

Activitat 3. Configuració d'un entorn d'Integració i entrega continuada amb Gitlab

Index

Activitat 3. Configuració d'un entorn d'Integració i entrega continuada amb Gitlab.....	1
1. Introducció.....	1
1.1. Preparació del projecte.....	2
2. Definició del primer pipeline.....	2
2.1 Fitxer .gitlab-ci.yml.....	3
2.2 Integració Continuada.....	4
Definició de les etapes.....	4
Definició de tasques.....	4
2.3 Entrega Continuada.....	8
Definició de la etapa.....	9
Definició de Variables i Secrets compartits.....	9
Definició de les tasques.....	11
3. Bibliografia / Webgrafia.....	13

1. Introducció

En aquesta pràctica veurem una **introducció a la ferramenta de CI/CD** proporcionada per **gitlab.com**. Aquesta ferramenta ens permet configurar un entorn d'integració i entrega contínua dels projectes allotjats als seus repositoris, d'aquesta forma ens proporciona una eina integral per a gestionar el desenvolupament i posada en marxa d'una aplicació.



Amb **CI/CD de GitLab** podem, en el mateix lloc, **crear tiquets, unir sol·licituds (pull requests), escriure codi i establir la integració i entrega continuada sense dependre d'una altra aplicació**. És bàsicament un viatge sense escales, **GitLab CI/CD** utilitza el que anomena **GitLab Runners** per executar cada un dels passos predefinit en el procés de integració del nou codi desenvolupat i el seu desplegament. **Estos runners són màquines virtuals** aïllades sobre les que s'executen els scripts y/o contenidors docker que anirem definint per dur a terme el procés de integració i/o desplegament.



Amb la finalitat d'entendre **tots els components** necessaris per a definir **l'entorn de CI/CD** afegirem algunes automatitzacions a un projecte base. Farem que es **disparen proves** sobre **el codi** quan es faça un **commit** i, automàticament, construirem el fitxer compilat de la aplicació exemple. Per finalitzar, veurem com desplegar la imatge en un servidor de producció real.



1.1. Preparació del projecte

Començarem fent un **fork** amb **visibilitat privada** del [següent repositori](https://gitlab.com/alecogi-edu/ddaw-ud6-a3.git) i pegant un ull al projecte que utilitzarem d'exemple.

```
$ git clone https://gitlab.com/alecogi-edu/ddaw-ud6-a3.git
```

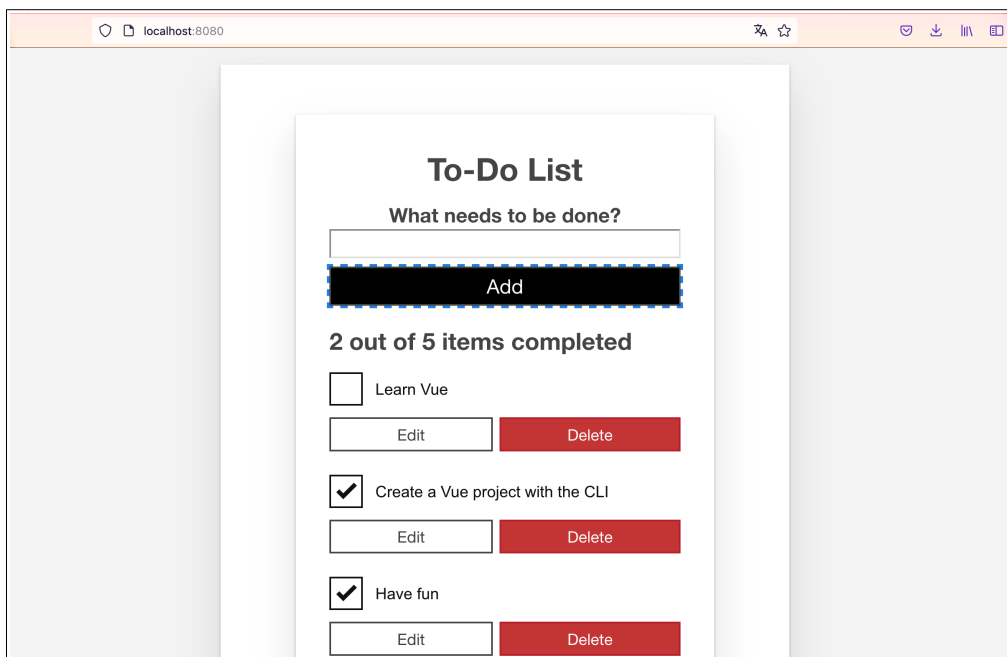
Per fer-lo funcionar, instal·larem les dependències

```
$ npm install
```

i executarem el servidor de desenvolupament que té integrat vue.js.

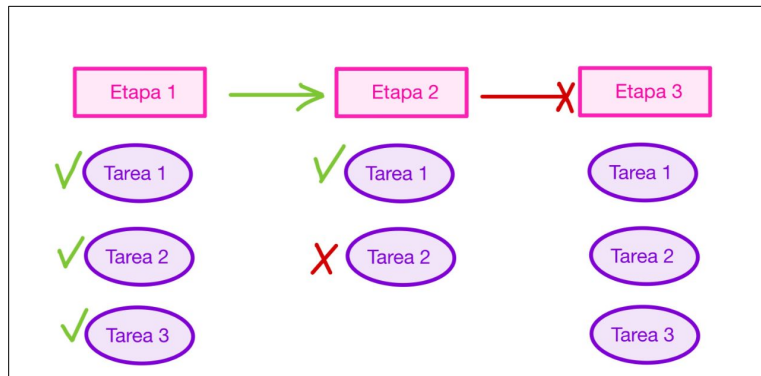
```
$ npm run serve
```

Si accedim ara a **localhost:8080** veurem el projecte a desplegar.



2. Definició del primer pipeline.

Tots els **repositoris de gitlab** tenen activada per defecte la capacitat d'executar un pipeline o conjunt de tasques. Per activar-ho sols hem de crear un fitxer **.gitlab-ci.yml** en la carpeta arrel del projecte i definir el conjunt de **etapes (steps)** i **tasques (tasks)** que han d'executar-se.



S'ha de tenir en compte que les tasques de les etapes no s'executaran fins que totes les tasques de l'etapa anterior hagen finalitzat la seua execució amb èxit. Un **pipeline** és la manera abreujada de referir-se a este **cúmulo d'etapes en un projecte**.

La **primera etapa** que crearem ens servirà per **crear el executable** de l'aplicació que hem de **desplegar**. Les subtasques que ha de dur a terme són:

Etapa 1

1. Crear un contenidor docker amb la imatge base de nodeJs
2. Clonar el repositori (aço ho fa automàticament)
3. Instal·lar les dependències
4. Crear l'executable

2.1 Fitxer `.gitlab-ci.yml`

Es tracta d'un **fitxer de text** amb **format yaml**. La primera opció per crear aquest fitxer, és directament al nostre entorn local utilitzant el IDE que gastem per desenvolupar el projecte i pujar-lo al repositori remot per a que s'active el mòdul d'integració contínua. En la [documentació oficial](#) podem consultar la seua sintaxi, [exemples](#) i plantilles bases per a aquest fitxer.

No obstant això, **gitlab** ens **proporciona** un **editor online** amb revisió de la sintaxi per a aquest fitxer que ens faciliten la tasca de definir les diferents etapes. Per accedir premem sobre el menú "Build → pipeline Editor".

2.2 Integració Continuada

Definició de les etapes

En primer lloc, cal dir que aquest fitxer presenta **yaml** com el que vam utilitzar a les darreres pràctiques. El primer que farem es definir aquestes 2 **etapes**:

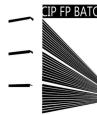
- **build**: Ens servirà per crear l'executable a desplegar.
- **test**: Executarà els test unitaris per verificar la correctivitat de l'aplicació.

.gitlab-ci.yml
<pre>stages: - build - test</pre>

Definició de tasques

Una vegada **definides les etapes a nivell global**, **especificarem les tasques** indicant-li la etapa a la que pertany (Una tasca pot pertànyer a diferents etapes). Editem el fitxer i afegim la tasca **build-job** encarregada de crear l'executable.

gitlab-ci.yml
<pre>stages: - build - test</pre>



```
build-job:
  stage: build
  image: node:21
  script:
    - npm install --progress=false
    - npm run build
  artifacts:
    expire_in: 1 week
    paths:
      - dist
```

A continuació analitzem cada una de les directives utilitzades:

- **build-job**: nom identificatiu de la tasca.
- **stage: build**: Etapa en la que volem que s'execute aquesta tasca.
- **image: node:21**: image base del repositori de docker hub en la que volem que s'execute la tasca
- **script**: Comandaments u ordres que han d'executar-se. En el nostre cas necessitem que s'instal·len les dependències `npm install --progress=false` i compilar el projecte `npm run build`.
- **artifacts**: Aquesta etiqueta fa referència al **producte final** (artefacte) construït. Hem especificat que la seua durabilitat siga de 1 setmana `expire_in: 1 week` i que el guardi en una carpeta anomenada `dist`.

```
paths:
  - dist
```

Una vegada definides les tasques, pujarem el nou fitxer al repositori, si hem utilitzat l'editor online sols haurem de ficar el missatge del commit i prémer sobre el botó "**Confirmar canvis**" de la interfície web que tenim a la part inferior de la pàgina.

Alexandre Coloma / DDAW-UD6-A3-SOL-V2 / Editor de pipeline

```
5 stage: build
6 image: node:21
7 script:
8   - npm install --progress=false
9   - npm run build
10 artifacts:
11   expire_in: 1 week
12   paths:
13     - dist
14
```

Mensaje del cambio

Actualizar el archivo .gitlab-ci.yml


Branch

main

Confirmar cambios Reset

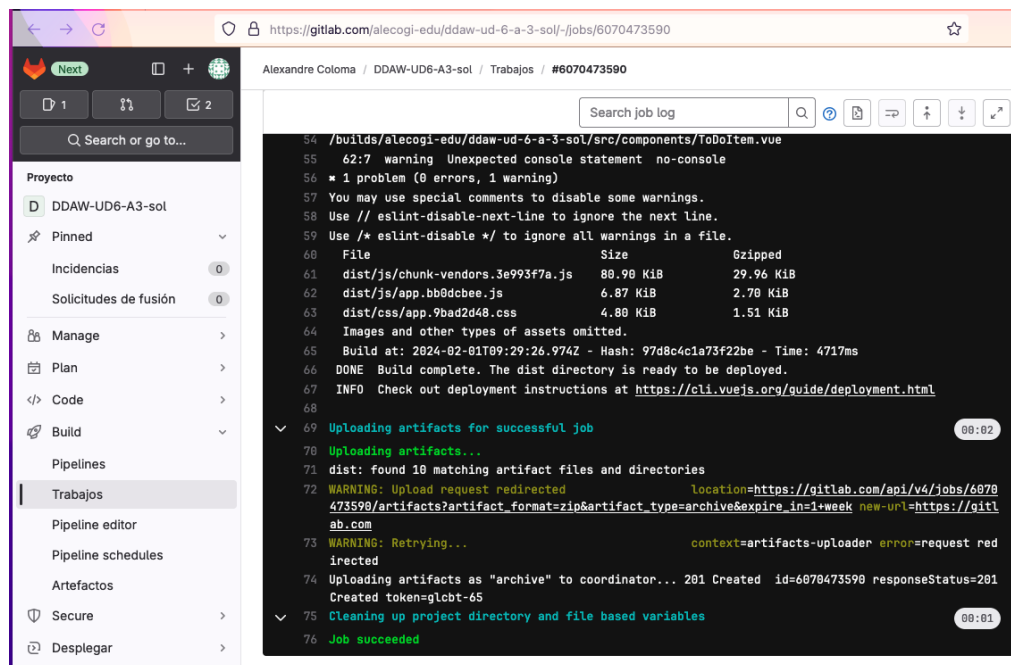
En el moment en el que es confirmen els canvis i **gitlab detecta** que hi ha un **fitxer .gitlab-ci.yml activa** el **mòdul** de CI/CD i comença a executar les tasques definides. Podem veure que està executant les tasques en la informació que ens proporciona la mateixa pàgina en la que ens trobem.

The screenshot shows the GitHub Actions interface for a repository named 'DDAW-UD6-A3-sol'. The left sidebar contains navigation links: 'Proyecto', 'D DDAW-UD6-A3-sol', 'Pinned', 'Incidentes', 'Solicitudes de fusión', and 'Manage'. The main area displays a notification 'Los cambios se han guardado correctamente.' and a list of pipeline runs. The first run, 'Pipeline #1160286744 Running for 883a4550: Actualizar el archivo .gitlab...', is highlighted with a red box. A red arrow points to this box from the right. Below the pipeline list, there are tabs for 'Editar', 'Visualizar', 'Validar', and 'Full configuration'.

Si premem sobre el símbol  accedirem al **veure** la **etapa** que acabem de configurar i els **treballs actius** que tenim en eixe moment.



Podem obtenir més informació sobre la tasca que està realitzant-se accedint a la pestanya "Trabajos" i seleccionant el treball que està executant-se.



Ara, **definirem les tasques** que volem que s'executen en la **etapa de "test"**, com que aquesta aplicació de prova no té test unitaris definits, simplement executarem un comandament que emule la seua execució. (canvia XXXYYY pel nom i cognom dels membres del grup)

gitlab-ci.yml

```
...
test-job:
  stage: test
  script:
    - echo "Hola Test Executat de XXXYYY"
    - echo " Hola Test Executat de XXXYYY"
```

Una vegada pujats els canvis al repositori, si accedim a l'opció del menú "Build → Pipelines" veurem que **tenim definida una nova etapa** anomenada **test** (abans no apareixia perquè no tenia tasques associades).



Activitat 1

- Realitza les accions esmentades i configura un pipeline amb una etapa que ens permeti compilar l'aplicació comprovant que s'executa correctament. Seguidament clona el projecte en local, introdueix un error de sintaxi al fitxer i puja un nou commit. ¿Què ha passat?

// Hauràs de pegar captures del resultat d'execució vàlid e invalid dels diferents treballs que s'han executat accedint al resum de tots els treball i a l'execució de cadascun d'ells.

2.3 Entrega Continuada

La entrega continuada ens permet desplegar l'aplicació en els servidors final. Per a fer aquest exemple hem de tenir configurat un servidor web amb una IP pública, En el nostre cas, utilitzarem un **màquina virtual** del servici **de EC2**. La configuració que hem aplicat és la següent:

- **IP pública/domini de la màquina:** example.com
- **Servici openssh** escoltan pel port 22 amb autenticació per clau **publica/privada** activada.

- **Servici nginx** instal·lat
- **Carpeta de desplegament:** `var/www/001-example.com/html/`
- **Document Root:** `var/www/001-example.com/html/dist`
- **Usuari de desplegament:** `exampleuser`

Definició de la etapa

Definirem una **nova** etapa a la que anomenarem **deploy** i que ens permetrà copiar l'executable que hem compilat al pas anterior a la màquina de producció per a que pugui ser utilitzada pels usuaris finals. Les passes que inclou aquesta etapa són:

Etapa 2

1. Crear un contenidor docker amb una imatge linux alpine.
2. Copiar la clau privada de l'usuari `exampleuser` a la màquina docker
3. Donar permisos a la clau privada per a que pugui ser utilitzada
4. Copiar l'executable al document Root de la màquina en producció

Per a continuar, editem el fitxer `.gitlab-ci.yml` i declararem la nova etapa

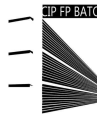
`gitlab-ci.yml`

```
stages:
  - build
  - test
  - deploy
...
```

Definició de Variables i Secrets compartits

Abans de definir les tasques, necessitem tindre accessible per la màquina de docker (encarregada de fer el desplegament) de un parell de claus **publica/privada** per a que l'**usuari** que vagi a executar les tasques des **del contenidor que llençarà gitlab** pugui **autenticar-se** amb la **màquina final** on desplegarem l'aplicació (EC2 de AWS). Seguirem el següents passos:

1. En primer lloc crearem el parell de claus:



```
$ ssh-keygen -t rsa -b 2048 -C "gitlab-cy" -f $HOME/id_rsa_gitlab
```

Una vegada executada la ordre, veurem que se'ns ha creat un nou parell de claus en directori home de l'usuari que estiguem gastant:

```
→ ~ ssh-keygen -t rsa -b 2048 -C "gitlab-cy" -f $HOME/id_rsa_gitlab
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/profeddaw/id_rsa_gitlab
Your public key has been saved in /Users/profeddaw/id_rsa_gitlab.pub
The key fingerprint is:
SHA256:HSB20Z+NfdwgegnweJEii9YvqV4fC0vEArtWfGXL0gI gitlab-cy
The key's randomart image is:
+----[RSA 2048]-----+
|      o +o..      |
|      ..o++o.. .  |
|      o o.++B + o |
|    Eo oo o=. = o .|
|. ..o =oS .. .   |
| o + *o=.         |
|. o =.=..         |
| o +.. . .        |
|o  .o .           |
+-----[SHA256]-----+
→ ~
```

2. Afegirem la clau privada com a **variable d'entorn** en gitlab

1. Anirem a la opció **Configuración > CI/CD**, expandirem la secció de **Variables** i pulsarem sobre el botó **"Add Variable"**.

Variables

Variables store information that you can use in job scripts. Each project can define a maximum of 8000 variables. [Obtener mas información.](#)

Variables can be accidentally exposed in a job log, or maliciously sent to a third party server. The masked variable feature can help reduce the risk of accidentally exposing variable values, but is not a guaranteed method to prevent malicious users from accessing variables. [How can I make my variables more secure?](#)

Variables can have several attributes. [Obtener mas información.](#)

- Protected: Only exposed to protected branches or protected tags.
- Masked: Hidden in job logs. Must match masking requirements.
- Expanded: Variables with `$` will be treated as the start of a reference to another variable.

Clave	Valor	Entornos	Acciones
Todavía no hay variables.			

2. Emplenarem els següents camps:

1. **clave:** És tracta del nom que volem donar-li, una sola línia sense espais en blanc, nosaltres ficarem **SSH_PRIVATE_KEY**
2. **value:** copiarem el contingut del fitxer **id_rsa_gitlab** creat al anteriorment.
3. Crearem la variable polsant sobre el botó "Guardar cambios".
4. A partir d'ara, ja tenim la variable **SSH_PRIVATE_KEY** disponible per a qualssevol **"treball"** que s'execute en **una rama protegida** com és la **rama main**.

Definició de les tasques

Tornarem a editar el fitxer **.gitlab-ci.yml** i afegirem la tasca **deploy_job** per a que s'execute en la **etapa deploy**, però en aquest cas, **sols** volem que ho faça quan detecte un **commit** en la **rama master**.

gitlab-ci.yml
<pre>... deploy-job: only: - main image: alpine stage: deploy script: - apk add --no-cache rsync openssh - mkdir -p ~/.ssh - echo "\$SSH_PRIVATE_KEY" >> ~/.ssh/id_dsa - chmod 600 ~/.ssh/id_dsa - echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config - rsync -rav --delete dist/ exampleuser@example.com:/var/www/001-example.com/html/dist</pre>

A continuació analitzarem algunes de les directives utilitzades:

- **only:** Indica les rames en les que volem que s'execute aquesta tasca. Hem ficat main perquè sols volem que es desplegui l'aplicació al servidor de producció quan es faça una integració en **main**.

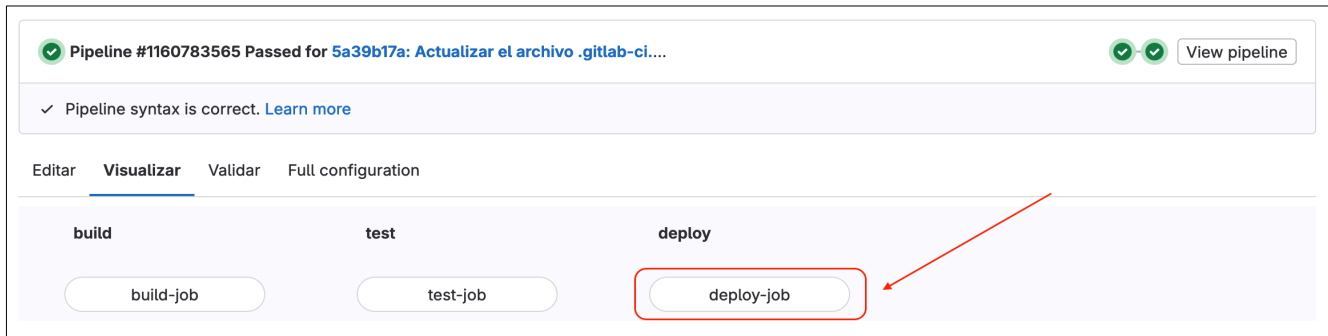
```
only:
  - main
```

- **script:** El que fem es:
 1. **apk add --no-cache rsync openssh:** Instalem l'eina **rsync** i **openssh**. Utilitzarem l'eina rsync per a copiar l'aplicació Vue compilada al servidor de producció. Aquesta eina és molt versàtil i funciona sobre el servici ssh. Pots obtindre

més informació [ací](#).

2. `echo "$SSH_PRIVATE_KEY" >> ~/.ssh/id_dsa`: Copiem la clau privada guardada en la variable `$SSH_PRIVATE_KEY` de gitlab al contenidor.

Una vegada pugem els canvis al repositori, podrem veure com s'ha afegit la nova etapa al pipeline.



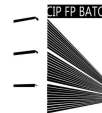
Activitat 2

- Crea una màquina en el **servici EC2** de **AWS**, i configura-la per a que pugui realitzar el desplegament automàtic del projecte tot list que estem treballant en aquesta pràctica. Has de tenir en compte que

- Has de crear una màquina en EC2, assignar-li una IP elàstica i obrir els ports corresponents mitjançant els grups de seguretat.
- Hauràs de crear un usuari de desplegament i ficar la clau pública del repositori de gitlab com a autoritzada per a connectar-se amb eixe usuari.
- Instal·la el servidor nginx i configura'l per a que pugui servir el fitxer compilat de l'aplicació.
- Afegeix el nou step en el pipeline de gitlab per a que es connecti al servidor i faci el desplegament quan hi haja un nou commit en la branca `main`.

Hauràs de pegar captures que mostren:

- El pipeline configurat.
- El desplegament exitós de l'aplicació, tant en el panell de control de gitlab com en el detall d'execució del treball.
- L'aplicació funcionant accedint des del navegador i els logs d'accés del servidor root corresponent.



3. Bibliografia / Webgrafia

- Documentació oficial de gitlab CI.
https://docs.gitlab.com/ee/topics/build_your_application.html
- Documentació de rsync. <https://linux.die.net/man/1/rsync>