

DevOps: Implantación de un entorno de integración continua

Por Juan Daniel Méndez Marín

2º ASIR A

Índice

Índice	1
Visión general	2
El modelo de Integración y Entrega Continua CI/CD	2
Justificación del proyecto	4
¿Porque adoptar la metodología DevOps?	4
Introducción	5
Objetivos	8
Aumentar la rentabilidad	8
Aumentar frecuencia de publicación	8
Mejorar la calidad de software	8
Mejorar la seguridad	8
Aumentar la productividad	9
Reducción de costes	9
Desarrollo	10
Requisitos previos	10
¿Qué es docker-compose?	10
Gitlab	11
Conclusiones	15
Bibliografía	16
Anexos	17

Visión general

La idea de este proyecto surgió durante mi primera entrevista en la empresa cuando inicié la etapa de formación en centros de trabajo. El encargado de tecnología me habló sobre las tendencias y el flujo de trabajo de la empresa y las habilidades que me permitirían iniciar una carrera como un administrador de sistemas. Esa fue mi primer contacto con el concepto de DevOps.

DevOps es el acrónimo de Development & Operations (Desarrollo y Operaciones en español) es una metodología, un término que incluye conceptos, técnicas y prácticas, todas ellas enfocadas al desarrollo de software. Gracias a estos conceptos y técnicas es posible la automatización de los procesos entre los equipos de desarrolladores y los de operaciones/administradores, mejorando la interacción entre estas dos áreas, agilizando la interacción reduciendo el tiempo desde el proceso de planificación hasta la puesta en producción del producto.

La metodología DevOps tiene como pilares principales la automatización y el monitoreo de todos los pasos en la construcción de software, desde la integración, las pruebas, la liberación, hasta la implementación y la administración de la infraestructura.

El modelo de Integración y Entrega Continua CI/CD

En el desarrollo de aplicaciones modernas se lleva a cabo por múltiples desarrolladores que trabajan en múltiples funciones de la misma aplicación. Sin embargo fusionar todo el código en un único momento al final del proceso de desarrollo puede ser una tarea tediosa, por no hablar de los posibles imprevistos que puedan surgir y el tiempo que llevaría resolverlos. Esto puede suceder cuando un desarrollador trabaja de forma aislada introduce un cambio en una aplicación que entra en conflicto con aquellos cambios realizados simultáneamente por otros desarrolladores. La integración continua permite que los desarrolladores fusionen los cambios que introducen en el código, en una división compartida o rama. Una vez se lleva a cabo esta fusión los cambios se validan con la ejecución de distintas pruebas automatizadas (generalmente pruebas de unidad e integración) para verificar que dichos cambios no hayan dañado la aplicación. Si se detecta un conflicto en las pruebas la integración continua se emitirá el reporte permitiendo tener control sobre los errores y agilizando la resolución de los conflictos que se han producido.

Si la fase de la integración continua se ejecuta con éxito, la distribución continua automatiza la liberación del código validado hacia un repositorio. Siendo entonces el objetivo de la entrega continua tener una base de código que pueda implementarse en un entorno de producción en cualquier momento.

Al final de este proceso el equipo de operaciones puede implementar una aplicación para que llegue a la de producción de forma rápida y sencilla.



Justificación del proyecto

¿Porque adoptar la metodología DevOps?

- Mejorar la cooperación y el entendimiento entre los diferentes departamentos es la clave de la metodología DevOps, por eso muchas de las herramientas están diseñadas para facilitar y mejorar la comunicación, llevando al siguiente nivel la dinámica de comunicación y retroalimentación entre los equipo de sistemas y desarrollo.
- Por otra parte, se reduce el tiempo hasta llegar a la puesta en producción. Los equipos son capaces de satisfacer las necesidades de sus clientes mucho más rápido, siendo capaces de poner en producción productos software mucho más a menudo, debido a una mayor cooperación entre los equipos de desarrollo, testing y operaciones y a la implantación de metodologías de trabajo que permiten detectar errores a tiempo, siendo más fácil y menos costoso corregirlos.
- El aumento de las tasas de despliegue conlleva la creación más fluida de nuevos productos, una mayor innovación y por ende un mayor aprendizaje.
- Mejora la escalabilidad de los proyectos, el sistema DevOps aporta la flexibilidad para añadir nuevo código. Esto, permite adaptarse a las circunstancias cambiantes que requieran la ampliación del software en cualquier sentido, sin perjudicar la producción mediante la implementación de versiones paralelas que satisfagan diferentes demandas.
- Las prioridades estratégicas de las organizaciones han cambiado. Actualmente, el negocio se basa en la entrega continua de producto, algo inalcanzable sin la agilidad, la cooperación y la comunicación que implica la adopción de DevOps.

Introducción

El proyecto consiste en la implantación de un entorno de integración continua que permita a una empresa evaluar un nuevo flujo de trabajo basado en la filosofía DevOps. Se hará uso de diversas herramientas de software desplegadas en contenedores usando docker. La infraestructura tendrá los siguientes componentes:

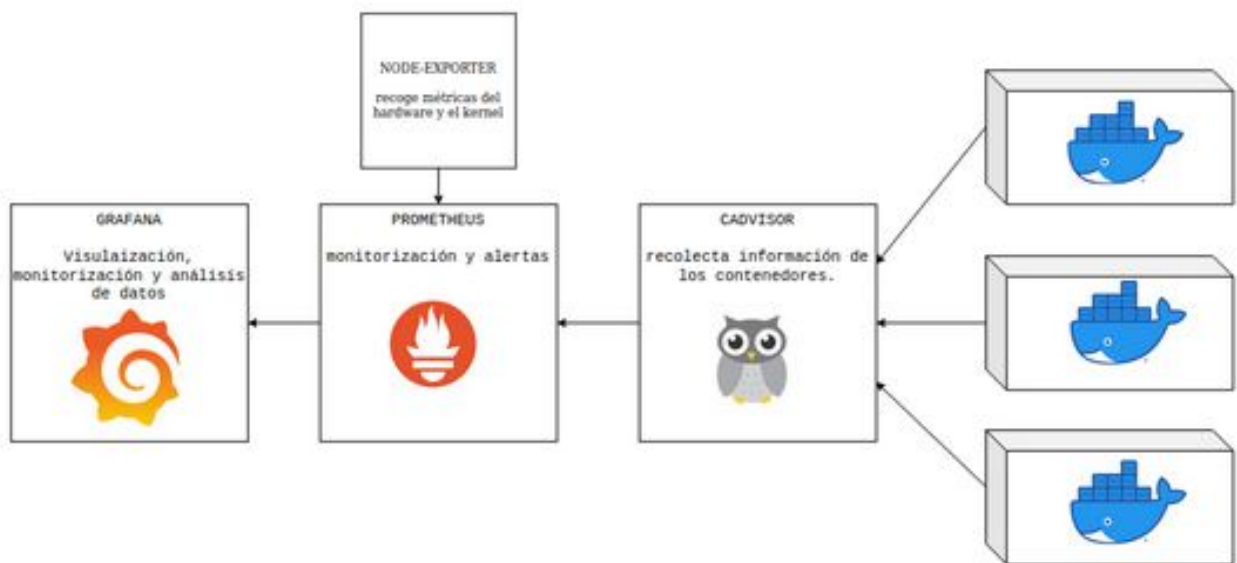
- Haproxy: Un proxy inverso que controlará el acceso a todos los servicios en los contenedores dentro del servidor mediante conexión segura HTTPS y proporcionará servicios de balanceo de carga si fuera necesario.
- Gitlab: Un servicio de control de versiones. Además de esto, se instalarán servicios adicionales como gitlab runner necesario para las tareas de integración y entrega continua. También se hará uso de Gitlab Registry un complemento que permitirá que cada proyecto en gitlab tenga su propio espacio para almacenar contenedores docker.
- Nexus: Una herramienta que puede ser configurada como un repositorio privado.
- Kubernetes: Una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Ofrece un entorno de administración centrado en contenedores y se encarga orquestación de la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo.
- Un sistema de monitorización que nos permitirá tener un mejor control de los recursos del servidor utilizando cadvisor, prometheus y Grafana.

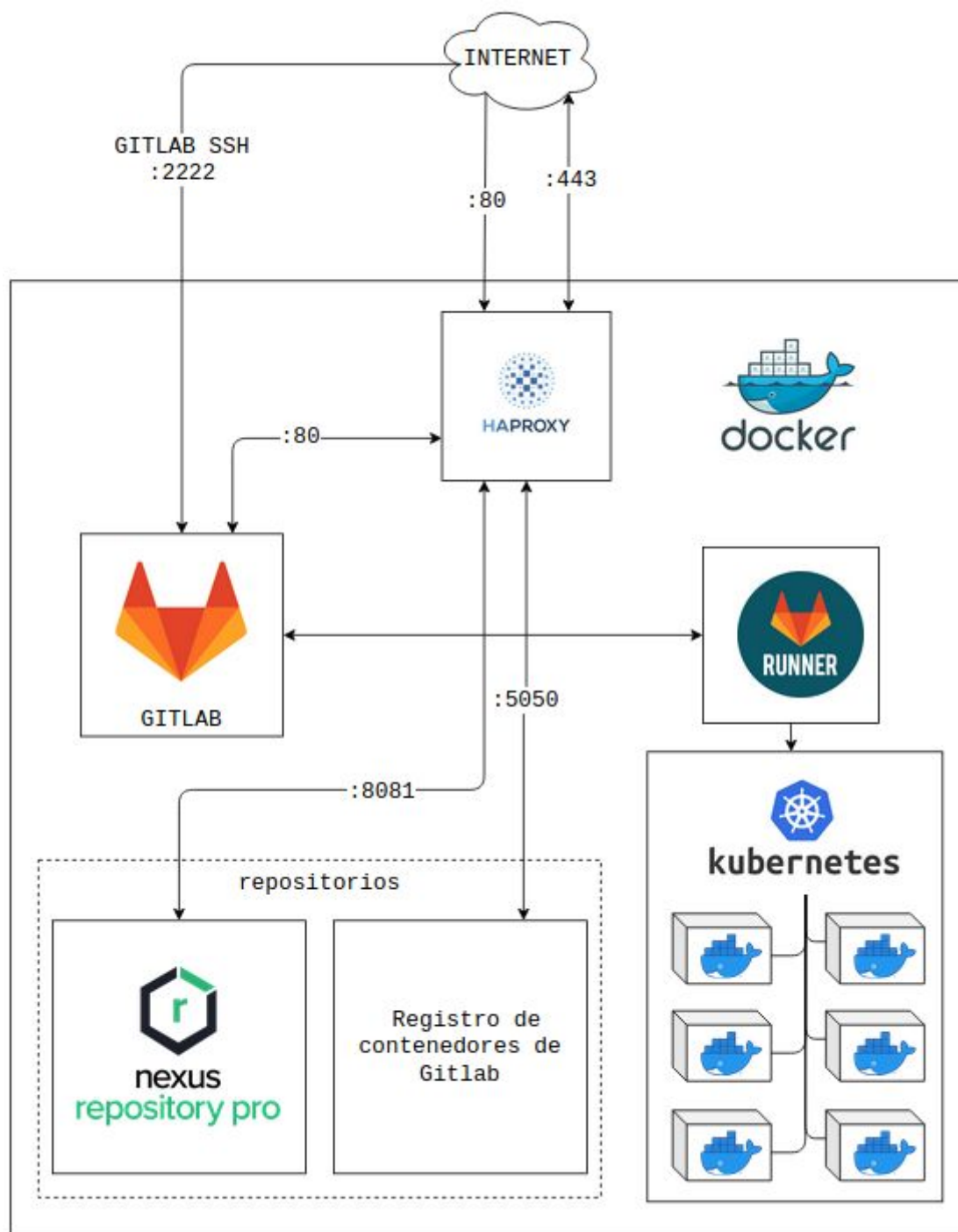
El comportamiento es el siguiente:

- Los programadores subirán su código en una nueva versión del proyecto a Gitlab.
- Gitlab alojará el código de los diferentes desarrolladores de forma centralizada, además versionará el código pudiendo recuperar una versión específica en un momento dado reduciendo el tiempo que se emplearía normalmente en deshacer los cambios. Cualquier cambio se encuentra etiquetado y es trazable, además sólo se actualizan los cambios específicos, no se reemplaza el proyecto por completo ni lo cambiados.

- Automáticamente el runner se pondrá en marcha y ejecutará las tareas incluidas en un archivo llamado **.gitlab-ci.yml** incluido en el proyecto, estas pueden ir desde la ejecución de pruebas de código hasta la construcción de contenedores.
- Los contenedores generados se almacenarán en los repositorios disponibles a la vez que se despliegan en un clúster de kubernetes vinculado a Gitlab.

sistema de monitorización de contenedores





Objetivos

1. Aumentar la rentabilidad

El número de negocios que demandan productos digitales aumenta significativamente cada año. La metodología DevOps mejora la productividad gracias a la optimización progresiva del trabajo. El modelo de integración y entrega continua reduce el tiempo de lanzamiento del producto.

2. Aumentar frecuencia de publicación

La integración continua permite a los desarrolladores añadir su código a un repositorio central en el que se compilan y se ejecutan pruebas permitiendo detectar errores desde las primeras etapas. Como consecuencia se habilita el sistema de entrega continua con el que se automatizan las tareas para entregar versiones estables, aumentando la tasa de despliegue de software de forma que los usuarios finales tienen acceso a un producto probado con mayor frecuencia.

3. Mejorar la calidad de software

La calidad dependerá tanto de que se cumplan los requisitos como del número de errores que pueda contener el producto. El proceso de integración incluye la comprobación desde las primeras etapas hasta el despliegue, evitando arrastrar errores a versiones posteriores del producto. De igual forma la fluidez de la información entre los equipos permitirá responder a las incidencias con mayor rapidez y eficacia.

4. Mejorar la seguridad

El propio sistema de desarrollo DevOps facilita la generación de operaciones seguras, la automatización de los procesos de control reduce la participación humana que no aporta valor, minimizando la posibilidad de error.

5. Aumentar la productividad

Dentro de un sistema DevOps, un trabajador puede percibir con mayor facilidad el impacto de sus aportaciones al conjunto del trabajo, sintiéndose más valorados y favoreciendo a una mayor implicación en el proceso de desarrollo. Otra ventaja es que permite liberar trabajadores de cargas que pueden resultar repetitivas, como revisar código, ya que éstas estarán automatizadas.

6. Reducción de costes

Como consecuencia directa del aumento de la eficacia de los trabajadores, la productividad y un menor tiempo de producción.

Desarrollo

Requisitos previos

Este es el software necesario para poder iniciar la instalación del entorno.

- git
- docker
- docker-compose

¿Qué es docker-compose?

Docker Compose es una herramienta que permite simplificar el uso de Docker, generando scripts que facilitan el diseño y la construcción de servicios o de aplicaciones con múltiples componentes.

docker-compose permite crear diferentes contenedores y al mismo tiempo, en cada contenedor, diferentes servicios, unirlos a un volumen común, iniciarlos y apagarlos, etc. Es un componente fundamental para poder construir aplicaciones y microservicios.

En vez de utilizar una serie de comandos bash y scripts, Compose implementa recetas similares a las de Ansible (en YAML), para poder instruir al engine a realizar tareas, programáticamente.

En resumen, docker-compose nos permitirá lanzar todos los contenedores que constituyen el entorno con único comando siguiendo las instrucciones de un archivo que por defecto se llama **docker-compose.yml**. Su estructura es la siguiente:

```

🐦 docker-compose.yml
1  #versión de docker-compose 3.8 es la más reciente
2  version: "3.8"
3
4  volumes:
5  |   #Aquí se definen los volúmenes/almacenamiento de los contenedores
6  networks:
7  |   #Aquí se definen las redes utilizadas por los contenedores
8
9  services:
10 |   #Aquí se definen los diferentes servicios/contenedores

```

Gitlab

Vamos añadir nuestro primer servicio/contenedor. Para crear un contenedor con gitlab debemos definirlo en **docker-compose.yml** de la siguiente manera:

```

services:
  #Aquí se definen los diferentes servicios/contenedores

  gitlab:
    container_name: gitlab
    restart: unless-stopped
    image: 'gitlab/gitlab-ce:latest'
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'http://localhost'
        gitlab_rails['gitlab_shell_ssh_port'] = 2222
    ports:
      - '8080:80'
      #ssh port desde el que se accede desde el exterior 'gitlab_shell_ssh_port'
      - "2222:22"
    volumes:
      - GITLAB_CONFIG:/etc/gitlab
      - GITLAB_LOGS:/var/log/gitlab
      - GITLAB_DATA:/var/opt/gitlab

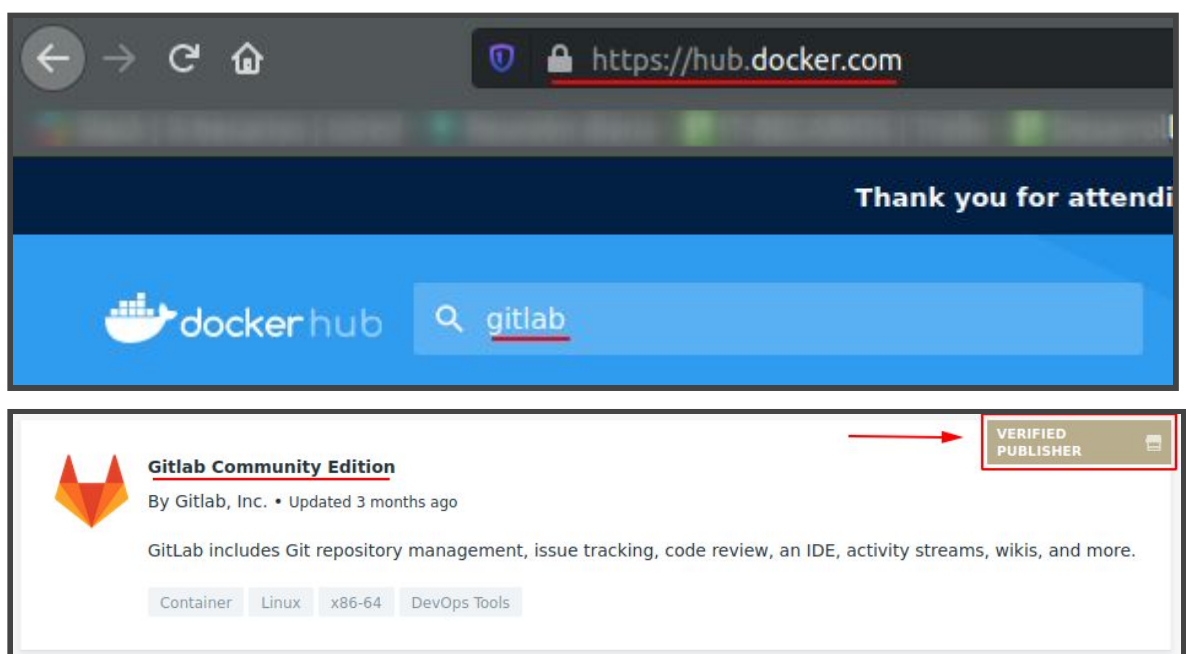
```

- **gitlab:** Es el nombre del servicio. En docker es posible conectarse de un contenedor a otro haciendo referencia al nombre del servicio siempre que estén en la misma red. Por ejemplo: podemos conectarnos al contenedor de gitlab usando la dirección **nombre_de_servicio:puerto** en este caso gitlab:80 (gitlab usa el puerto 80).
- **restart:** Aquí se define el comportamiento del contenedor al iniciar en el servidor en el que se ejecuta. la opción “unless-stopped” hace que el contenedor se inicie automáticamente al iniciar el servidor a no ser que sea

explícitamente detenido. Otras opciones disponibles son always, on-failure, etc.

- **container_name:** es el nombre que se le asigna al contenedor, esto es de gran ayuda a la hora identificar contenedores.
- **image:** Es la imagen que se utiliza para construir el contenedor, la primera vez que se ejecute el contenedor docker buscará la imagen localmente, si no la encuentra, se conectará a **docker hub** y la descargará desde allí. Las imágenes se pueden buscar en la página de docker hub:

- <https://hub.docker.com/>



Es aconsejable utilizar la imágenes de fuentes oficiales siempre que haya una disponible.



No es necesario ejecutar **docker pull nombre_de_la_imagen** docker-compose lo hará automáticamente si la imagen no se encuentra localmente.

Añadiendo el sufijo **:latest** le indicamos que descargue la última versión disponible.

- **environment:** en esta sección se incluyen variables de entorno que definen la configuración del contenedor. Son específicas de cada imagen y en algunos casos es necesario realizar investigación cuáles son, normalmente los fabricantes de software suelen tener esta información a disposición de los usuarios en la página del producto. en este caso contamos con dos variables, **external_url** define la dirección desde la que se accede al contenedor con el servicio, en este caso **localhost**. La otra es **gitlab_rails['gitlab_shell_ssh_port']** esta se indica cuando es necesario cambiar el puerto externo por el que se realizan las conexiones ssh necesarias para subir los repositorios a gitlab. En este caso asignamos el puerto 2222 porque el puerto por defecto es el 22 el cual ya está en uso por el servicio ssh de la máquina anfitrión.
- **ports:** Esta sección nos permite conectar puertos del contenedor con los del anfitrión indicados de la forma **puerto_anfitrión:puerto_contenedor**. En este caso estamos conectando el puerto 8080 del anfitrión con el 80 del contenedor, así como el puerto 2222 con el puerto 22 del contenedor creando un “túnel” para las conexiones ssh.
- **volumes:** Son las unidades de almacenamiento designadas para el contenedor en la forma **nombre_del_volumen:ruta_dentro_del_contenedor**. Estos volúmenes deben ser creados previamente a la ejecución del contenedor. Para crear volúmenes en docker se utiliza el comando:

```
docker volume create nombre_volumen
```

Esto es lo que docker llama un volumen externo, estos deben ser declarados como externos al principio del archivo de **docker-compose.yml** en la sección de **volumes** siguiendo la estructura siguiente:

```
#versión de docker-compose 3.8 es la más reciente
version: "3.8"

volumes:
  #Aquí se definen los volúmenes/almacenamiento de los contenedores
  GITLAB_CONFIG:
    external: true
  GITLAB_LOGS:
    external: true
  GITLAB_DATA:
    external: true

networks:
  #Aquí se definen las redes utilizadas por los contenedores

services:
  #Aquí se definen los diferentes servicios/contenedores

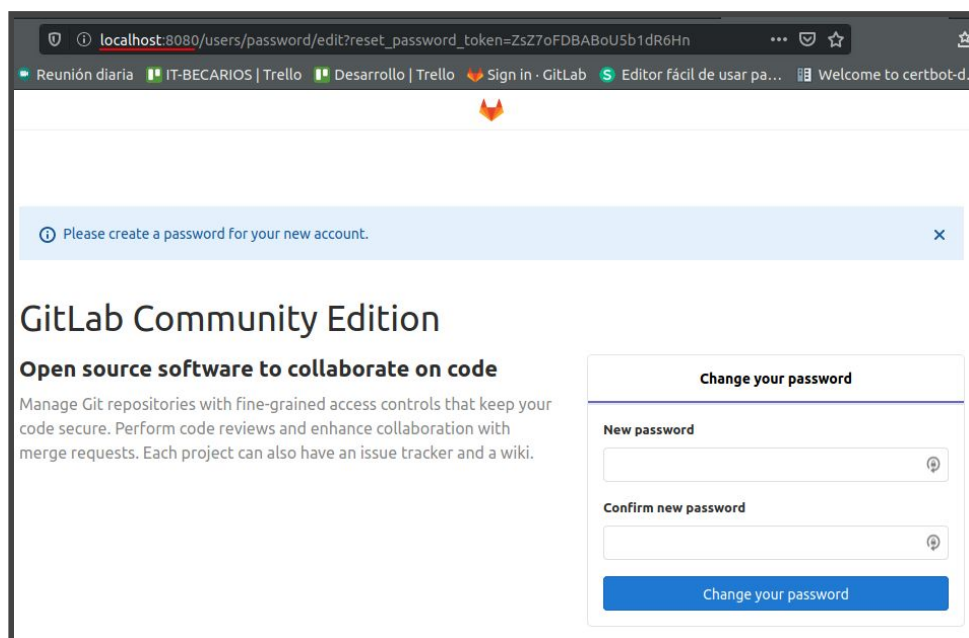
  gitlab:
```

Una vez tenemos esta configuración podemos iniciar el contenedor con:

```
docker-compose up -d
```

En la misma ruta donde está el archivo de docker-compose.yml

Tras unos minutos (gitlab tarda en iniciarse la primera vez) podemos acceder a Gitlab desde la dirección que hemos designado para el servicio localhost:8080



localhost:8080/users/password/edit?reset_password_token=ZsZ7oFDBABoU5b1dR6Hn

Reunión diaria IT-BECARIOS | Trello Desarrollo | Trello Sign in · GitLab Editor fácil de usar pa... Welcome to certbot-d.

Please create a password for your new account.

GitLab Community Edition

Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Change your password

New password

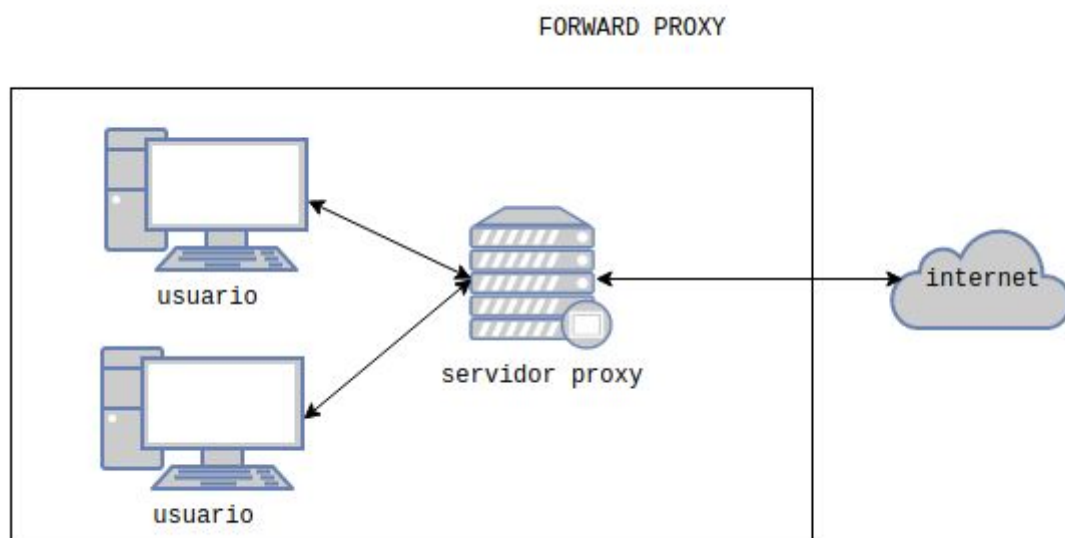
Confirm new password

Change your password

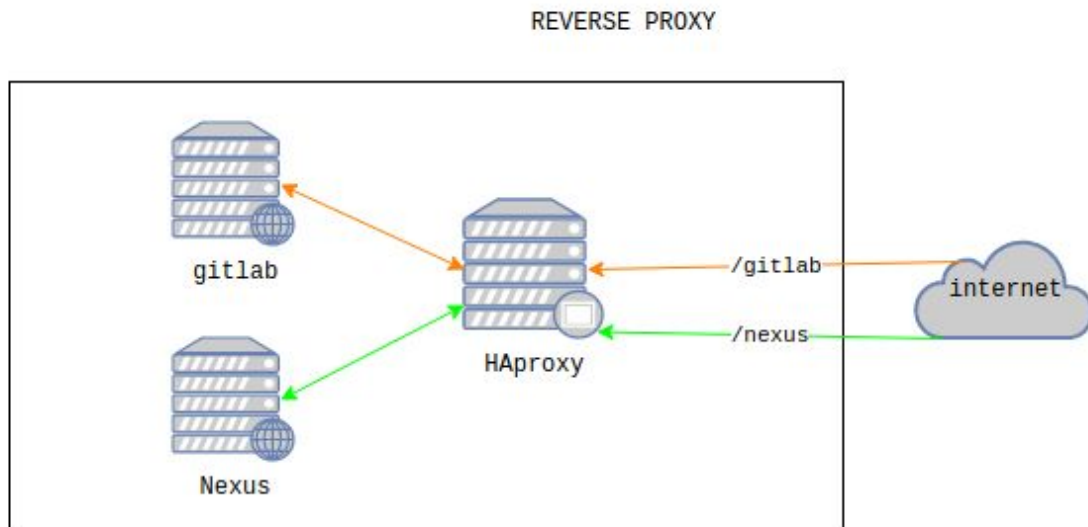
Debemos definir la contraseña para el usuario **root** con el que podremos administrar la instancia de Gitlab.

HAproxy

En términos generales un servidor proxy es una interfaz de comunicación de una red que se hace cargo de las peticiones y las transmite en calidad de representante a un ordenador de destino. En las redes corporativas se recurre a esta estructura para que los dispositivos cliente tengan un acceso controlado a internet. El servidor configurado como proxy se presenta en este caso como la única posibilidad de conexión con la red pública. Se puede hablar así de un proxy de reenvío (forward proxy).



Mientras que un proxy de reenvío protege a los dispositivos clientes en una red de las influencias de internet. Un proxy inverso por el contrario, trabaja en sentido opuesto, es decir, se ocupará de las peticiones provenientes de internet y las transmitirá a un servidor backend en segundo plano. Además de transmitir las peticiones haproxy se encargará de identificar las peticiones según su path y enviarlas al servidor/contenedor que corresponda, recibiendo el tráfico HTTP y redirigiendo a HTTPS de forma que las conexiones sean seguras utilizando la encriptación con certificado.



Para añadir el proxy debemos añadir el servicio al archivo **docker-compose.yml** :

```
services:
  haproxy:
    depends_on:
      - gitlab
    restart: unless-stopped
    image: haproxy:2.2-dev7-alpine
    container_name: haproxy
    volumes:
      - ./haproxy:/usr/local/etc/haproxy/
    ports:
      - "80:80"
      - "443:443"
```

A destacar:

- La directiva **depends_on** significa “depende de” significa que este contenedor no se iniciará hasta que se haya iniciado el/los servicios indicados en esta sección. El motivo es que gitlab y otros de los servicios que se implantaran toman más tiempo que HAProxy en iniciarse y éste dará a un error si no detecta los servidores declarados en la configuración. Por eso, a medida que el proyecto avance iremos añadiendo los demás servicios a esta sección de forma que HAProxy siempre sea el último servicio en ponerse en marcha.
- Los puertos 80 y 443 de este contenedor deben conectarse con sus homólogos en el servidor anfitrión de forma que todo el tráfico web que reciba el servidor vaya directamente a este contenedor.

- En la sección de **volumes** se ha indicado que la carpeta **haproxy** en la ruta actual se monte dentro del contenedor en la ruta **/usr/local/etc/haproxy**. Esta es la ruta en la que HAproxy busca los archivos de configuración.

Ahora debemos crear la carpeta **haproxy** dentro del directorio de trabajo (donde está el archivo `docker-compose.yml`). Esta carpeta contendrá 3 archivos:



Donde:

- **haproxy.cfg** es el archivo de configuración principal de HAproxy.
- **haproxy.pem** es un certificado que incluye la llave privada y pública, ya que es el formato que requiere haproxy. Utilizado para cifrar el tráfico mediante HTTPS.
- **dhparams** es un archivo de configuración que obtendremos de internet.

El archivo de configuración **haproxy.cfg** se divide en 4 secciones:

- **global:** Define la configuración global del servidor proxy.
- **defaults:** Define las configuraciones por defecto.
- **frontend:** Define los puntos en los que se reciben las peticiones y cómo se gestionan las mismas. Podemos crear tantos frontend como sean necesarios.
- **backend:** En esta sección se declaran los servidores internos disponibles para recibir las peticiones.

Para generar parte de la configuración utilizaremos una herramienta llamada **Mozilla SSL Configuration** disponible en este enlace:

- <https://ssl-config.mozilla.org/>

En este página seleccionamos el servicio para el que estamos generando la configuración y la versión en este caso HAProxy 2.2.

moz://a SSL Configuration Generator

Server Software

- ☐ Apache
- ☐ AWS ALB
- ☐ AWS ELB
- ☐ Caddy
- ☐ Dovecot
- ☐ Exim
- ☐ Golang
- ☒ HAProxy
- ☐ Jetty
- ☐ lighttpd

Mozilla Configuration

- ☐ Modern
 - Services with clients that support TLS 1.3 and don't need backward compatibility
- ☒ Intermediate
 - General-purpose servers with a variety of clients, recommended for almost all systems
- ☐ Old
 - Compatible with a number of very old clients, and should be used only as a last resort

Environment

Server Version 2.2

OpenSSL Version 1.1.1d

Miscellaneous

☒ HTTP Strict Transport Security
This also redirects to HTTPS, if possible

☒ OCSP Stapling

haproxy 2.2, intermediate config, OpenSSL 1.1.1d

Supports Firefox 27, Android 4.4.2, Chrome 31, Edge, IE 11 on Windows 7, Java 8u31, OpenSSL 1.0.1, Opera 20, and Safari 9

```
# generated 2020-06-07, Mozilla Guideline v5.4, HAProxy 2.2, OpenSSL 1.1.1d, intermediate configuration
# https://ssl-config.mozilla.org/#server=haproxy&version=2.2&config=intermediate&openssl=1.1.1d&guideline=5.4

global
    # intermediate configuration
    ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384
    ssl-default-bind-ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
    ssl-default-bind-options no-sslv3 no-tls10 no-tls11 no-tls-tickets

    ssl-default-server-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384
    ssl-default-server-ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
    ssl-default-server-options no-sslv3 no-tls10 no-tls11 no-tls-tickets

    # curl https://ssl-config.mozilla.org/ffdhe2048.txt > /path/to/dhparam
    ssl-dh-param-file /path/to/dhparam

frontend ft_test
    mode http
    bind :443 ssl crt /path/to/<cert+privkey+intermediate> alpn h2,http/1.1
    bind :80
    redirect scheme https code 301 if !{ ssl_fc }

# HSTS (63072000 seconds)
http-response set-header Strict-Transport-Security max-age=63072000
```

Copy

Este contenido debemos copiarlo dentro del archivo **haproxy.cfg**, pero editando las rutas marcadas en la imagen:

```
ssl-dh-param-file /usr/local/etc/haproxy/dhparams
```

Donde **/usr/local/etc/dhparams** es la ruta dentro del contenedor donde se montará el mismo archivo que creamos antes. Para obtener el contenido de este archivo ejecutamos:

```
curl https://ssl-config.mozilla.org/ffdhe2048.txt >
./haproxy/dhparams
```

El contenido del fichero **dhparams** debería ser el siguiente:

```
> dhparams
-----BEGIN DH PARAMETERS-----
MIIBCAKCAQEA////////+t+FRYortKmQ/cViAnPTzx2LnFg84tNpWp4TZBFGQz
+8yTnc4kmz75fS/jY2MMddj2gbICrsRhetPfHtXV/WVhJDP1H18GbtCFY2VVPe0a
87VXE15/V8k1mE8Mc0Dmi3fipona8+/och3xWKE2rec1MKzKT0g6eXq8CrGCsyT7
YdEIqUuyy0P7uWrat2DX9GgdT0Kj3jln9K5W7edjcrsZCweny04KbXCeAvzhzffi
7MA0BM0oNC9hkXL+n0mFg/+OTxIy7vKBg8P+0xtMb61z07X8vC7CIAXFjvGDfRaD
ssbzSibBsu/6iGtC0GEoXJf////////wIBAg==
-----END DH PARAMETERS-----
```

El siguiente paso será generar un certificado para encriptar las conexiones:

```
openssl req -newkey rsa:2048 -new -nodes -x509 -days 3650 -keyout
key.pem -out cert.pem
```

Este comando nos creará un certificado auto firmado y su llave correspondiente. Debemos copiar el contenido de ambos y copiarlo dentro del fichero **haproxy.pem**.

Vamos a actualizar la sección que define los puntos de entrada, **frontend** en el fichero de configuración:

```
frontend ft_test
  mode http
  bind :443 ssl crt /path/to/<cert+privkey+intermediate> alpn h2,http/1.1
  bind :80
  redirect scheme https code 301 if !{ ssl_fc }

  # HSTS (63072000 seconds)
  http-response set-header Strict-Transport-Security max-age=63072000
```

Esta línea en la configuración es la que se encarga de redirigir el tráfico HTTP a HTTPS:

```
redirect scheme https code 301 if !{ ssl_fc }
```

Primero cambiaremos **ft_test** esto es sólo un nombre descriptivo y podemos cambiarlo por uno que nos parezca más adecuado, como **http-https-in**. Además tenemos que actualizar la ruta en la que está nuestro certificado dentro del contenedor.

Por defecto haproxy trabaja en modo TCP por eso vamos a quitar la directiva `mode` de esta sección y ponerla en la sección **defaults** para que se aplique como un valor por defecto. Esta sección del fichero debería quedar de la siguiente manera:

```
defaults
    mode http

frontend http-https-in
    mode      http
    bind      :443 ssl crt /usr/local/etc/haproxy/haproxy.pem alpn h2,http/1.1
    bind      :80
    redirect  scheme https code 301 if !{ ssl_fc }

    # HSTS (63072000 seconds)
    http-response set-header Strict-Transport-Security max-age=63072000
```

Para que haproxy pueda identificar las peticiones según el path debemos configurar una **ACL** siguiendo esta estructura, en la sección de frontend:

```
acl nombre_acl path_beg -i /path_ejemplo
use_backend nuestro_backend if nombre_acl
```

Donde:

- **path_beg**: es una directriz que significa “comienza por”. El modificador **-i** se utiliza para que la búsqueda de la coincidencia no sea **case-sensitive**.
- **/path_ejemplo** es el path que queremos que identifique en las peticiones entrantes.
- **use_backend** es un término reservado y se utiliza para definir el backend al que se redirige la petición en caso de que se produzca una coincidencia.

Una **acl** que se encarga de identificar y reenviar las peticiones a nuestra instancia de **gitlab** tendría esta forma:

```
acl gitlab_acl path_beg -i /gitlab
use_backend gitlab if gitlab_acl
```

Sólo nos faltaría definir el backend de gitlab. Para definir un backend debemos seguir la siguiente estructura:

```
backend nombre_backend
  server nombre_server ruta_contenedor:puerto check
```

De forma que un backend para gitlab sería similar a:

```
backend gitlab
  server gitlab_server gitlab:80 check
```

Recordamos que podemos conectar contenedores haciendo referencia al nombre del servicio siempre que estén en la misma red. Ya que ambos se ejecutan el mismo archivo docker-compose.yml estos se ponen en una misma red por defecto a no ser que se indique una configuración diferente.

Antes de poner en marcha el proyecto debemos actualizar la configuración de gitlab para que el servicio se sirva en una dirección y protocolo diferentes a los que se habían definido anteriormente:

```
gitlab:
  container_name: gitlab
  restart: unless-stopped
  image: 'gitlab/gitlab-ce:latest'
  environment:
    GITLAB_OMNIBUS_CONFIG: |
      external_url 'http://localhost/gitlab'
      gitlab_rails['gitlab_shell_ssh_port'] = 2222
  ports:
    # - '8080:80'
    #ssh port desde el que se accede desde el exterior 'gitlab_shell_ssh_port'
    - "2222:22"
```

Hemos cambiado la directiva **external_url** de forma que ahora gitlab es accesible a través de la dirección **localhost/gitlab** en lugar de localhost:8080, por lo que ya no es necesario conectar el puerto 8080 del anfitrión con ningún puerto del contenedor. Ahora todas las conexiones entrantes al servidor se gestionan por HAProxy en el puerto 443, siendo este el encargado de comunicarse con los contenedores.

Detenemos el servicio con en caso de que esté en marcha:

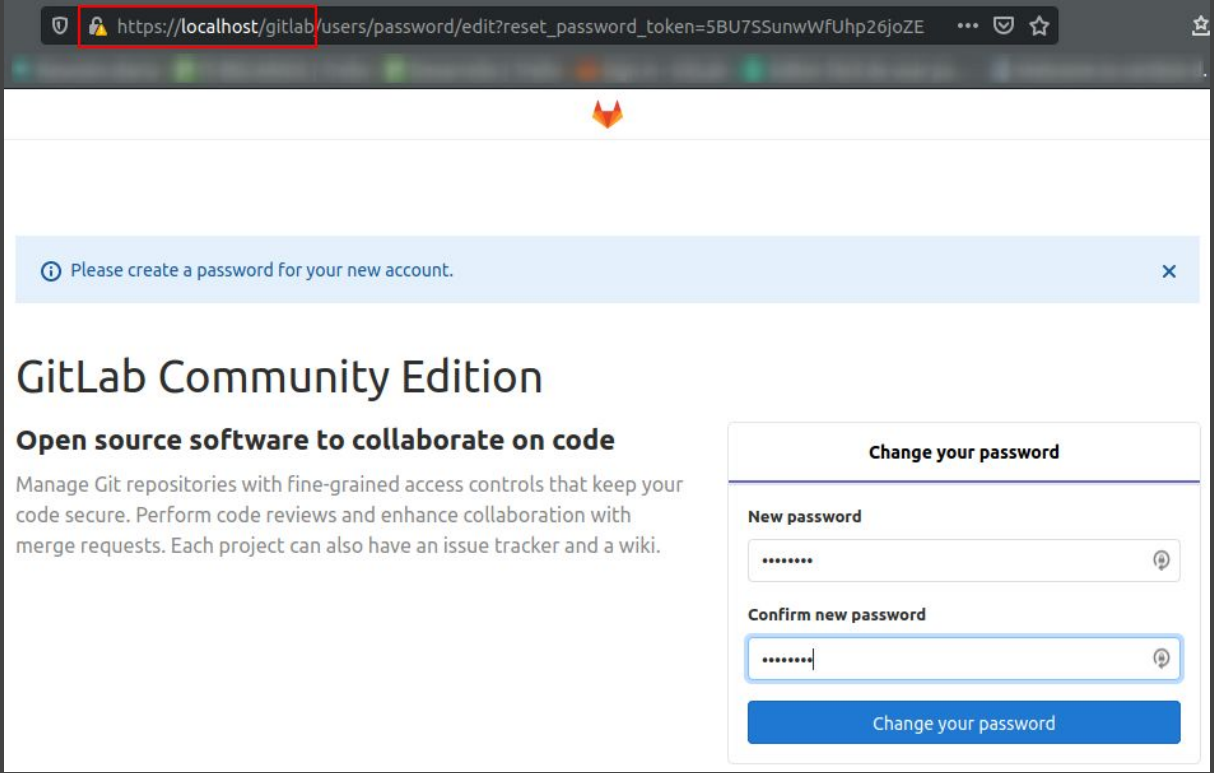
```
docker-compose down
```

Y ejecutamos:

```
docker-compose up
```


para que los servicios se vuelvan a iniciar con la nueva configuración de **docker-compose.yml**

Ahora deberíamos poder conectarnos a Gitlab desde el navegador a través de la dirección **localhost/gitlab** :



The screenshot shows a web browser window with the address bar displaying `https://localhost/gitlab/users/password/edit?reset_password_token=5BU7SSunwWfUhp26joZE`. The page features the GitLab logo at the top center. Below it, a light blue notification bar states: "Please create a password for your new account." The main heading is "GitLab Community Edition" followed by the tagline "Open source software to collaborate on code". A descriptive paragraph mentions managing Git repositories with fine-grained access controls. On the right side, there is a "Change your password" form with two input fields: "New password" and "Confirm new password", both containing masked text (dots). A blue button labeled "Change your password" is positioned at the bottom of the form.

Conclusiones

Bibliografía

- ¿Qué es DevOps? <https://www.youtube.com/watch?v=vNmWODug1fE&t=760s>
- Definición de DevOps: <https://es.wikipedia.org/wiki/DevOps>
- ¿Qué es kubernetes?
<https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>
- ¿Qué es DevOps y por qué es importante?
<https://www.chakray.com/es/devops-que-es-y-por-que-es-tan-importante/>
- Entendiendo DevOps:
<https://www.autentia.com/2018/08/17/entendiendo-devops-en-5-minutos/>
- Integración & entrega continua:
<https://www.redhat.com/es/topics/devops/what-is-ci-cd>
- ¿Por qué adoptar DevOps?
<https://www.excentia.es/Beneficios-de-adoptar-devops>
- Instalación de git en Ubuntu 18.04 LTS:
<https://www.digitalocean.com/community/tutorials/como-instalar-git-en-ubuntu-18-04-es>
- Instalar Docker ubuntu 18.04 LTS:
<https://docs.docker.com/engine/install/ubuntu/>
- Instalar docker-compose en Ubuntu 18.04 LTS:
<https://docs.docker.com/compose/install/>
- Instalar Gitlab usando docker: <https://docs.gitlab.com/omnibus/docker/>
- Cómo utilizar docker-compose: <https://docs.docker.com/compose/>
- ¿Qué es un servidor proxy inverso?
<https://www.ionos.es/digitalguide/servidores/know-how/que-es-un-servidor-proxy-inverso/>
- Las 4 secciones esenciales en la configuración de haproxy:
<https://www.haproxy.com/blog/the-four-essential-sections-of-an-haproxy-configuration/>
- Generar certificado ssl:
<https://support.microfocus.com/kb/doc.php?id=7013103>
- Docker-networking: <https://docs.docker.com/network/>
-

Anexos