



## Infraestructura II

# Continuamos trabajando con nuestro Pipeline

Es momento de ir agregándole complejidad a nuestro Pipeline, con el propósito de poner en práctica los conceptos de Integración Continua y creación de artefactos a partir de nuestro código.

### Objetivo final de la práctica

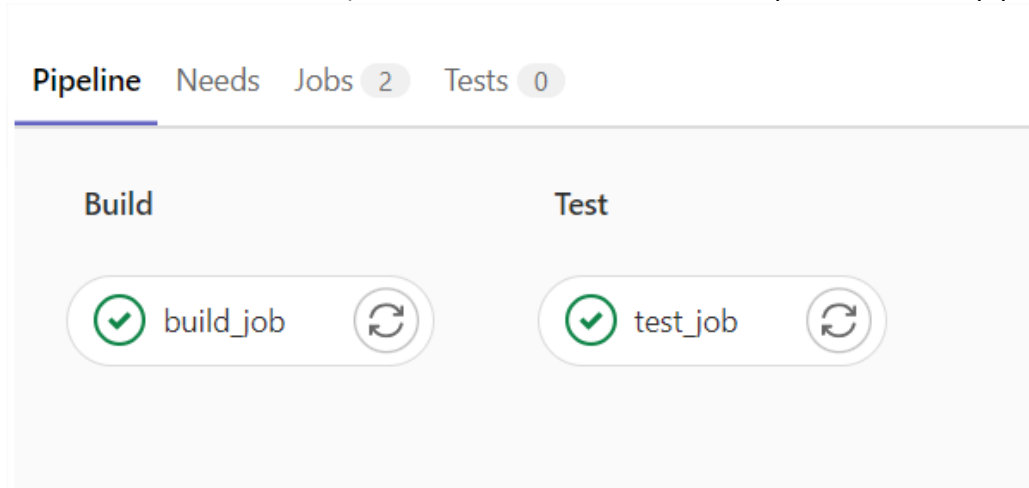
- Modificar nuestro código con el objetivo que nuestro Pipeline falle en diferentes stages
- Conocer como GitLab maneja los fallos
- Crear y obtener el artefacto resultante de nuestro código.

## ¡Manos a la obra!

En el final de la práctica anterior, se proponía agregar una stage de test a nuestro Pipeline. Si no lograste, te pasamos como debería verse tu Pipeline

```
1  stages:
2    |   - build
3    |   - test
4
5  build_job:
6    stage: build
7    script:
8    |   - "mvn compile"
9
10
11 test_job:
12   stage: test
13   script:
14   |   - "mvn test"
15
16
17
```

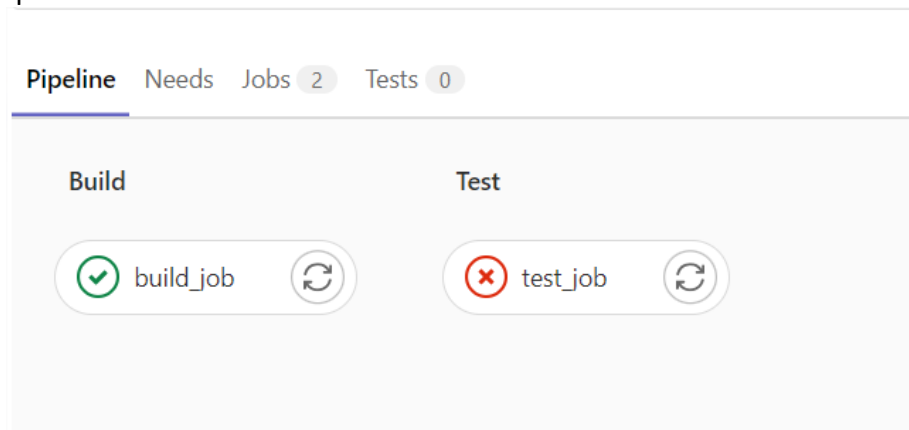
Nuestro Pipeline se debió ejecutar de manera satisfactoria, con lo cual al ver los detalles del mismos deberíamos verlo así (lo vemos en el menú CI/CD → Pipelines → # de pipeline)



Ahora lo que vamos a realizar son modificaciones en alguna parte de nuestro código fuente con el propósito que fallen nuestras stages.

**¿Qué cambiarías para que falle el stage de build? ¿Y para el de test?**

Cuando un Pipeline falla, en este ejemplo hicimos fallar el stage de test, vamos a ver en nuestro resumen del Pipeline



Y es allí cuando debemos investigar porque sucedió, en este caso nos vamos a dirigir al test\_job y nos abrirá el verbose / debug de todo el proceso, desde que se genera el entorno para las pruebas hasta el momento en que se prueba.

```

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: 2022-03-31T12:33:52Z
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
547 Cleaning up project directory and file based variables
549 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

En este caso, tenemos un primer ERROR que se marca en las líneas 524 en adelante, explicando para este caso que un “assertTrue” no cumplió con la condición. Igualmente, siempre esto dependerá del lenguaje y de cómo programemos nuestras pruebas.


Lo importante, es que si un stage falla, el siguiente (en este caso no lo había) NO SE EJECUTARA.

### ¿Cómo me entero de que un Pipeline fallo?


Por defecto, GitLab nos notifica vía correo electrónico de la falla, por ejemplo, para el fallo de más arriba llego este mail, en donde tenemos todos los detalles


GitLab <gitlab@gl.deitech.online>  
para mí

---



✖ Pipeline #134 has failed!

Project	Emilio Dorio / infra2v
Branch	/ test
Commit	<a href="#">88db8701</a> Update src/test/java/com/dhinfra/app/AppTest.java
Commit Author	 Emilio Dorio

Pipeline #134 triggered by  Emilio Dorio  
 had 1 failed job.

## Generando artefactos

Ya vimos como compilar y probar nuestro código fuente, y sabemos que ante cualquier cambio en el, GitLab dispara la ejecución del Pipeline, asegurando la INTEGRACION CONTINUA (CI).

Ahora, a fines de pensar en poder ejecutar nuestra aplicación, no hemos realizado algo importante, el empaquetado de este, con el propósito de luego, ejecutarlo en 1 o varios servidores.

Vamos a seguir trabajando con nuestro código fuente Java, con el propósito de obtener nuestro archivo .jar

+info: [https://es.wikipedia.org/wiki/Java\\_Archive](https://es.wikipedia.org/wiki/Java_Archive)

Debemos agregar una nueva stage a nuestro pipeline, posterior a build y test, la cual vamos a denominar “package”.

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

Como el propósito de esta stage es generar un archivo, se debe configurar un repositorio en donde GitLab almacenara dicho archivo, para ello vamos a configurar algunos ítems adicionales al Pipeline utilizando un repositorio local.

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

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

## Definiendo el trabajo de empaquetado

A continuación, vamos a definir nuestro trabajo de empaquetado

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

  script:
    - "mvn package"
```

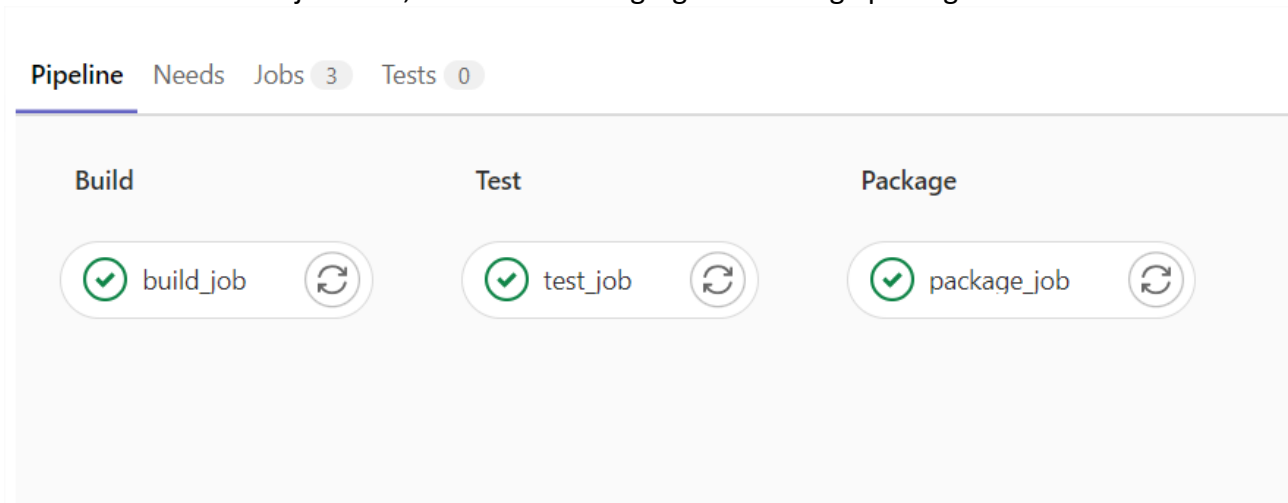
Algunos puntos para observar son:

- El agregado de artifacts, y dentro de ella, definimos el path en donde se guardará nuestro artefacto.
- El comando para empaquetar, para el caso de Maven es “mvn package”

Nuestro pipeline finalmente debe quedar de la siguiente manera:

```
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    script:
30      - "mvn package"
```

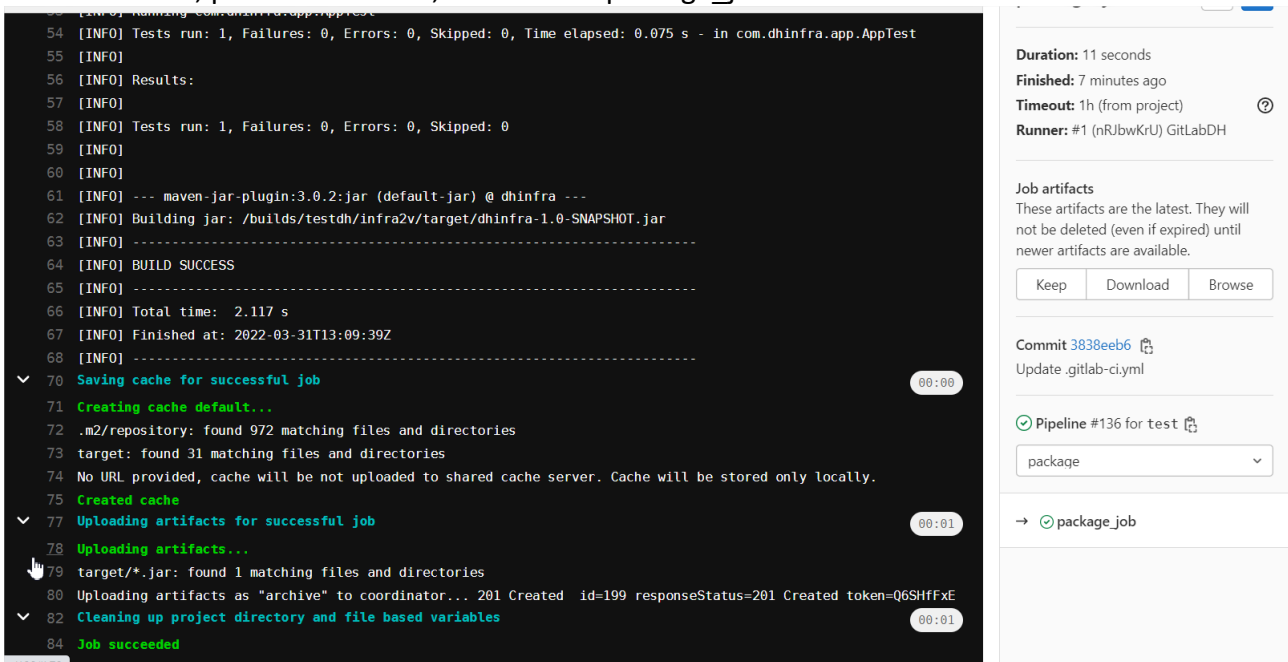
Al ver los detalles de ejecución, observamos el agregado del stage package.



**¿Dónde esta  
nuestro  
artefacto?**



Como habíamos configurado más arriba, el artefacto se guardara en un repositorio de artefactos local de GitLab, para acceder a él, vamos a el “package\_job”



Además de mostrarse los detalles de este trabajo, aparece una sección llamada “Job Artifacts” en donde podemos Conservarlo, Descargarlo o Explorar el repositorio.

Si descargamos, nos empaquetará en un .zip todos los artefactos generados y dentro del estará nuestro .jar

