¿Qué es y para que sirve Gatsby?

Gatsby permite crear sitios web estáticos, vamos a utilizarlo para crear nuestro primer proyecto y hacer el despliegue a mediante GitLab.

Un proyecto Gatsby es una serie de archivos HTML, JavaScript y CSS.

¿Qué es Docker?

Docker es una tecnología basada en contenedores para proveer virtualización. Imaginemos por ejemplo que tenemos que instalar una aplicación como nodejs pero no deseamos realizar dicha instalación dentro de nuestro sistema operativo. Para esto Docker provee imágenes que pueden ser descargadas y utilizadas inmediatamente. Al ejecutar nuestra maquina virtual podemos personalizar las herramientas y paquetes que necesitamos.

Creación de nuestra pipeline

Empezamos creando nuestro archivo .gitlab-ci.yml en donde vamos a definir nuestras stages.

```
build website:
  script:
    - npm i
    - npm i -g gatsby-cli
    - gatsby build
```

Guardamos los cambios y empujamos a GitLab.

```
git add .gitlab-ci.yml
git commit -m "Agregada la configuración de la pipeline"
```

En este momento veremos que empieza a ejecutarse el job dentro de la pipeline de nuestro proyecto en GitLab. También veremos que se ejecuta un error dentro de la ejecución.

```
Command npm not found.
```

GitLab por default utiliza una imagen de Docker para ruby que no dispone del paquete npm, pero GitLab también nos permite especificar una imagen alternativa.

Cambiamos nuestro código para utilizar la imagen de Docker llamada **node**.

```
build website:
  image: node
  script:
    - npm i
    - npm i -g gatsby-cli
```

```
    - gatsby build
```

Volvemos a empujar nuestros cambios, ahora veremos que se ejecutan todos los pasos correctamente y al final el mensaje.

Job succeeded

Para validar los archivos compilados podemos definir nuestros **artifacts** que en este caso sería todo el directorio public.

```
build website:
  image: node
  script:
    - npm i
    - npm i -g gatsby-cli
    - gatsby build
  artifacts:
    paths:
      - ./public
```

Empujamos los cambios, esperamos a que se vuelva a completar el job en la pipeline y descargamos el artifact. El zip contendrá todo el directorio public con los archivos compilados por Gatsby.

¿Por qué los jobs fallan?

Cada que un comando es ejecutado, el sistema responde con un estatus, este puede ser:

- 0, para una ejecución exitosa.
- 1 a 255, para reportar un error.

Una forma de validar la integridad de nuestro proyecto es buscar un patrón dentro de los archivos.

Dentro del proyecto, en el directorio public vamos a verificar que la palabra “Gatsby” aparezca dentro del archivo “index.html”, esto debido a que cuando se genera dicho archivo el contenido de este incluye dicha palabra.

```
cd public
grep "Gatsby" index.html
echo $?
```

0

Si por ejemplo intentamos buscar un texto que sabemos que no existe dentro de index.html, la salida seria la siguiente.

```
grep "aleluya" index.html
```

```
echo $?
1
```

El comando echo $? nos entrega el código de la última respuesta de la línea de comandos.

Ahora podemos expandir nuestro archivo .gitlab-ci.yml para que ejecute el escenario de prueba.

```
stages:
  - build
  - test

build website:
  stage: build
  image: node
  script:
    - npm i
    - npm i -g gatsby-cli
    - gatsby build
  artifacts:
    paths:
      - ./public

test artifact:
  image: node
  script:
    - grep "Gatsby" "./public/index.html"
```

Podemos confirmar que la ejecución del pipeline se realiza de forma exitosa, el artifact es compartido a través de ambas stages y en el stage de test se verifica que index.html contenga el texto “Gatsby”.

¿Cómo ejecutar tareas en paralelo?

GitLab tiene al capacidad de ejecutar varios procesos de forma simultanea. Supongamos que posterior al build deseamos realizar 2 pruebas en paralelo.

- La primera prueba es para verificar que el archivo index.html se haya escrito correctamente dentro de public.
- La segunda prueba es para verificar que el archivo manifest.webmanifest se haya escrito correctamente también dentro de public.

Para ello podemos actualizar nuestro .gitlab-ci.yml, en este caso con 2 escenarios de test.

```
stages:
  - build
  - test


build website:
  stage: build
  image: node
  script:
    - npm i
    - npm i -g gatsby-cli
    - gatsby build
  artifacts:
    paths:
      - ./public


test index:
  stage: test
  image: alpine
  script:
    - grep "Gatsby" "./public/index.html"


test manifest:
  stage: test
  image: alpine
  script:
    - grep "icons" "./public/manifest.webmanifest"
```

La imagen **alpine** es una imagen que contiene un sistema operativo Linux de forma superoptimizada. Es decir que solo requiere de 5MB para funcionar.

Al empujar los cambios a GitLab, podremos ver que tanto **test index** como **test manifest** se ejecutan de forma simultanea.

La principal ventaja de la ejecución de jobs en paralelo es que permite realizar agilizar las operaciones de despliegue de la pipeline, al realizar dichas operaciones de forma simultanea.