizar a correção de *bugs* mais rapidamente aos utilizadores.

Resumindo, o CI/CD permite que as equipas de desenvolvimento possam entregar software com mais qualidade, mais rapidamente, e com menor risco de bugs por causa dos testes automáticos. O CI/CD é essencial no desenvolvimento ágil e na competitividade das empresas de desenvolvimento de software.

GITLAB CI/CD | CONCEITOS BÁSICOS

- O GitLab oferece CI/CD como uma extensão às suas capacidades de Git. Existem outras plataformas de CI/CD (e.g. Jenkins), mas o GitLab oferece esta funcionalidade como uma extensão da sua função de repositório de código. Após a receção de código, os pipeline conseguem executar as tarefas CI/CD como um complemento.

Termo	Descrição
PIPELINE	Processo onde são definidas as tarefas e a sequência destas que devem ser executadas no CI/CD. O <i>pipeline</i> é constituído por <i>jobs</i> e <i>stages</i> .
JOBS	Definem os passos / tarefas a serem executados (e.g. compilar, testar). A ordem é definida pelos <i>stages</i> .
STAGES	Definem a ordem pela qual os <i>jobs</i> devem ser executados. O trabalho a executar é definido dentro dos <i>jobs</i> .
RUNNERS	São aplicações que efetuam realmente os trabalhos definidos pelas instruções inseridas dentro dos <i>jobs</i> . Os <i>runners</i> podem ser instalados na cloud, na máquina local, ou em sistemas internos das equipas de desenvolvimento (servidores locais). O GitLab disponibiliza <i>shared runners</i> que podem ser referenciados para executarem testes.
VARIABLES	Dentro de um <i>pipeline</i> é possível definir variáveis. Essas variáveis são acessíveis em todos os <i>jobs</i> .
ARTIFACTS	Se um <i>job</i> produz <i>output</i> para ficheiros, deve ser indicado o caminho onde devem ser guardados.
TAGS	Utilizam-se <i>tags</i> (<i>labels</i>) para identificar o <i>runner</i> onde um <i>job</i> deve ser executado. Se não forem indicadas <i>tags</i> , os <i>jobs</i> serão executados em <i>shared runners</i> .

GITLAB CI/CD | CONTEXTUALIZAÇÃO



- O GitLab permite efetuar testes do código do repositório de forma distribuída. Pode-se executar *jobs* de forma paralela, e pode executar-se testes em vários ambientes e em paralelo.
- Os *jobs* podem correr tarefas de *build* em paralelo em várias máquinas, tornando o processo de integração e testes muito mais rápido.
- No GitLab as funcionalidades são *opensource* e incluídas em todas as versões do Gitlab.
 - Multi-plataforma (Windows, Linux, macOS).
 - Multi-linguagem (e.g. dotnet, nodeJS, python, php).
 - Estável.
 - *Builds* paralelas.
 - *Logging* em tempo real.
 - *Pipelines*.
 - *Autoscalling*.
 - Permite a construção de artefactos.
 - Suporta Docker.
 - Permite trabalhar com *runners* locais.

GITLAB CI/CD | RUNNERS



- Os *jobs* do GitLab são efetivamente executados em *runners*. O GitLab disponibiliza *shared runners*, mas também podem ser configurados *runners* fora do GitLab (e.g. ambiente de desenvolvimento dos programadores, ou servidores das equipas de desenvolvimento).
- Existem vários tipos de *runners*: Shell, SSH, VirtualBox, Parallels, Docker, Kubernetes, etc.
- Os *runners* são ambientes isolados, que podem ser usados para várias tarefas:
 - ✓ Testar a compilação de projetos;
 - ✓ Executar testes aos projetos; e
 - ✓ Executar tarefas para *deployment* de projetos.
- Os *runners* em GitLab permitem flexibilidade nas tarefas a executar e na sua configuração. Um projeto criado no <https://gitlab.com> contém automaticamente `gitlab runners` que podem ser usados.

GITLAB CI/CD | RUNNERS | SETTINGS

- Para aceder às configurações dos runners no GitLab, aceda à opção indicada abaixo.
➤ Projects | «seleccione projeto» | Settings | CI/CD | Runners
- Nesta página pode-se, por exemplo, definir novos runners e podem ser desativados os shared runners do GitLab.

Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. [What is GitLab Runner?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine.

How do runners pick up jobs?

Runners are either:

- active - Available to run jobs.
- paused - Not available to run jobs.

Tags control which type of jobs a runner can handle. By tagging a runner, you make sure shared runners only handle the jobs they are equipped to run. [Learn more.](#)

Project runners

These runners are assigned to this project.

[New project runner](#)



Shared runners

These runners are available to all groups and projects.

Each CI/CD job runs on a separate, isolated virtual machine.

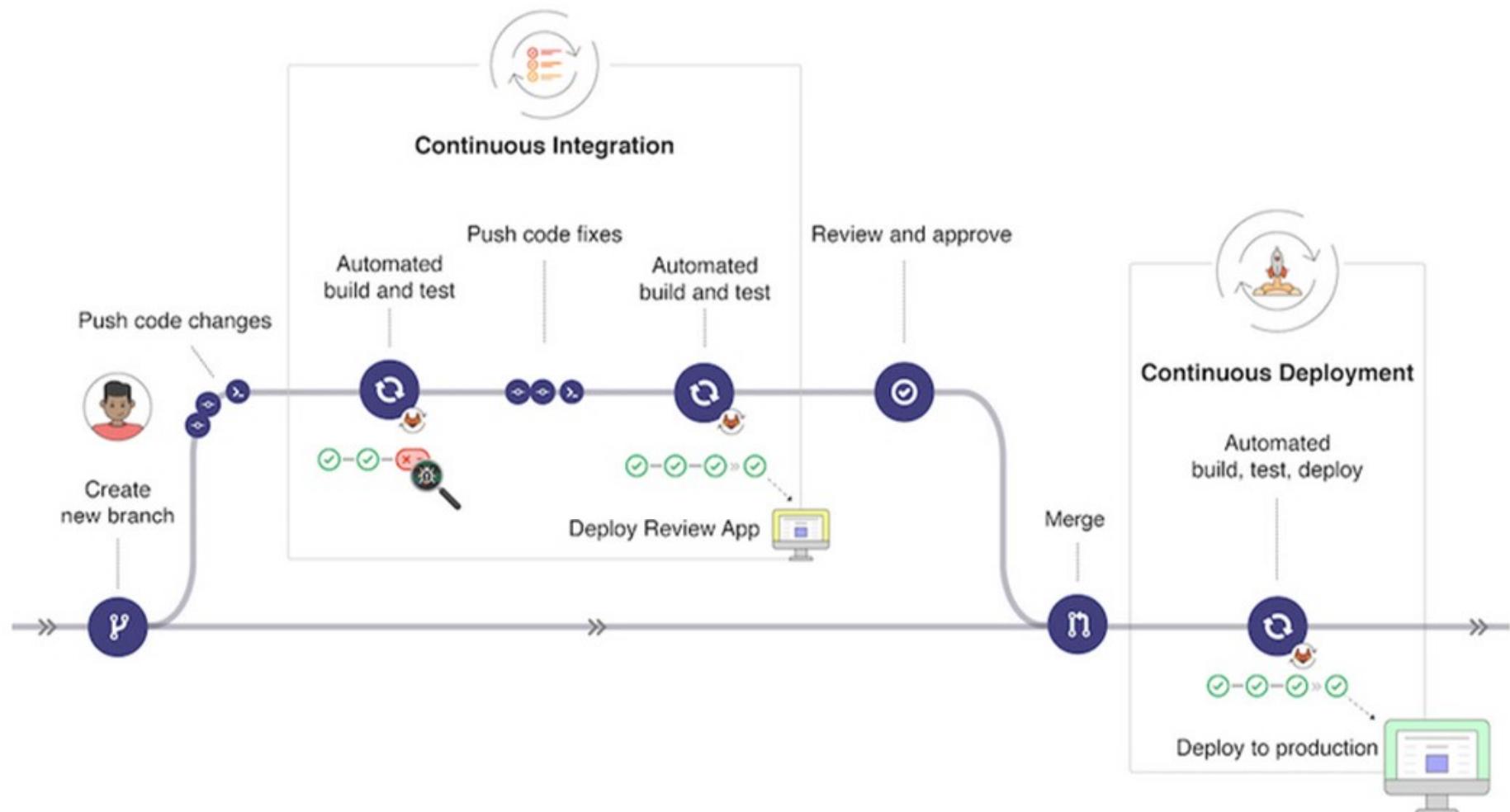
[Enable shared runners for this project](#)



Available shared runners: 78

 #1506020 (HsRmheX51)

GITLAB CI/CD | PIPELINE GENÉRICO / TÍPICO



GITLAB CI/CD | .gitlab-ci.yml

- O ficheiro `.gitlab-ci.yml` é responsável pelo processo CI/CD no GitLab (deve existir na *root* do projeto).
- O ficheiro pode ter outro nome, mas é usual manter o que é sugerido de base pelo GitLab. Quando o ficheiro existe na *root* do projeto, sabe-se que é sujeito a *Continuous Integration* e/ou *Continuous Delivery* do GitLab.
- O nível de CI/CD de um projeto depende das tarefas implementadas pelas equipas de desenvolvimento dentro do ficheiro `.gitlab-ci.yml`.
- Dentro deste ficheiro, é especificada a lista e sequência de tarefas a realizar para fazer o CI/CD. Respeita o formato YAML, e tem uma sintaxe específica do GitLab (que será abordada neste documento).
- Trata-se de um ficheiro texto e, portanto, pode ser editado com qualquer editor de texto. No entanto, deve respeitar o formato YAML e a sintaxe do GitLab. Por isso, O GitLab disponibiliza na definição do *pipeline* um editor que valida a sintaxe do ficheiro, antes de este ser gravado no repositório.
- O GitLab disponibiliza vários ficheiros *template* de *pipeline* que podem ser consultados no link abaixo.
 - <https://gitlab.com/gitlab-org/gitlab-foss/-/tree/master/lib/gitlab/ci/templates>

GITLAB CI/CD | .gitlab-ci.yml | KEYWORDS (1/2)

- O GitLab disponibiliza documentação acerca das palavras reservadas e respetivas regras de utilização, que podem ser utilizadas na definição de um ficheiro `.gitlab-ci.yml`. O quadro seguinte resume alguns desses termos. No entanto, para referência futura, convém ter o link abaixo aquando da configurações de ficheiros YAML.

➤ <https://docs.gitlab.com/ee/ci/yaml/index.html>

Termo	Descrição
<code>before_script</code>	Especifica uma lista de comandos que devem ser executados antes de um job (e.g. criar diretoria).
<code>after_script</code>	Especifica uma lista de comandos a serem executados depois de cada job (e.g. limpeza de ficheiros).
<code>needs</code>	Permite especificar precedências (e.g. <code>needs: job1</code> , indica que o job atual só executa depois do job1).
<code>dependencies</code>	Tem um comportamento parecido com o <code>needs</code> .
<code>only / except</code>	São <i>keywords</i> utilizadas para indicar se um job pode ou não ser adicionado ao pipeline a executar. O termo <code>only</code> deve ser utilizado para definir quando um job deve ser adicionado ao pipeline, e o <code>except</code> é utilizado para definir quando é que o job não deve ser adicionado ao pipeline.
<code>rules</code>	É uma alternativa à utilização do <code>only</code> e do <code>except</code> . Pode ser utilizado para adicionar ou excluir jobs do pipeline mediante determinadas condições.

GITLAB CI/CD | .gitlab-ci.yml | KEYWORDS



Termo	Descrição
image	Indica uma imagem Docker a ser utilizada.
allow_failure	É admissível que um job falhe. Uma falha neste job não é considerada para o resultado final do job.
when	Define quando é que um job executa: <code>on_success</code> , <code>on_failure</code> , <code>always</code> ou <code>manual</code> .
script	Define um <i>shell script</i> dentro do job, para indicar o que o runner deve executar.
cache	Define uma lista de ficheiros que devem ser mantidos em cache para as execuções posteriores.
environment	Define o nome do ambiente para o qual o <i>deployment</i> deve ser executado pelo job atual.
retry	Define quantas tentativas um job deve fazer antes de considerar o seu trabalho como falhado.

GITLAB CI/CD | .gitlab-ci.yml | Exemplo 1 (1/2)

- A imagem ao lado, apresenta-se um ficheiro exemplo `.gitlab-ci.yml`.
- No primeiro bloco define-se o conjunto de `stages` existente no *pipeline*. Repare-se que cada `job` é identificado com o `stage` a que pertence. Por exemplo, o `job build-job` pertence ao `stage build`.
- No bloco `variables` definem-se as variáveis a utilizar no *pipeline*.
- Repare que os restantes blocos são `jobs`. Cada `job` identifica o `stage` a que pertence (e.g. o `job test-code`, pertence ao `stage test`).
- Dentro de cada `job`, existe um sub-bloco chamado `script` que contém as instruções que serão realmente executadas em cada `job`.
- Reparar também que, o `job deploy-code` só pode ser executado depois do `build-job`, e o `test-code` só pode ser executado depois do `deploy-code` (instrução `needs`).

```
stages:  
  - build  
  - deploy  
  - test  
  
variables:  
  RAILS_ENV: "test"  
  NODE_ENV: "test"  
  GIT_STRATEGY: "clone"  
  CHROME_VERSION: "103"  
  DOCKER_VERSION: "20.10.14"  
  
build-job:  
  stage: build  
  script:  
    - echo "build binaries/docker image..."  
    - node -v  
    - bash buildScript.sh  
  
deploy-code:  
  stage: deploy  
  needs: build-job  
  script:  
    - echo "code deploy..."  
    - cd to/your/desired/folder  
    - bash deployScript.sh  
  
test-code:  
  stage: test  
  needs: deploy-code  
  script:  
    - echo "run tests..."  
    - cd to/your/desired/folder  
    - npm run test
```

GITLAB CI/CD | .gitlab-ci.yml | Exemplo 1 (2/2)

- A ordem dos stages é importante. Os stage funcionam como rótulos, e identificam a ordem pelos quais os jobs devem executar.
- No exemplo, primeiro serão executados os job do stage build, depois serão executados os do stage deploy e por fim os do stage test.
- As precedências de execução podem ser definidas através da keyword needs. Pode também ser utilizada a keyword dependencies para definir as precedências de execução.
- Cada job tem obrigatoriamente um script que define o que executar e a ordem e execução dentro desse job.

```
stages:  
  - build  
  - deploy  
  - test  
  
variables:  
  RAILS_ENV: "test"  
  NODE_ENV: "test"  
  GIT_STRATEGY: "clone"  
  CHROME_VERSION: "103"  
  DOCKER_VERSION: "20.10.14"  
  
build-job:  
  stage: build  
  script:  
    - echo "build binaries/docker image..."  
    - node -v  
    - bash buildScript.sh  
  
deploy-code:  
  stage: deploy  
  needs: build-job  
  script:  
    - echo "code deploy..."  
    - cd to/your/desired/folder  
    - bash deployScript.sh  
  
test-code:  
  stage: test  
  needs: deploy-code  
  script:  
    - echo "run tests..."  
    - cd to/your/desired/folder  
    - npm run test
```

GITLAB CI/CD | .gitlab-ci.yml | Exemplo 2

- No *pipeline* definido pelo `.gitlab-ci.yml` da imagem, é utilizada uma imagem Docker preparada para executar com a última versão do .Net.
- Trata-se de um pipeline simples que faz o *publish* do projeto `MyProject`, mas antes faz os testes necessários previstos no respetivo projeto de testes – `MyProject.Test.csproj`.
- Trata-se um *pipeline* simples que executa os testes definidos no projeto `MyProject.Test` e, posteriormente, executa a publicação do projeto `MyProject`.
- Notar a regra `only` do job `release`. O job `release` apenas é executado quando o branch for o `master` e o seu *output* será guardado na pasta `publish`. Isto significa que, são executados testes sempre que o código é submetido ao repositório, mas o *publish* só é executado quando o código chega ao branch `master`. Quando o código é publicado, os ficheiros produzidos ficam na pasta `publish`.

```
image: microsoft/dotnet:latest

stages:
  - test
  - deploy

debug:
  stage: test
  script:
    - dotnet test MyProject.Test/MyProject.Test.csproj

release:
  stage: deploy
  only:
    - master
  artifacts:
    paths:
      - publish/
  script:
    # relative path to csproj-file
    - dotnet publish -c Release -o ../publishMyProject/MyProject.csproj
```

GITLAB CI/CD | .gitlab-ci.yml



GITLAB CI/CD | Demonstração 1



- Nesta demonstração é criado um *pipeline* muito simples que, na realidade, não integra, nem testa, nem publica código. O objetivo é tentar demonstrar de forma simples o funcionamento de um pipeline criado no GitLab. Crie ou reutilize um qualquer projeto do GitLab para realizar esta demonstração.
- Crie o ficheiro `.gitlab-ci.yml` apresentado na imagem, na *root* do projeto. Pode criar o ficheiro utilizando um editor local, a opção *New file* do repositório, ou então a opção: [Code | Pipeline editor](#) disponível após selecionar o projeto.
- Trata-se de um *pipeline* que, apenas, produz *output*. Serve para testar o funcionamento do pipeline. Para aceder ao *output* produzido por cada job, clique em cima do nome.

Running Célio Carvalho created pipeline for commit `aaa2cb16` created just now

For `main`

latest 4 Jobs In progress, queued for 0 seconds

Pipeline	Needs	Jobs	Tests
<code>build</code>		4	0

build test deploy

build-job test-job1 deploy-job
 test-job2

```

GETTING SOURCE FROM GIT REPOSITORY
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/smart-cities-dev-team/traffic-operations-front
end/.git/
Created fresh repository.
Checking out aaa2cb16 as detached HEAD (ref is main)...
Skipping Git submodules setup
$ git remote set-url origin "${CI_REPOSITORY_URL}"
Executing "step_script" stage of the job script
Using docker image sha256:d0d66a4bcd... for ruby:3.1 with digest ruby@sha256:ce07ca486ea1589cd... 2e9de20726 ...
$ echo "testing 2..." 00:01
testing 2...
$ echo "sleeping 20s" 00:01
sleeping 20s
Cleaning up project directory and file based variables
Job succeeded
  
```

```

stages:
  - build
  - test
  - deploy

build-job:
  stage: build
  script:
    - echo "building...."

test-job1:
  stage: test
  script:
    - echo "testing 1..."

test-job2:
  stage: test
  script:
    - echo "testing 2..."
    - echo "sleeping 20s"

deploy-job:
  stage: deploy
  script:
    - echo "deploying..."
  
```

GITLAB CI/CD | Demonstração .Net [App1] (1/6)

- Nesta demonstração vai ser criada uma pequena aplicação **.Net core**.
 - ✓ Crie um projeto no GitLab com o nome App1.
 - ✓ Adicione o ficheiro `.gitignore` recomendado pelo GitLab para o VisualStudio (adicione o ficheiro no *dashboard* de projeto do GitLab).
 - ✓ Faça o clone do repositório remoto na sua máquina local.
- Depois de criar o repositório e sincronizá-lo com a máquina local, crie uma aplicação com o nome **App1** e framework **.Net7.0**. Para simplificar, certifique-se que a pasta principal do projeto é a mesma que a pasta onde fica o ficheiro `.sln`.
- A solução deve ser composta por dois projetos:
 - ✓ **App1** - com o código que resolve o problema (código principal); e
 - ✓ **App1.Test** – que testa o código do projeto principal (testa se o problema está a ser bem resolvido).

App1

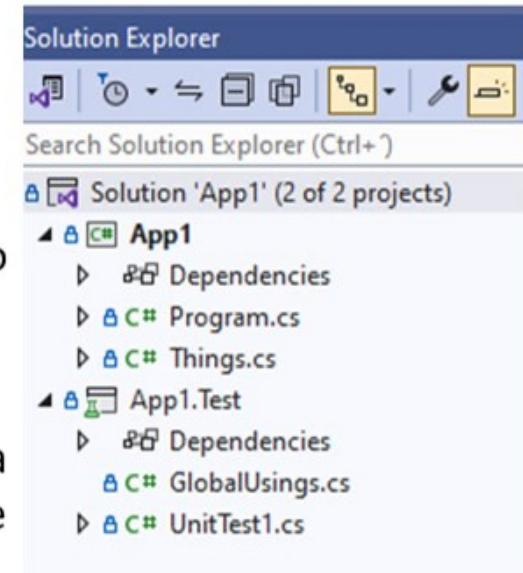
Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS

C# Linux macOS Windows Console

App1.Test

MSTest Test Project
A project that contains MSTest unit tests that can run on .NET on Windows, Linux and Mac OS.

C# Linux macOS Windows Test



Nome	Data de modificação	Tipo	Tamanho
.git	02/10/2023 14:54	Pasta de ficheiros	
.vs	02/10/2023 14:56	Pasta de ficheiros	
App1	02/10/2023 14:40	Pasta de ficheiros	
App1.Test	02/10/2023 15:11	Pasta de ficheiros	
TestResults	02/10/2023 15:15	Pasta de ficheiros	
.gitignore	02/10/2023 14:49	Arquivo Fonte Git ...	
App1.sln	02/10/2023 14:58	Visual Studio Solu...	2 KB
README.md	02/10/2023 14:39	Arquivo Fonte Ma...	1 KB

continua no slide seguinte...

GITLAB CI/CD | Demonstração .Net [App1] (2/6)



- Apresenta-se abaixo o código de cada um dos ficheiros do projeto **App1**. Por questões de economia de espaço, não foi incluída comentaçāo de métodos e código.
- O ficheiro `program.cs` contém o código executado quando se pede a execução do projeto App (main).

```
Things.cs ✘ X
namespace App1
{
    6 references | 0 changes | 0 authors, 0 changes
    public class Things
    {
        3 references | 0/2 passing | 0 changes | 0 authors, 0 changes
        public string GetFullName(string firstName, string lastName)
        {
            if (firstName == null || lastName == null)
                throw new ArgumentNullException();

            if (lastName.Trim() == string.Empty)
                throw new ArgumentException("lastName cannot be empty", "lastName");

            return $"{lastName}, { (firstName.Trim() == string.Empty ? "-" : firstName) }";
        }
    }
}
```

continua no slide seguinte...

```
Program.cs ✘ X
// variables
using App1;

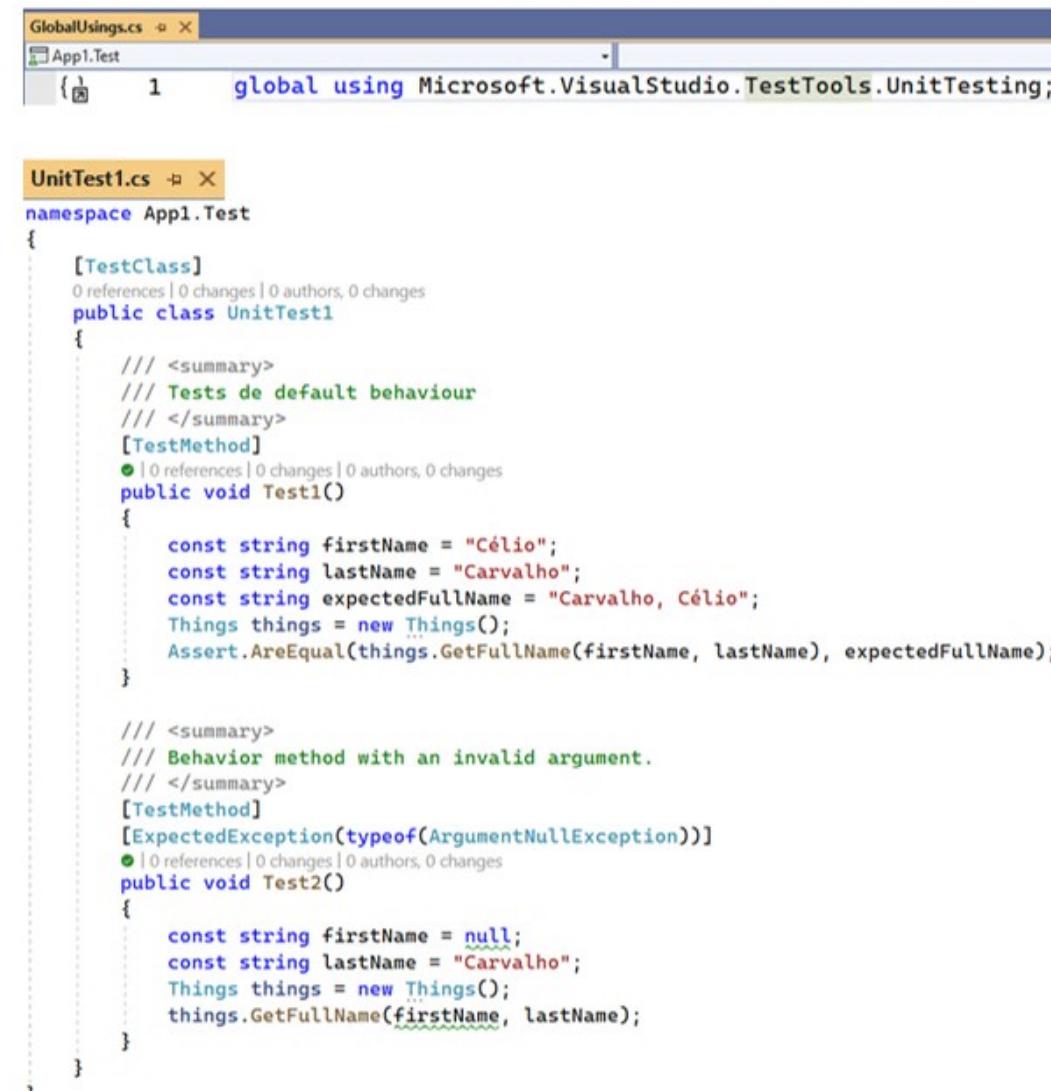
string firstName, lastName;
Things things;

// get data
Console.WriteLine("Enter your firstName: ");
firstName = Console.ReadLine() ?? "";
Console.WriteLine("Enter your lastName: ");
lastName = Console.ReadLine() ?? "";

try
{
    things = new Things();
    Console.Write("FullName: ");
    Console.WriteLine(things.GetFullName(firstName, lastName));
}
catch (Exception e)
{
    Console.WriteLine(e.ToString());
}
```

GITLAB CI/CD | Demonstração .Net [App1] (3/6)

- De seguida apresenta-se o projeto **App1.Test**. Este projeto será útil para pedir ao Visual Studio para fazer os testes, mas também será utilizado no *pipeline* do GitLab.
- Antes de continuar com a demonstração valide que o Visual Studio consegue executar os testes com sucesso.
- Test | Run All Tests
- Tente alterar o valor da constante `expectedFullName`.
- Execute novamente os testes, e verificará que irão falhar, pois o conteúdo devolvido pelo método não corresponde ao valor esperado.



The screenshot shows two open files in Visual Studio:

- GlobalUsings.cs**: A file containing the line: `global using Microsoft.VisualStudio.TestTools.UnitTesting;`
- UnitTest1.cs**: A file containing the following C# code:

```
namespace App1.Test
{
    [TestClass]
    public class UnitTest1
    {
        /// <summary>
        /// Tests de default behaviour
        /// </summary>
        [TestMethod]
        public void Test1()
        {
            const string firstName = "Célio";
            const string lastName = "Carvalho";
            const string expectedFullName = "Carvalho, Célio";
            Things things = new Things();
            Assert.AreEqual(things.GetFullName(firstName, lastName), expectedFullName);
        }

        /// <summary>
        /// Behavior method with an invalid argument.
        /// </summary>
        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        public void Test2()
        {
            const string firstName = null;
            const string lastName = "Carvalho";
            Things things = new Things();
            things.GetFullName(firstName, lastName);
        }
    }
}
```

continua no slide seguinte...

GITLAB CI/CD | Demonstração .Net [App1] (4/6)

- Retifique o código para que os testes executem com sucesso e, posteriormente, envie o código para o repositório remoto. Notar que o código não tem que ser submetido com os testes a serem bem sucedidos.
- Notar que, considerando o ambiente demonstrativo do CI/CD e por questões de simplicidade, não se está a atender às boas práticas de criar um novo branch para cada tarefa. No entanto, em contextos reais, tal seria o aconselhado como já referido anteriormente.
- Depois de validar que o conteúdo do projeto foi corretamente atualizado no projeto GitLab, crie o ficheiro `.gitlab-ci.yml` com o conteúdo apresentado na imagem.
- Note que está a ser usada uma imagem Docker com o **.Net 7.0**.

The screenshot shows the GitLab CI/CD interface. On the left, there is a code editor window displaying a `.gitlab-ci.yml` file with the following content:

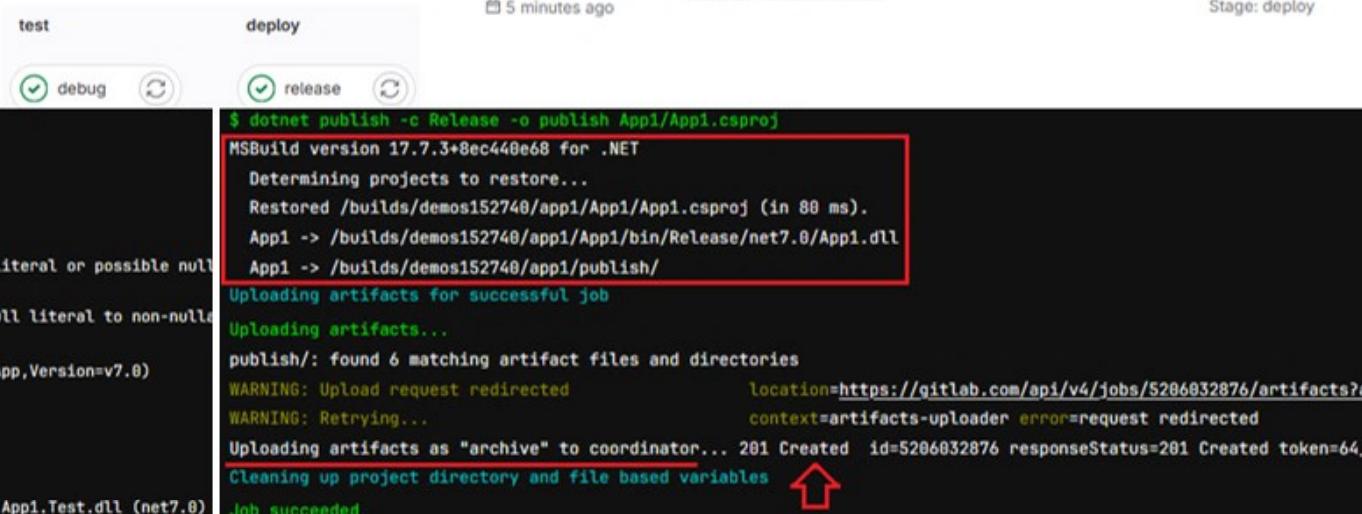
```
3 # see https://hub.docker.com/r/microsoft/dotnet/
4 image: mcr.microsoft.com/dotnet/sdk:7.0
5
6 stages:
7   - test
8   - deploy
9
10 release:
11   stage: deploy
12   only:
13     - main
14   artifacts:
15     paths:
16       - publish/
17   script:
18     # The output path is relative to the position of the csproj-file
19     - dotnet publish -c Release -o publish App1/App1.csproj
20
21 debug:
22   stage: test
23   script:
24     - dotnet test App1.Test/App1.Test.csproj
25
```

Below the code editor, there are two input fields: "Commit message" containing "Update .gitlab-ci.yml file" and "Branch" containing "main". At the bottom right, there are "Commit changes" and "Reset" buttons.

continua no slide seguinte...

GITLAB CI/CD | Demonstração .Net [App1] (5/6)

- Depois de se certificar que a sintaxe está correta, grave o ficheiro e consulte os *logs* dos vários jobs para garantir que tudo funcionou como esperado.



Status	Job	Pipeline
Passed 00:00:28 5 minutes ago	#5205712880: release main & 7933402e	#1023187795 created by Stage: deploy

```
$ dotnet test App1.Test/App1.Test.csproj
Determining projects to restore...
Restored /builds/demos152748/app1/App1/App1.csproj (in 143 ms).
Restored /builds/demos152748/app1/App1.Test/App1.Test.csproj (in 2.54 sec).
App1 -> /builds/demos152748/app1/App1/bin/Debug/net7.0/App1.dll
/builds/demos152748/app1/App1.Test/UnitTests1.cs(26,38): warning CS8600: Converting null literal or possible null
proj]
/builds/demos152748/app1/App1.Test/UnitTests1.cs(29,32): warning CS8625: Cannot convert null literal to non-null
App1.Test -> /builds/demos152748/app1/App1.Test/bin/Debug/net7.0/App1.Test.dll
Test run for /builds/demos152748/app1/App1.Test/bin/Debug/net7.0/App1.Test.dll (.NETCoreApp,Version=v7.0)
Microsoft (R) Test Execution Command Line Tool Version 17.7.1 (x64)
Copyright (c) Microsoft Corporation. All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
Passed! - Failed: 0, Passed: 2, Skipped: 0, Total: 2, Duration: 34 ms - App1.Test.dll (net7.0)

$ dotnet publish -c Release -o publish App1/App1.csproj
MSBuild version 17.7.3+8ec440e68 for .NET
Determining projects to restore...
Restored /builds/demos152748/app1/App1/App1.csproj (in 80 ms).
App1 -> /builds/demos152748/app1/App1/bin/Release/net7.0/App1.dll
App1 -> /builds/demos152748/app1/publish/
Uploading artifacts for successful job
Uploading artifacts...
publish/: found 6 matching artifact files and directories
WARNING: Upload request redirected
location=https://gitlab.com/api/v4/jobs/5206832876/artifacts?ar
WARNING: Retrying...
context=artifacts-uploader error=request redirected
Uploading artifacts as "archive" to coordinator... 201 Created id=5206832876 responseStatus=201 Created token=64_
Cleaning up project directory and file based variables
Job succeeded
```

- Aceda ao menu Build | Artifacts, e faça download do ficheiro **artifacts.zip**.
- De seguida, descompacte a pasta, e execute a aplicação com o comando abaixo.
➤ **dotnet App1.dll**

continua no slide seguinte...

GITLAB CI/CD | Demonstração .Net [App1] (6/6)



- Por fim, simule um erro na aplicação. Altere o código do método `Test1()` para esperar um valor diferente do que realmente será devolvido (ver exemplo na imagem).
- Submeta novamente o código ao repositório. Logo que acabar de fazer o push para o servidor, o pipeline de testes é invocado porque o push foi feito para o branch `main` (a regra do *pipeline* aguarda precisamente alterações neste branch).

Status	Pipeline	Created by	Stages
Running	Merge branch 'main' of https://gitlab.com/dem... #1023261456 ↗ main ➔ a4d63c84		
Passed ⌚ 00:01:04 🕒 15 minutes ago	Update .gitlab-ci.yml file #1023247453 ↗ main ➔ fd556a3d		

- Sem surpresas, o job `debug` falhou porque os testes definidos falharam.

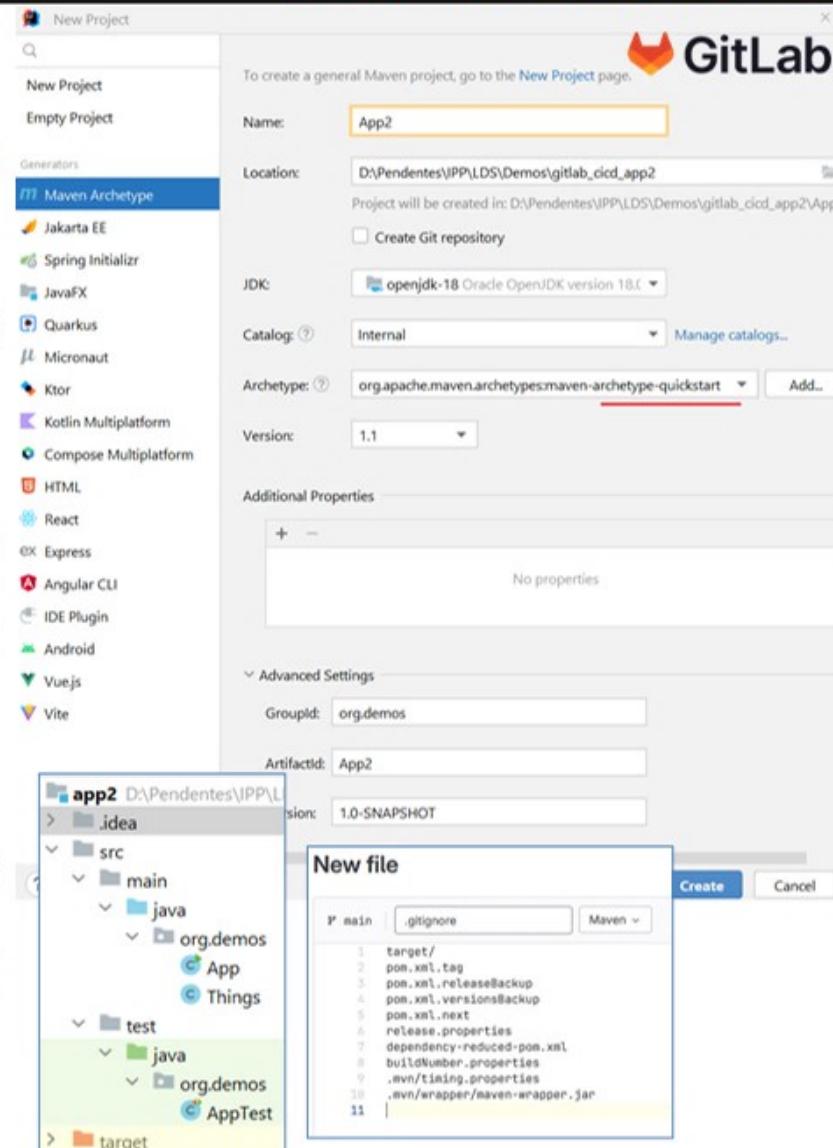
Pipeline	Needs	Jobs	Failed Jobs	Tests	Status	Pipeline	Created by	Stages
test		deploy			Failed ⌚ 00:00:34 🕒 4 minutes ago	Merge branch 'main' of https://gitlab.com/dem... #1023261456 ↗ main ➔ a4d63c84		
		debug			Passed ⌚ 00:01:04 🕒 20 minutes ago	Update .gitlab-ci.yml file #1023247453 ↗ main ➔ fd556a3d		

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
Failed Test1 [22 ms]
Error Message:
Assert.AreEqual failed. Expected:<Carvalho, Célio>. Actual:<Carvalho, Manuel>.
Stack Trace:
at App1.Test.UnitTest1.Test1() in /builds/demos152740/app1/App1.Test/UnitTests.cs:line 13
at System.RuntimeMethodHandle.InvokeMethod(Object target, Void** arguments, IntPtr method, IntPtr typeFilter)
at System.Reflection.MethodInvoker.Invoke(Object obj, IntPtr args, BindingFlags invokeAttr)
Failed! - Failed: 1, Passed: 1, Skipped: 0, Total: 2, Duration: 46ms

GITLAB CI/CD | Demonstração Java (Maven) [App2] (1/6)

- Nesta demonstração vai ser criada uma pequena aplicação Java utilizando Maven.
 - ✓ Crie um projeto no GitLab com o nome App2.
 - ✓ Adicione o ficheiro `.gitignore` recomendado pelo GitLab para Maven (adicione o ficheiro no *dashboard* de projeto).
 - ✓ Faça o clone do repositório remoto na sua máquina local.
- Depois de criar o repositório e sincronizá-lo com a máquina local, crie uma aplicação com o nome **App2** utilizando o archetype-quickstart. Para simplificar, certifique-se que a pasta principal do projeto é a mesma que a pasta onde fica o ficheiro `pom.xml`.
- O projeto deve conter as pastas:
 - ✓ **src/main/java** - com o código que resolve o problema (código principal); e
 - ✓ **src/test/java** – que testa o código do projeto principal (testa se o problema está a ser bem resolvido).

continua no slide seguinte...



GITLAB CI/CD | Demonstração Java (Maven) [App2] (2/6)

- Apresenta-se abaixo o código de cada um dos ficheiros do projeto **App2**. Por questões de economia de espaço, não foi incluída comentação de métodos e código.
- O ficheiro App.java contém o código a executar para iniciar a aplicação principal.

```
public class Things {  
  
    2 usages new *  
    public String getFullName(String firstName, String lastName)  
        throws IllegalArgumentException  
    {  
        if (firstName == null || lastName == null)  
            throw new IllegalArgumentException();  
  
        if (lastName.isEmpty())  
            throw new IllegalArgumentException("lastName cannot be empty");  
  
        return String.format("%s, %s",  
            lastName, firstName.isEmpty() ? "-" : firstName);  
    }  
}
```

```
import java.util.Scanner;  
  
no usages new *  
public class App  
{  
    no usages new *  
    public static void main( String[] args )  
    {  
        // variables  
        Scanner scanner = new Scanner(System.in);  
        String firstName, lastName;  
        Things things;  
  
        System.out.println("Enter your firstName: ");  
        firstName = scanner.nextLine();  
        System.out.println("Enter your lastName: ");  
        lastName = scanner.nextLine();  
  
        try  
        {  
            things = new Things();  
            System.out.printf("Fullname: %s",  
                things.getFullName(firstName, lastName));  
        }  
        catch (Exception e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

continua no slide seguinte...



GITLAB CI/CD | Demonstração Java (Maven) [App2] (3/6)

- De seguida apresenta-se a classe AppTest onde estão implementados os testes unitários. Ao pedir a execução dessa classe, os testes unitários lá constantes vão ser executados. Estes testes serão, também, executados no pipeline do GitLab.
- Antes de continuar com a demonstração valide se os testes executam corretamente no ambiente local (e.g. execute o código da classe AppTest no editor de código).
- Tente alterar o valor da constante expectedFullName.
- Execute novamente os testes, e verificará que irão falhar, pois o conteúdo devolvido pelo método não corresponde ao valor esperado.
- Depois de fazer o teste anterior, volte ao ponto correto do código, e faça push para o repositório remoto.
- Note que, também nesta demonstração, não se está a criar um branch novo. No entanto, no ambiente real, deveria ser seguida essa boa prática.

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

1 usage new *
public class AppTest extends TestCase
{
    no usages new *
    public AppTest( String testName ) { super( testName ); }

    no usages new *
    public static Test suite() { return new TestSuite( AppTest.class ); }

    no usages new *
    public void test1()
    {
        final String firstName = "Célio";
        final String lastName = "Carvalho";
        final String expectedFullName = "Carvalho, Célio";
        Things things = new Things();
        assertEquals(things.getFullName(firstName, lastName),
                    expectedFullName);
    }
}
```

continua no slide seguinte...

GITLAB CI/CD | Demonstração Java (Maven) [App2] (4/6)

- Depois de validar que o conteúdo do projeto foi corretamente atualizado no projeto GitLab, crie o ficheiro `.gitlab-ci.yml` com o conteúdo apresentado na imagem. Note que:
 - Está a ser usada uma imagem Docker com a versão 3.9.4 do Maven;
 - Ativou-se a diretiva da `cache` para armazenar e recuperar dependências entre execuções;
 - Existem duas atividades de testes que são executadas em paralelo (está a utilizar-se o `test` e o `verify` para simular execução paralela).
 - Está a gerar-se `artifacts output` no `test`, `verify` e `package`.
 - O comando `mvn verify` produz artefactos diferentes, em resultado da maior quantidade de testes que faz em relação ao `mvn test`.
 - Existe 3 job que produzem artefactos neste pipeline.

NOTA: Para se configurar o `.gitlab-ci.yml`, tem que se conhecer o funcionamento do Maven (e.g. onde são guardados os vários `output`). Consulte documentação na internet (e.g. abaixo).

- <https://cmakkaya.medium.com/gitlab-ci-cd-1-building-a-java-project-using-maven-and-docker-within-the-gitlab-ci-pipeline-278feaf7ee12>

```
image: maven:3.9.4

stages:
  - build
  - test
  - package

variables:
  MAVEN_CLI_OPTS: "--batch-mode"
  MAVEN_OPTS: "-Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository"

cache:
  paths:
    - .m2/repository

build-job:
  stage: build
  script:
    - echo "Compiling the code..."
    - mvn $MAVEN_CLI_OPTS compile

unit-test-job1:
  stage: test
  script:
    - echo "Running unit tests..."
    - mvn $MAVEN_CLI_OPTS test # test
  artifacts:
    when: always
    paths:
      - target/

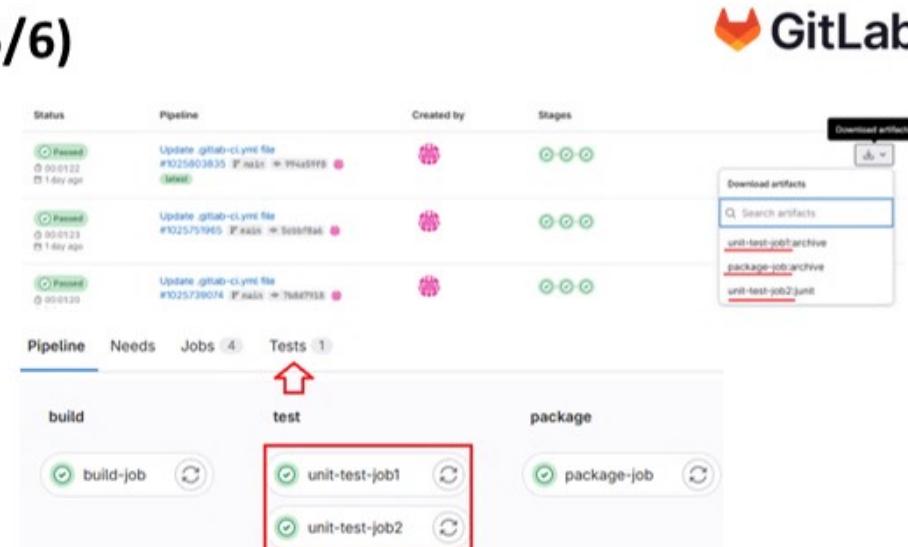
unit-test-job2:
  stage: test
  script:
    - echo "Running unit tests..."
    - mvn $MAVEN_CLI_OPTS verify # verify
  artifacts:
    when: always
    reports:
      junit:
        - target/surefire-reports/TEST-*.xml
        - target/failsafe-reports/TEST-*.xml

package-job:
  stage: package
  artifacts:
    paths:
      - target/
  script:
    - echo "Packaging application..."
    - mvn $MAVEN_CLI_OPTS package
```

continua no slide seguinte...

GITLAB CI/CD | Demonstração Java (Maven) [App2] (5/6)

- Depois de se certificar que a sintaxe está correta, grave o ficheiro e consulte os *logs* dos vários job.
- Note que existe uma secção para apresentar graficamente o conteúdo do ficheiro TEST-org.demos.AppTest.xml. Note também que os job unit-test-* foram executados em paralelo.
- Aceda ao menu Build | Arfacts, e faça o download do ficheiro artifacts.zip.
- Depois de descompactar a pasta, execute a aplicação com o comando abaixo.
➤ `java -cp App2-1.0-SNAPSHOT.jar org.demos.App`
(a opção -cp permite executar o ficheiro sem mencionar o 'Main-Class' no arquivo Manifest)



GITLAB CI/CD | Demonstração Java (Maven) [App2] (6/6)

- Por fim, simule um erro na aplicação. Altere o código do método `Test1()` para esperar um valor diferente do que realmente será devolvido (ver exemplo na imagem).
- Submeta novamente o código ao repositório. Logo que acabar de fazer o push para o servidor, o pipeline de testes é invocado porque o push foi feito para o branch `main` (a regra do pipeline aguarda precisamente alterações neste branch).

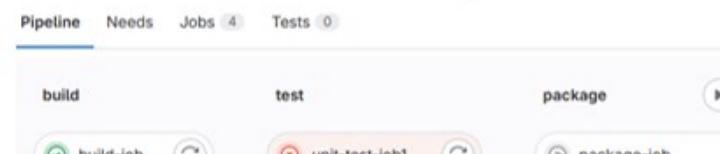
```
public void test1()
{
    final String firstName = "Célio";
    final String lastName = "Carvalho";
    final String expectedFullName = "Carvalho, Manuel";
    Things things = new Things();
    assertEquals(things.getFullName(firstName, lastName),
                expectedFullName);
}
```



Status	Pipeline	Created by	Stages
Running	Merge branch 'main' of https://gitlab.com/dem... #1027996940 ↗ main → 947c077b ⓘ latest	●	○ ○ ○
Passed 00:01:19 1 minute ago	Update .gitlab-ci.yml file #1027993795 ↗ main → 994a59f0 ⓘ	●	○ ○ ○

- Sem surpresas, os job de testes falharam porque os testes definidos no código não obtiveram os valores esperados. O erro pode ser consultado no GitLab, mas também é possível obter o detalhe através dos artefactos de erro gerados.

Status	Pipeline	Created by	Stages
Failed 00:00:51 2 minutes ago	Merge branch 'main' of https://gitlab.... #1027996940 ↗ main → 947c077b ⓘ latest	●	○ ○ ○



- Submeta novamente o código, desta vez com o código retificado. Certifique-se que o pipeline é executado com sucesso.

```
[ERROR] Failures:
[ERROR]   org.demos.AppTest#test1 ComparisonFailure expected:<...Célio> but was:<...Manuel>
[INFO]
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
```

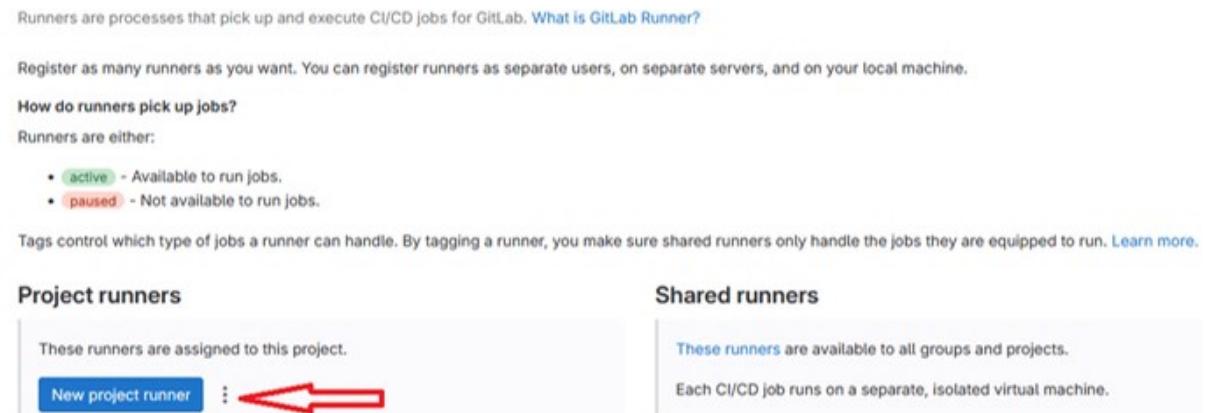
GITLAB CI/CD | RUNNER LOCAL | SETUP



- Há vários motivos que podem levar à decisão de se configurar *runners* locais. Um dos motivos pode ser o facto de não se querer estar dependente dos shared runners do GitLab. A lista de *runners* existentes e disponíveis, pode ser consultada na opção Settings | CI/CD | Runners.
- Um *runner* local permite que a execução de job dos pipeline, possa ser executado localmente, i.e., sem haver dependência dos *runners* disponibilizados pelo GitLab.
- Para que uma máquina externa ao GitLab (e.g. servidor) possa desempenhar o papel de *runner*, antes de mais, deve ter a aplicação GitLab Runner instalada, configurada e a funcionar.
- Neste contexto, antes de se proceder à demonstração de funcionamento do GitLab Runner, tem, antes de mais, fazer-se a configuração do GitLab Runner.
 - As instruções para instalação do GitLab Runner dependem do sistema operativo, e podem ser encontradas no link abaixo.
 - <https://docs.gitlab.com/runner/install/>

GITLAB CI/CD | RUNNER LOCAL | REGISTER (1/4)

- Após a instalação do GitLab Runner no ambiente local, tem que se efetuar o registo desse *runner* na instância GitLab à qual se pretende ligar. Esse passo é importante para que o GitLab consiga pedir trabalho ao *runner* configurado localmente (ver <https://docs.gitlab.com/runner/register/index.html>).
- O registo é o processo que liga o *runner* (local) a uma instância GitLab (estabelece um *link* entre o *runner* e a instância GitLab). Podem ligar-se vários *runners* no mesmo *host* à instância GitLab, através da repetição do comando `register`. Também se podem ligar *runners* a executar em diferentes máquinas (*host*), através da repetição do comando `register`, utilizando o mesmo *authentication token*.
- Durante o processo de registo do *runner*, será necessário estar na posse de um *token* gerado pelo GitLab.
- Esse *token* é gerado na opção abaixo.
➤ Settings | CI/CD | New project runner



Runners are processes that pick up and execute CI/CD jobs for GitLab. [What is GitLab Runner?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine.

How do runners pick up jobs?

Runners are either:

- `active` - Available to run jobs.
- `paused` - Not available to run jobs.

Tags control which type of jobs a runner can handle. By tagging a runner, you make sure shared runners only handle the jobs they are equipped to run. [Learn more.](#)

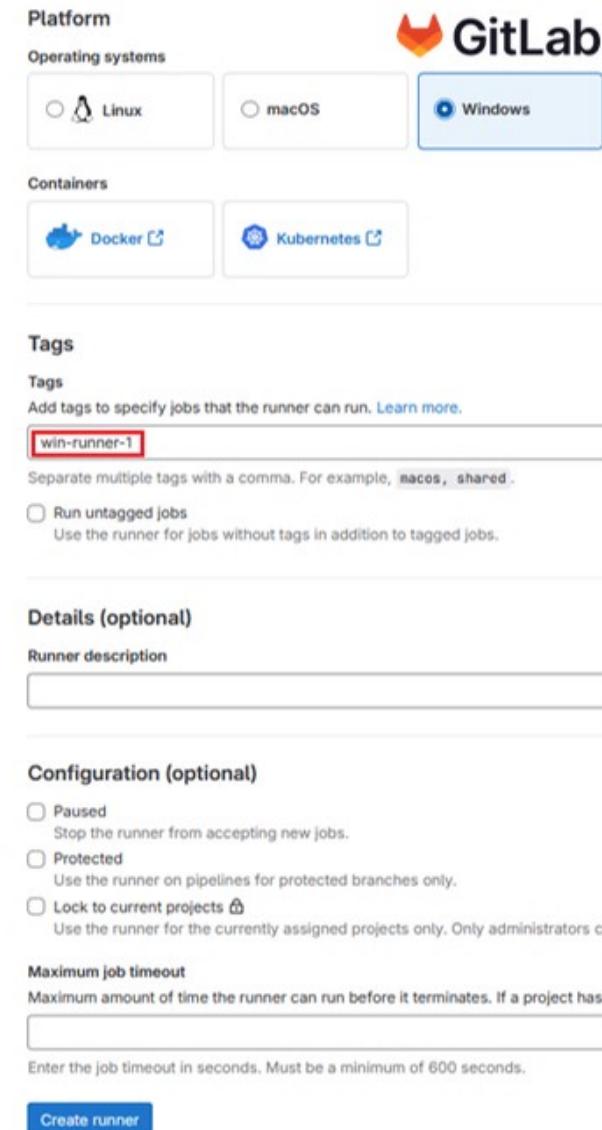
Project runners	Shared runners
These runners are assigned to this project.	These runners are available to all groups and projects.
New project runner : 	Each CI/CD job runs on a separate, isolated virtual machine.

continua no slide seguinte...

GITLAB CI/CD | RUNNER LOCAL | REGISTER (2/4)

- Nesta janela é possível identificar o sistema operativo ou *container* em que o *runner* poderá executar (e.g. Windows), e as tag que facilitarão a escolha pelo sistema para escolher este *runner* para executar job (não confundir estas tag com as tag do separador Code).
- Nas configurações do *runner*, também é possível:
 - Definir um *runner* como em pausa;
 - Indicar que um *runner* só pode ser usado em branch definidos como **protected**;
 - Indicar que o *runner* só está disponível para executar para determinados projetos; e o
 - *Timeout* para a execução de cada job.
- Posteriormente, na edição do *runner*, existem mais algumas opções de configuração. Por exemplo, é possível indicar se o *runner* pode ser executado em job que não referenciam tag de *runner* (esta configuração é utilizada à frente neste documento).

continua no slide seguinte...



The screenshot shows the 'Create runner' form on the GitLab interface. It includes sections for Platform (Linux, macOS, Windows - Windows is selected), Operating systems, Containers (Docker, Kubernetes), Tags (win-runner-1), Details (optional) (Runner description), and Configuration (optional) (Paused, Protected, Lock to current projects, Maximum job timeout). A 'Create runner' button is at the bottom.

Platform

Operating systems

Containers

Tags

Add tags to specify jobs that the runner can run. [Learn more](#).

win-runner-1

Separate multiple tags with a comma. For example, `macos, shared`.

Run untagged jobs

Use the runner for jobs without tags in addition to tagged jobs.

Details (optional)

Runner description

Configuration (optional)

Paused

Protected

Lock to current projects

Maximum job timeout

Create runner

GITLAB CI/CD | RUNNER LOCAL | REGISTER (3/4)

- Após a criação do *runner*, serão apresentadas as instruções para terminar o processo de registo do GitLab Runner na aplicação GitLab (ligar ambos os sistemas). Notar que os comandos seguintes, têm que ser executados na linha de comandos com **permissão de administrador**.

```
C:\GitLab-Runner>gitlab-runner.exe register --url https://gitlab.com --token glrt-XNwBfJ-xH...LYFF
Runtime platform
Created missing unique system ID
Enter the GitLab instance URL (for example, https://gitlab.com/):
[https://gitlab.com]: https://gitlab.com
Verifying runner... is valid
Enter a name for the runner. This is stored only in the local config.toml file:
[CC493]: my-local-runner-1
Enter an executor: instance, kubernetes, custom, docker, shell, ssh, docker-machine, docker-windows, parallels, virtualbox, docker-autoscaler:
shell
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!
Configuration (with the authentication token) was saved in "C:\GitLab-Runner\config.toml"
```

- O nome da instância deve ser o correto. O nome do *runner* pode ser dado livremente. Selecioneu-se o executor Shell porque, provavelmente, será o mais simples de utilizar. No entanto, é necessário ter o PowerShell Core (pwsh) instalado no ambiente local. O pwsh.exe foi compilado em .Net Core e é multiplataforma. O caminho para chegar ao executável tem que estar no %TEMP%.
 - <https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.3>
- O serviço GitLab Runner pode ser executado como um serviço. Neste caso, vai ser executado diretamente na linha de comandos para poder receber pedidos de execução de job a partir do GitLab.

continua no slide seguinte...

Register runner

GitLab Runner must be installed before you can register a runner. [How do I install GitLab Runner?](#)

Step 1

Copy and paste the following command into your command line to register the runner.

```
> .\gitlab-runner.exe register
--url https://gitlab.com
--token glrt-XNwBfJ-xH...LYFF
```

ⓘ The runner token glrt-XNwBfJ-xH...LYFF displays only for a short time, and is stored in the config.toml after you register the runner. It will not be visible once the runner is registered.

Step 2

Choose an executor when prompted by the command line. Executors run builds in different environments. Not sure which one to select? ⓘ

Step 3 (optional)

Manually verify that the runner is available to pick up jobs.

```
> .\gitlab-runner.exe run
```

This may not be needed if you manage your runner as a [system or user service](#) ⓘ.

[Go to runners page](#)

GITLAB CI/CD | RUNNER LOCAL | REGISTER (4/4)

- Depois do registo efetuado, a página dos *runners* identificará cada *runner* em execução. A imagem abaixo apresenta o *runner* acabado de registar. O círculo verde significa que está em execução no SO local.
- Notar que, caso se pretenda que um *runner* esteja disponível para outros projetos GitLab, tem que se ativar o *runner* para esse projeto (ver imagem abaixo)
- Navegue para outro projeto que tenha criado no GitLab, e visite novamente a página dos *runners*. Depois clique no botão Enable for this project.

The screenshot shows the 'Project runners' section of the GitLab interface. It includes a 'New project runner' button and an 'Enable for this project' button. A red arrow points to the 'Enable for this project' button.

Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. [What is GitLab Runner?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine.

How do runners pick up jobs?

Runners are either:

- active** - Available to run jobs.
- paused** - Not available to run jobs.

Tags control which type of jobs a runner can handle. By tagging a runner, you make sure shared runners only handle the jobs they are equipped to run. [Learn more](#).

Project runners

These runners are assigned to this project.

[New project runner](#) :

Shared runners

These runners are available to all groups and projects.

Each CI/CD job runs on a separate, isolated virtual machine.

Enable shared runners for this project

Assigned project runners

#28241822

[Edit](#) || [Remove runner](#)

(XNwBfJ-xH)

win-runner-1

Available shared runners: 78

#1506020 (Hs8mheX51)

windows-shared-runners-manager-1

[shared-windows](#) [windows](#) [windows-1809](#)

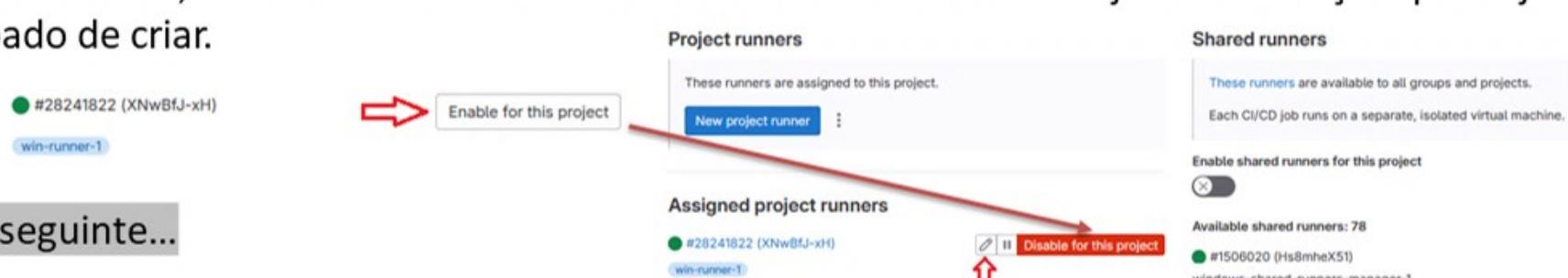
#1506021 (R7nvvEDvDr1)

- Para informações adicionais acerca de como gerir os *runners* visite a página indicada abaixo.
 - https://docs.gitlab.com/ee/ci/runners/runners_scope.html

GITLAB CI/CD | RUNNER LOCAL | Demonstração (1/3)

- Para que o *runner* local funcione, tem que ter instalado localmente o ambiente necessário para que os comandos funcionem corretamente. Considerando as demonstrações efetuadas anteriormente, o ambiente **.Net 7.0** tem que estar instalado para que o projeto App1 seja compilável no *runner* local acabado de configurar, assim como o **Maven** para a App2.
- Nesta demonstração, vai ser utilizada a App1 para testar a execução do pipeline com o *runner* local.
- Por defeito, um *runner* acabado de criar, apenas é utilizado quando existe uma tag definida no job a identificá-lo como o *runner* a utilizar. À frente, nesta demo, far-se-á a configuração de um job desta forma.
- Neste teste, vai-se autorizar o *runner* a ser escolhido pelo GitLab mesmo que o job não o refere. Utilize o botão editar (seta vermelha na imagem abaixo) para ativar a opção Indicates whether this runner can pick jobs without tags. Além disso, os Shared Runners serão desativados com o objetivo de forçar que seja usado o *runner* acabado de criar.

continua no slide seguinte...



The screenshot shows the 'Project runners' settings for a GitLab project. On the left, there's a list of runners assigned to the project, with one runner named 'win-runner-1' highlighted. A red arrow points from the 'Enable for this project' button to the 'Assigned project runners' section. In the center, the 'Assigned project runners' section shows the same runner 'win-runner-1'. A red arrow points from the 'Disable for this project' button to the 'Shared runners' section on the right. The 'Shared runners' section is currently disabled, indicated by a greyed-out toggle switch. The overall interface is light-colored with blue and grey UI elements.

GITLAB CI/CD | RUNNER LOCAL | Demonstração (2/3)

- Certifique-se que o serviço está em execução na linha de comandos. Notar que o *runner* local pode ser executado como serviço do SO. Se não estiver configurado dessa forma, o comando `gitlab-runner run` deve ser executado e mantido ativo.
- De seguida, execute o pipeline definido para a App1, e analise os resultados obtidos. Repare que o GitLab solicita a execução dos *script* ao *runner* local.

```
C:\GitLab-Runner>gitlab-runner.exe run
Runtime platform          arch=amd64 os=windows pid=37972 revision=d89a789a version=16.4.1
Starting multi-runner from C:\GitLab-Runner\config.toml...  builds=0 max_builds=0
Configuration loaded       builds=0 max_builds=1
listen_address not defined, metrics & debug endpoints disabled builds=0 max_builds=1
[session_server].listen_address not defined, session endpoints disabled builds=0 max_builds=1
Initializing executor providers
Checking for jobs... received
Added job to processing list
Appending trace to coordinator...ok
Job succeeded
Appending trace to coordinator...ok
Updating job...
Submitting job to coordinator...accepted, but not yet completed bytesize=2826 checksum=crc32:5525f756 code=202 job=5244028097 job-status=runner=
Updating job...
Submitting job to coordinator...ok
Removed job from processing list
Checking for jobs... received
Added job to processing list
Appending trace to coordinator...ok
Job succeeded
Appending trace to coordinator...ok
Updating job...
Submitting job to coordinator...accepted, but not yet completed bytesize=2516 checksum=crc32:fec5277e code=202 job=5244028098 job-status=runner=
Updating job...
Submitting job to coordinator...ok
Removed job from processing list
```

```
Running with gitlab-runner 16.4.1 (d89a789a)
on my-local-runner-1 XNbFJ-xH, system ID: s_4e19544#4a89
Preparing the "shell" executor
Using Shell (pwsh) executor...
Preparing environment
Running on CC493...
Getting source from Git repository
Fetching changes with git depth set to 20...
Initialized empty Git repository in C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/.git/
Created fresh repository.
Checking out 5813bbb4 as detached HEAD (ref is main)...
git-lfs/3.2.0 (GitHub; windows amd64; go 1.18.2)
Skipping Git submodules setup
Executing "step_script" stage of the job script
$ dotnet test App1.Test/App1.Test.csproj
Determining projects to restore...
Restored C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/App1/App1.csproj (in 77 ms).
Restored C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/App1.Test/App1.Test.csproj (in 741 ms).
App1 -> C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/App1/bin/Debug/net7.0/App1.dll
C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/App1.Test/UnitTest1.cs(26,38): warning CS8600: Converting
  type. [C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/App1.Test/App1.Test.csproj]
C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/App1.Test/UnitTest1.cs(29,32): warning CS8625: Cannot
  convert null literal to non-nullable reference type. [C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/App1.Test/App1.Test.csproj]
  App1.Test -> C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/App1.Test/bin/Debug/net7.0/App1.Test.dll
Test run for C:/GitLab-Runner/builds/XNbFJ-xH/0/demos152740/app1/App1.Test/bin/Debug/net7.0/App1.Test.dll (.NET Core)
Microsoft (R) Test Execution Command Line Tool Version 17.7.0-preview-23364-03+bc17bb9693fc4778ded51aa0ab7f10
Copyright (c) Microsoft Corporation. All rights reserved.
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
  Passed: 0, Failed: 2, Skipped: 0, Total: 2, Duration: 36 ms - App1.Test.dll (net7.0)
Cleaning up project directory and file based variables
Job succeeded
```



GITLAB CI/CD | RUNNER LOCAL | Demonstração (3/3)

- No teste seguinte o `.gitlab-ci.yml` vai ser reconfigurado no sentido de identificar o *runner* que se pretende executar (através da tag atribuída durante a criação).
- Antes de mais, os Shared Runners serão reativados. Como o *runner* a executar será indicado no pipeline, mesmo com os Shared Runners ativos, será invocado o *runner* local em detrimento de qualquer outro disponível no GitLab.
- Falta indicar no pipeline que deve utilizar o *runner* com a tag `win-runner-1` lembrar que foi este o nome ao *runner* aquando do registo do GitLab-Runner local.
- Altere o `.gitlab-ci.yml` em conformidade com o que é apresentado na imagem ao lado.
- Analise os resultados, e confirme que os Job foram executados no *runner* configurado localmente.



```
# see https://hub.docker.com/r/microsoft/dotnet/
image: mcr.microsoft.com/dotnet/sdk:7.0

stages:
  - test
  - deploy

release:
  stage: deploy
  tags:
    - win-runner-1
  only:
    - main
  artifacts:
    paths:
      - publish/
  script:
    # The output path is relative to the position of the csproj-file
    - dotnet publish -c Release -o publish App1/App1.csproj

debug:
  stage: test
  tags:
    - win-runner-1
  script:
    - dotnet test App1.Test/App1.Test.csproj
```

P. PORTO