



## Infraestrutura II

# Continuamos trabalhando com nosso Pipeline

É hora de agregar complexidade ao nosso Pipeline, com o objetivo de colocar em prática os conceitos de Integração Contínua e criação de artefatos a partir do nosso código.

## Objetivo final da prática

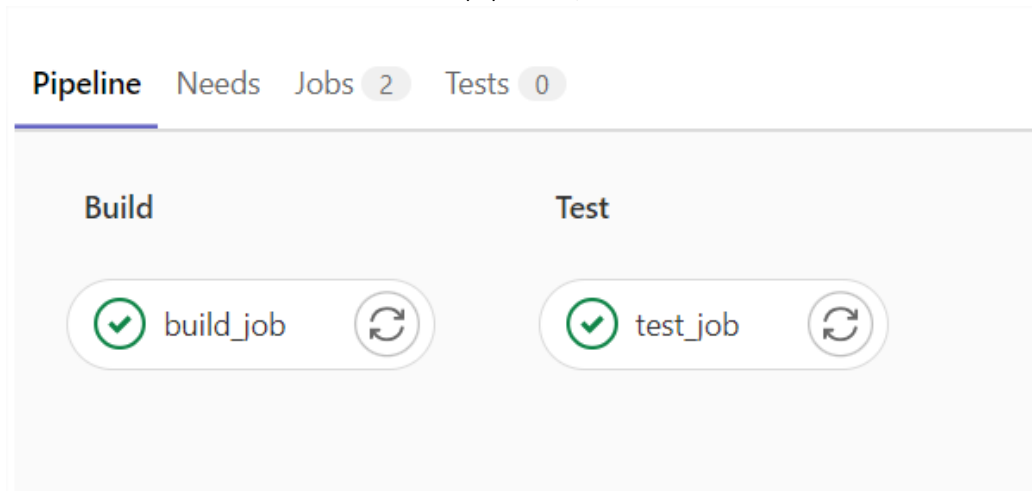
- Modificar nosso código com o objetivo de que nosso Pipeline falhe em diferentes etapas
- Conheça como o GitLab trata as falhas
- Criar e obter o artefato resultante do nosso código.

## Mãos à obra!

Ao final da prática anterior, foi proposto adicionar uma etapa de teste ao nosso Pipeline. Se você não conseguiu, vamos te passar como deve ficar o seu Pipeline

```
1 stages:
2   - build
3   - test
4
5 build_job:
6   stage: build
7   script:
8     - "mvn compile"
9
10 test_job:
11   stage: test
12   script:
13     - "mvn test"
```

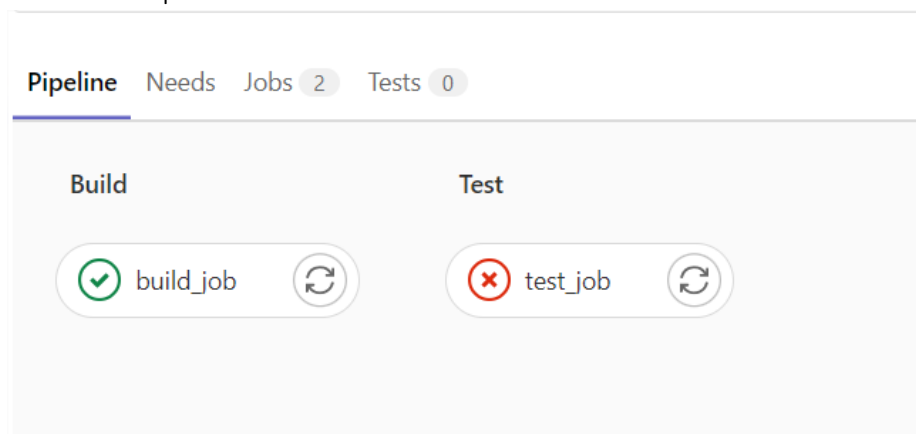
Nosso Pipeline deveria ter sido executado satisfatoriamente, então quando vemos seus detalhes devemos vê-lo assim (vemos no menu CI/CD Pipelines #número do pipeline)



Agora o que vamos fazer são modificações em alguma parte do nosso código-fonte com a finalidade de que nossos estágios falhem.

O que você mudaria para que o estágio de construção falhe? E para o teste?

Quando um Pipeline falha, neste exemplo fizemos o estágio de teste falhar, vamos ver em nosso resumo do Pipeline



E é aí que devemos investigar por que aconteceu, neste caso vamos para o test\_job e será aberto o verbose/debug de todo o processo, desde quando o ambiente para teste é gerado até quando o teste é finalizado.

```

521 [INFO]
522 [INFO] Results:
523 [INFO]
524 [ERROR] Failures:
525 [ERROR]   AppTest.shouldAnswerWithTrue:18
526 [INFO]
527 [ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
528 [INFO]
529 [INFO] -----
530 [INFO] BUILD FAILURE
531 [INFO] -----
532 [INFO] Total time: 56.185 s
533 [INFO] Finished at: 2022-03-31T12:33:55Z
534 [INFO] -----
535 [ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.1:test (default-test) on project dhinfra: There are test failures.
536 [ERROR]
537 [ERROR] Please refer to /builds/testdh/infra2v/target/surefire-reports for the individual test results.
538 [ERROR] Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date].dumpstream.
539 [ERROR] -> [Help 1]
540 [ERROR]
541 [ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
542 [ERROR] Re-run Maven using the -X switch to enable full debug logging.
543 [ERROR]
544 [ERROR] For more information about the errors and possible solutions, please read the following articles:
545 [ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
546 [ERROR] Cleaning up project directory and file based variables
547 ERROR: Job failed: exit code 1

```

test\_job

New issue

Duration: 1 minute 3 seconds  
 Finished: 3 minutes ago  
 Timeout: 1h (from project)  
 Runner: #1 (nRjbwKrU) GitLabDH

Commit 88db8701  
 Update  
 src/test/java/com/dhinfra/app/AppTest.java

Pipeline #134 for test

test


→ test\_job

Neste caso, temos um primeiro ERROR que está marcado nas linhas 524 em diante, explicando para neste caso que um "assertTrue" não atendeu à condição. Da mesma forma, isso sempre dependerá do idioma e de como programamos nossos testes.

O importante é que se uma etapa falhar, a próxima (caso tenha outras) NÃO SERÁ EXECUTADA.

## Como descubro que um pipeline falhou?

Por padrão, o GitLab nos notifica via email sobre o bug, por exemplo, para o bug acima este email chegou, onde temos todos os detalhes.



✖ Pipeline #622 has failed!

Project	Nidio Dolfini / infra2
Branch	test
Commit	9100f49f build test
Commit Author	Nidio Dolfini

Pipeline #622 triggered by Nidio Dolfini  
 had 0 failed jobs.

Failed jobs

mas também podemos olhar dentro do menu CI/CD   Pipelines   #número do pipeline



🕒 00:00:06

📅 3 hours ago

build test

[#628](#) 🔗 test 🔗 4231ebfd

latest

## Gerando artefatos

Já vimos como compilar e testar nosso código fonte, e sabemos que no caso haja alguma alteração nele, o GitLab aciona a execução do Pipeline, garantindo a INTEGRAÇÃO CONTÍNUA (CI).

Agora, para pensar em poder executar nosso aplicativo, não fizemos algo importante, empacotando-o, para depois executá-lo em 1 ou vários servidores.

Vamos continuar trabalhando com nosso código-fonte Java, para obter nosso arquivo .jar

+info: [JAR \(formato de arquivo\) – Wikipédia, a enciclopédia livre \(wikipedia.org\)](https://pt.wikipedia.org/wiki/JAR_(formato_de_arquivo))

Devemos adicionar um novo estágio ao nosso pipeline, depois de construir e testar , que chamaremos de “pacote”.

```
stages:
  - build
  - test
  - package
```

Como o objetivo desta etapa é gerar um arquivo, deve-se configurar um repositório onde o GitLab armazenará tal arquivo, para isso vamos configurar alguns itens adicionais ao Pipeline usando um repositório local.

```
variables:
  MAVEN_OPTS: -Dmaven.repo.local=.m2/repository

cache:
  paths:
    - .m2/repository
    - target
```

## Definindo o job do empacotamento

Em seguida, vamos definir nosso job do empacotamento

```
package_job:
  stage: package
  artifacts:
    paths:
      - target/*.jar
  script:
    - "mvn package"
```

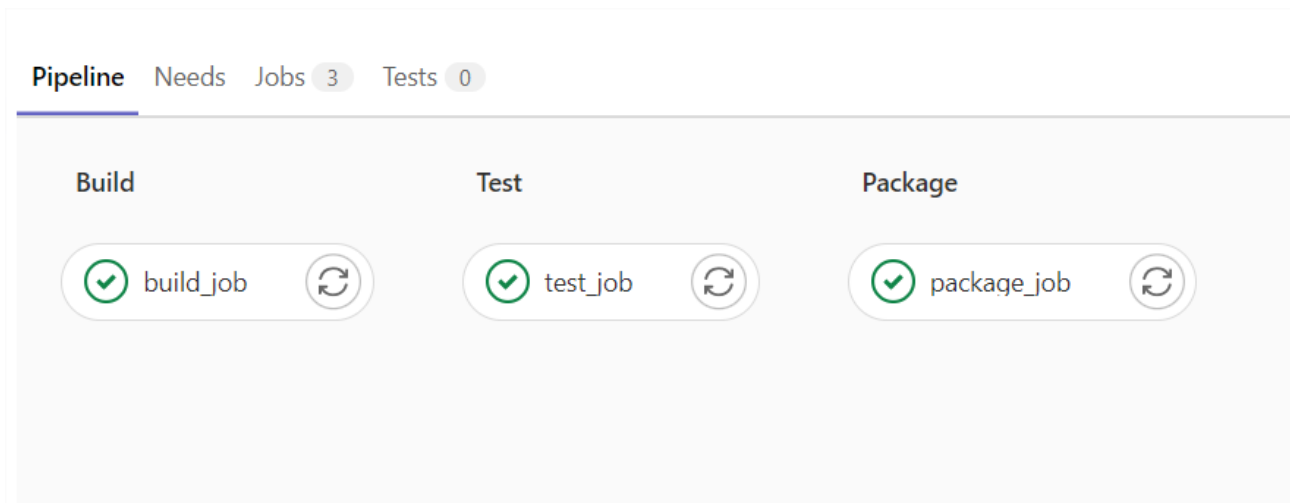
Alguns pontos a serem observados são:

- A adição de artefatos e definição do caminho onde nosso artefato será salvo.
- O comando para empacotar, no caso do Maven é "mvn package"

Nosso pipeline deve finalmente ser o seguinte:

```
1  stages:
2    - build
3    - test
4    - package
5
6  variables:
7    MAVEN_OPTS: -Dmaven.repo.local=.m2/repository
8
9  cache:
10   paths:
11     - .m2/repository
12     - target
13
14  build_job:
15    stage: build
16    script:
17      - "mvn compile"
18
19  test_job:
20    stage: test
21    script:
22      - "mvn test"
23
24  package_job:
25    stage: package
26    artifacts:
27      paths:
28        - target/*.jar
29
30    script:
31      - "mvn package"
```

Ao visualizar os detalhes de execução, vemos a adição do pacote stage.



Onde está  
nosso  
artefato?



Como havíamos configurado acima, o artefato será salvo em um repositório de artefatos local do GitLab, para acessá-lo, vamos em “package\_job” faça o download ou explore o repositório.

```

54 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.075 s - in com.dhinfra.app.AppTest
55 [INFO]
56 [INFO] Results:
57 [INFO]
58 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
59 [INFO]
60 [INFO]
61 [INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ dhinfra ---
62 [INFO] Building jar: /builds/testdh/infra2v/target/dhinfra-1.0-SNAPSHOT.jar
63 [INFO]
64 [INFO] BUILD SUCCESS
65 [INFO]
66 [INFO] Total time: 2.117 s
67 [INFO] Finished at: 2022-03-31T13:09:39Z
68 [INFO]
70 Saving cache for successful job
71 Creating cache default...
72 .m2/repository: found 972 matching files and directories
73 target: found 31 matching files and directories
74 No URL provided, cache will be not uploaded to shared cache server. Cache will be stored only locally.
75 Created cache
77 Uploading artifacts for successful job
78 Uploading artifacts...
79 target/*.jar: found 1 matching files and directories
80 Uploading artifacts as "archive" to coordinator... 201 Created id=199 responseStatus=201 Created token=Q6SHffxE
82 Cleaning up project directory and file based variables
84 Job succeeded

```

**Duration:** 11 seconds  
**Finished:** 7 minutes ago  
**Timeout:** 1h (from project)  
**Runner:** #1 (nRJbwKrU) GitLabDH

**Job artifacts**  
 These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Keep Download Browse

**Commit 3838eeb6**  
 Update .gitlab-ci.yml

✓ Pipeline #136 for test  
 package

→ package\_job



Se baixarmos, ele irá empacotar todos os artefatos gerados em um .zip e dentro dele estará nosso .jar

