

Mega-despliegue de GitLab con Integración Continua .



Realizado por Misael Noda .

- En base a lo estipulado en el [Pre-proyecto](#) ligado a este Documento . Aqui se mostrara la Documentación llevada a cabo para realizar el escenario .
- Para a llevar a cabo la correcta configuración de este proyecto y simular un entorno de “Producción” , se necesitará aparte un servidor LDAP , POSTFIX , y DNS que no aparecerán en esta documentación . Pero si lo necesario para configurar gitlab con estos .

Descripción

GitLab es un software de desarrollo colaborativo y control de versiones que se publicó bajo una licencia completamente libre (MIT) . Actualmente es usado por entidades como la NASA , el CERN , IBM o Sony .

Gitlab tiene dos versiones , la EE (Enterprise Edition) , publicada mediante una licencia propietaria , con propiedades que no tiene la version CE (community edition) .

La versión CE es una versión de software libre , apoyada por la Cloud Computing Native Foundation .

GitLab está considerado una de las herramientas más potentes en cuestión de integración continua [¿por que?](#)

Videos interesantes :

[flujo de trabajo DevOps](#)

Objetivos y valoraciones

Como se especificó en el pre-proyecto , he conseguido cumplir con los objetivos principales acordados y algunos objetivos secundarios propuestos.

Me hubiera gustado mucho más profundizar en el tema del despliegue continuo y su integración con distintos modos de metodología ágil como su integración con kubernetes, su funcionalidad “Reviews Apps” o su función en beta “Auto Devops” . Pero por cuestiones de profundidad y tiempo no he podido realizarlo.

Mi valoración final en este proyecto es que me ha gustado mucho la tecnología que he llevado a cabo y me ha servido para poder entender cosas que antes no asimilaba por completo, así mismo , gitlab es un buen software de control de versiones que cumple a la perfección con lo que necesites y es escalable perfectamente . Lo único malo era su escasa comunidad de desarrolladores , pero , con las últimas novedades del momento , eso se va acabando .

Instalacion de GitLab :

- En esta sección se detallaran los pasos para la instalación de este software en la **versión 10.5.4 para debian/stretch**

Existen diversos métodos de instalación , En esta documentación se llevará a cabo la instalación de un GitLab CE mediante la instalación automática de “Omnibus” .

Omnibus es un metodo de instalacion recomendado **fuertemente** por la documentación de GitLab ya que este método te permite actualizar luego muy fácilmente , aplica funcionalidades que no están en otros métodos y es más fácil . Un ejemplo de esto es por ejemplo que usa Runit para reiniciar un servicio por si crashea , por ejemplo con el servicio Sidekiq * que en instancias pesadas puede acabar usando mas memoria de la que debería

*Sidekiq es un proceso en segundo plano que GitLab usa para correr tareas asíncronamente .

Otros métodos de instalación (recomiendo el método de instalación manual desde el código fuente solo para valientes) :

- [Manualmente](#)
- [contenedores](#)
- [Kubernetes](#)
- [IaaS como AWS, Openshift , etc....](#)

Elegimos nuestro método y el paquete para nuestra distribución y comenzamos la instalación (en este caso , omnibus) :

```
wget --content-disposition
https://packages.gitlab.com/gitlab/gitlab-ce/packages/debian/stre
tch/gitlab-ce_10.5.4-ce.0_amd64.deb/download.deb
```

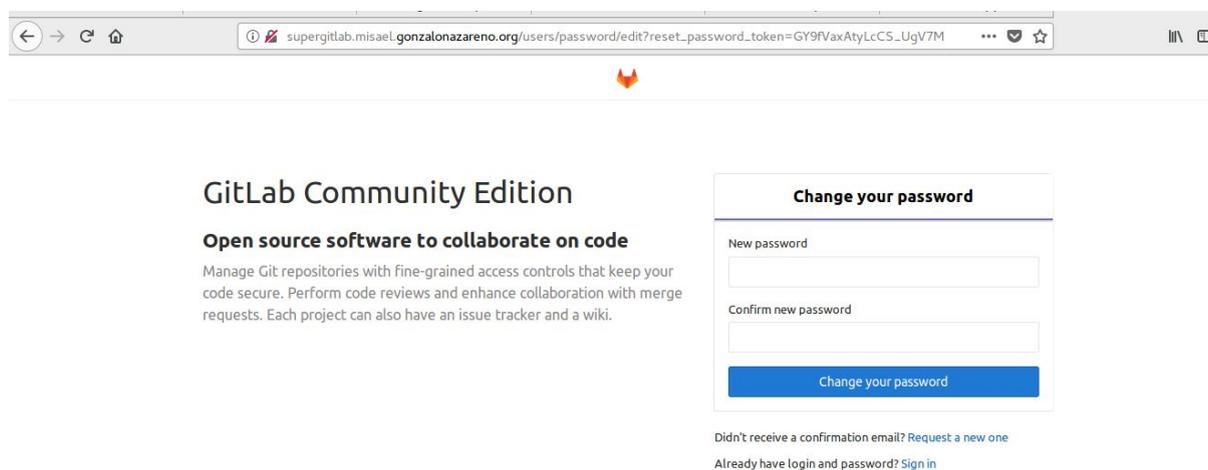
Instalamos las dependencias necesarias :

```
openssh-server
```


Una vez hecho esto ejecutamos el comando , gitlab-ctl , que controla el funcionamiento de nuestro gitlab

```
gitlab-ctl reconfigure
```

Tras esto , ya podremos acceder a la url descrita anteriormente :



The screenshot shows a web browser window with the address bar containing the URL: `supergitlab.misael.gonzalonazareno.org/users/password/edit?reset_password_token=GY9fVaxAtyLcCS_UgV7M`. The page content includes the GitLab logo and the text "GitLab Community Edition" and "Open source software to collaborate on code". Below this is a "Change your password" form with two input fields: "New password" and "Confirm new password", and a blue "Change your password" button. At the bottom of the form, there are two links: "Didn't receive a confirmation email? Request a new one" and "Already have login and password? Sign in".

Creamos una contraseña para el administrador llamado "root"

HTTPS :

Ahora vamos a configurar https en el servidor para acceder :

Creamos una entidad certificadora y la auto firmamos :

```
openssl genrsa -out gitlabCA.key 2048
```

```
openssl req -x509 -new -nodes -key gitlabCA.key -days 1024 -out  
gitlabCA.pem
```

Ahora creamos en nuestra instancia Gitlab una clave y enviamos una solicitud de firma de certificado

```
openssl genrsa -out supergitlab.misael.gonzalonazareno.key 2048

openssl req -new -key supergitlab.misael.gonzalonazareno.key -out
supergitlab.misael.gonzalonazareno.csr
```

*Nginx buscará automáticamente los ficheros .key y .csr con el nombre que hayamos puesto de url en /etc/gitlab/ssl .

Y firmamos esta solicitud

```
openssl x509 -req -days 360 -in
supergitlab.misael.gonzalonazareno.csr -CA gitlabCA.pem -CAkey
gitlabCA.key -CAcreateserial -out
supergitlab.misael.gonzalonazareno.crt
```

Ponemos los ficheros supergithub.misael.gonzalonazareno.org en el directorio /etc/gitlab/ssl y descargamos el certificado "gitlabCA.pem" en nuestro navegador .

en el fichero de configuración de gitlab , cambiamos el external_url a https y descomentamos algunas líneas

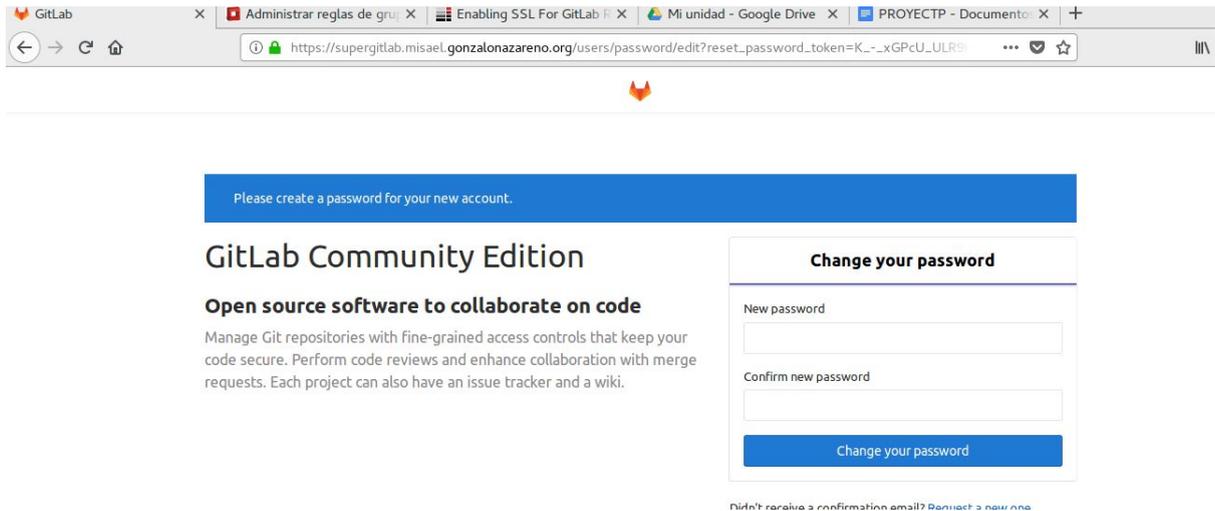
```
external_url 'https://supergitlab.misael.gonzalonazareno.org'
nginx['redirect_http_to_https'] = true
nginx['redirect_http_to_https_port'] = 80

nginx['ssl_certificate'] =
"/etc/gitlab/ssl/supergitlab.misael.gonzalonazareno.crt"
nginx['ssl_certificate_key'] =
"/etc/gitlab/ssl/supergitlab.misael.gonzalonazareno.key"
```

y reconfiguramos el servicio otra vez :

```
gitlab-ctl reconfigure
```

Con esto , habremos configurado exitosamente http y las peticiones se dirigirán a https



Notificaciones de e-mail :

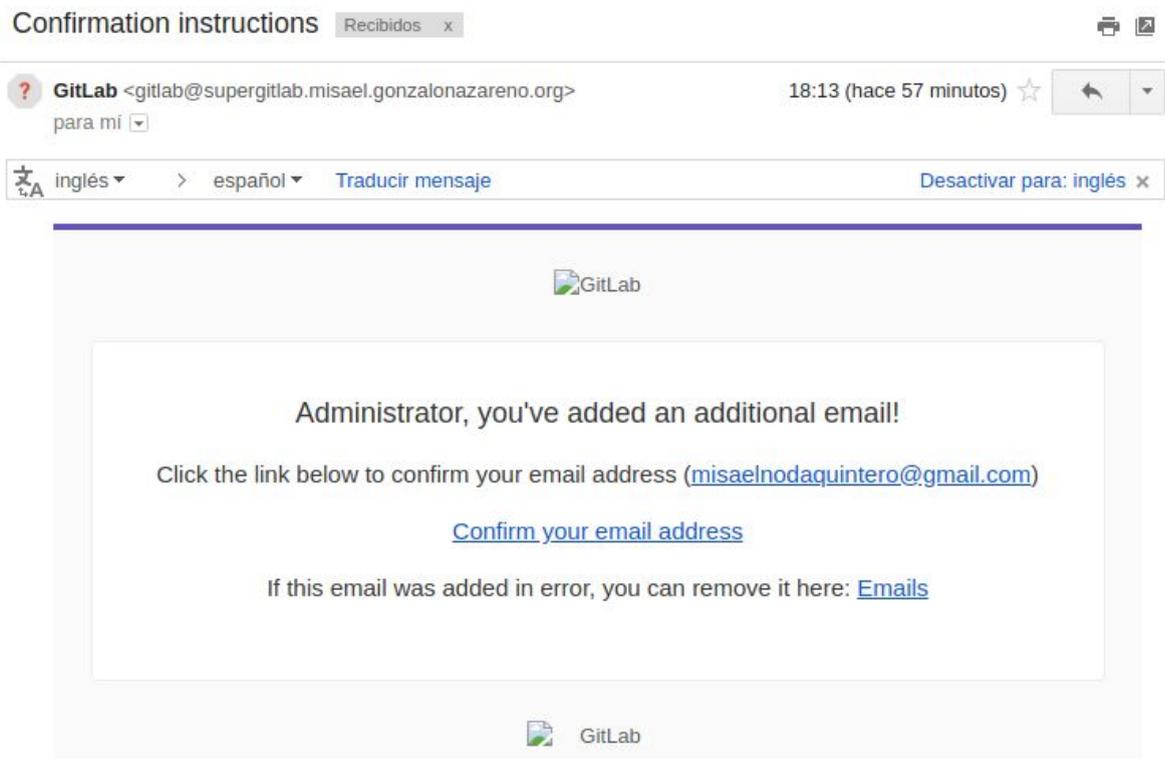
Ahora configuraremos nuestra instancia de gitlab para que envíe notificaciones a través de nuestro servidor de correos a los administradores .

En esta página existen ejemplos por si tenemos distintas configuraciones respecto a servidor de correo electrónico : [Opciones de Correo](#)

En mi caso , usaremos un servidor smtp simple que usara de relay a babuino, para configurarlo , realizamos la siguiente configuración en /etc/gitlab

```
gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "localhost"
gitlab_rails['smtp_port'] = 25
gitlab_rails['smtp_authentication'] = false
gitlab_rails['smtp_enable_starttls_auto'] = false
gitlab_rails['smtp_tls'] = false
```

Tras reiniciar la configuración y enviar un e-mail de confirmación para cambiar el correo del administrador , gitlab nos enviará un correo en la dirección indicada



Creación de usuarios

Para la creación de usuarios , el administrador puede crear los usuarios uno a uno , pero , en un entorno de empresa bastante grande que es para lo que está adecuado este proyecto , no es viable . Por eso , vamos a configurar usuarios mediante el protocolo LDAP .

Existen varios software que utilizan el protocolo LDAP , y gitlab ofrece integración con bastante de ellos, como son :

- [Oracle Internet Directory](#)
- [OpenLDAP](#)
- [389 Directory](#)
- [OpenDJ](#)
- [ApacheDS](#)

En esta documentación se incluirá documentación referida para incluir implementación junto con OpenLDAP (aunque la documentación de la web ponga de ejemplo el Active Directory de windows)

Abrimos el fichero de configuración /etc/gitlab/gitlab.rb y escribimos lo siguiente (habra seguramente una parte comentada de ejemplo)

```
gitlab_rails['ldap_enabled'] = true
gitlab_rails['ldap_servers'] = {
  'main' => {
    'host' => 'alberto.misael.gonzalonazareno.org',
    'uid' => 'cn',
    'port' => 389,
    'encryption' => 'plain',
    'verify_certificates' => false,
    'bind_dn' => 'cn=admin,dc=misael,dc=gonzalonazareno,dc=org',
    'password' => 'root',
    'active_directory' => false,
    'base' => 'ou=People,dc=misael,dc=gonzalonazareno,dc=org',
  }
}
```

Tras esto , guardamos el fichero y reiniciamos la configuracion y ejecutamos el siguiente comando .

```
gitlab-ctl reconfigure  
service gitlab-runsvdir restart
```

****superimportante**** : “service gitlab-runsvdir restart” este comando no viene en la documentación por ningún lado , y , por lo menos para mi , ha sido necesario cada vez que incluía funcionalidades nuevas en gitlab que afectaban a la interfaz web

Para asegurarnos de que los hemos introducido correctamente , podemos ejecutar una búsqueda de algún usuario desde el servidor gitlab hacia el servidor ldap situado en otra maquina

```
ldapsearch -x -h 10.0.0.8 -b  
"dc=misael,dc=gonzalonazareno,dc=org" -D  
"cn=misael,ou=People,dc=misael,dc=gonzalonazareno,dc=org" -W  
Enter LDAP Password:  
  
# extended LDIF  
#  
# LDAPv3  
# base <dc=misael,dc=gonzalonazareno,dc=org> with scope subtree  
# filter: (objectclass=*)  
# requesting: ALL  
#  
# misael.gonzalonazareno.org  
dn: dc=misael,dc=gonzalonazareno,dc=org  
objectClass: top  
objectClass: dcObject  
objectClass: organization  
o: Supergitlab  
dc: misael  
  
# admin, misael.gonzalonazareno.org  
dn: cn=admin,dc=misael,dc=gonzalonazareno,dc=org  
objectClass: simpleSecurityObject  
objectClass: organizationalRole
```

```
cn: admin
description: LDAP administrator

# People, misael.gonzalonazareno.org
dn: ou=People,dc=misael,dc=gonzalonazareno,dc=org
ou: People
objectClass: top
objectClass: organizationalUnit

# misael, People, misael.gonzalonazareno.org
dn: cn=misael,ou=People,dc=misael,dc=gonzalonazareno,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
sn: Noda
givenName: misa
cn: misael
userPassword:: bWlzYWVs
mail: fifty34@hotmail.es
title: puto amo

# search result
search: 2
result: 0 Success

# numResponses: 5
# numEntries: 4
```

Ahora esperamos , y accedemos a nuestra URL de gitlab :

LDAP Standard Register

LDAP Username

Password

Remember me

Sign in

Didn't receive a confirmation email? [Request a new one.](#)

como vemos , ha aparecido una pestaña mas para iniciar sesión a traves de ldap

Ahora , iniciaremos sesion con el usuario misael :

The screenshot shows the GitLab user profile for 'misael' (@fifty34). The profile includes a profile picture, the name 'misael', and the handle '@fifty34' with the note 'Member since April 25, 2018'. Below the profile information are tabs for 'Activity', 'Groups', 'Contributed projects', 'Personal projects', and 'Snippets'. The 'Activity' tab is selected, displaying a calendar grid for the months of April, May, June, July, August, September, October, November, December, January, February, March, and April. The calendar shows activity for the first three days of each month (Monday, Wednesday, Friday). A legend at the bottom indicates that the activity is a 'Summary of issues, merge requests, push events, and comments (Timezone: UTC)'. The legend shows three colored squares: light blue, medium blue, and dark blue.

Como vemos , se ha conseguido loguear con éxito . Solo quedaría crear un proyecto y empezar a picar código .

Docker Registry

Docker registry es un servicio de gitlab incluido en la versión 8.1 en la que se permiten incluir imágenes de docker a tu repositorio gitlab .Esto nos puede ser útil , por ejemplo , si queremos usar integración continua en nuestro proyecto con el ejecutor docker .

Existen dos formas de habilitarlo :

1. En nuestro actual dominio , utilizando otro puerto , así podemos reutilizar los certificados https , cosa que nos ahorraría unos costos
2. Bajo un dominio diferente

En esta documentación he elegido la primera opción :

Primero , descargamos docker en nuestra máquina (en esta documentación esa parte no se verá)

Luego , abrimos el fichero de configuración y escribimos las siguientes líneas

```
registry_external_url
'https://supergitlab.misael.gonzalonazareno.org:4567'

registry_nginx['ssl_certificate'] =
"/etc/gitlab/ssl/supergitlab.misael.gonzalonazareno.crt"
registry_nginx['ssl_certificate_key']="/etc/gitlab/ssl/supergitlab.misael.gonzalonazareno.key"

registry_nginx['enable'] = true
```

Ahora ejecutamos gitlab-ctl reconfigure y tenemos listo nuestro docker registry , esto nos ayudará más adelante en la implementación de integración continua .

Para usar nuestro docker registry :

Colocamos nuestros certificados en /usr/local/share/ca-certificates

ejecutamos :

```
update-ca-certificates
/etc/init.d/docker restart

docker login supergitlab.misael.gonzalonazareno.org
```

tras meter el usuario y la contraseña , ya podremos usar nuestro repositorio de imágenes docker .

```
root@josedomingo:/usr/local/share/ca-certificates/supergitlab# docker login supergitlab.misael.gonzalonazareno.org
Username: misael
Password:
Login Succeeded
root@josedomingo:/usr/local/share/ca-certificates/supergitlab# █
```

Ahora construimos una imagen y la subimos

```
docker build -t
supergitlab.misael.gonzalonazareno.org/root/superproyecto .

docker push
supergitlab.misael.gonzalonazareno.org/root/superproyecto
```

con esto , habremos subido una imagen a nuestro repositorio



Integración Continua :

Se puede habilitar la integración continua en gitlab de distintas maneras .

Auto DevOps

Auto Devops es una herramienta de integración continua en fase beta que controla todo el flujo de trabajo desde la comprobación hasta el despliegue y la monitorización de nuestro código y proyectos sin tocar nada , solo haciendo un push.

Solo necesita la conexión a un cluster de kubernetes y la configuración de las funcionalidades que le vemos.

En esta guia no la veremos ya que recomiendo esta opción cuando se quieren habilitar todo el flujo de trabajo ya que es complicado levantarlo y sus ventajas son mejores con todo el flujo activado.

Gitlab-ci.yml :

Si añadimos un fichero con extensión “.gitlab-ci.yml” en la raíz de nuestro directorio , cuando activemos la integración continua , con cada push o commit que hagamos en nuestro proyecto , gitlab desplegará una “pipeline” (conjunto de trabajos) que contendrá el resultado de la build .

Pero , necesitaremos instalar un software aparte , que se encargará de ejecutar los trabajos puestos en el fichero yml , este software se llama “[gitlab Runner](#)”

Así que , en resumen , para habilitar la integración continua mediante “[Gitlab CI/CD](#)” necesitaremos :

1. fichero .gitlab-ci.yml con las características necesarias
2. Gitlab runner , instalado y configurado

En este guia lo realizaremos todo , paso a paso :

creamos un fichero con esta extensión y lo rellenamos , en la página de la documentación viene especificado un ejemplo de prueba que levanta una aplicación ruby en proyecto rails

```
before_script:
  - apt-get update -qq && apt-get install -y -qq sqlite3
  libsqlite3-dev nodejs
  - ruby -v
  - which ruby
  - gem install bundler --no-ri --no-rdoc
  - bundle install --jobs $(nproc) "${FLAGS[@]}"

supertrabajo1:
  script:
    - bundle exec supertrabajo1

supertrabajo2:
  script:
    - bundle exec supertrabajo2
```

Aquí hemos definido dos trabajos , “supertrabajo1” y “supertrabajo2” , la zona de “before_script” se ejecutará después de cada trabajo . Cada trabajo tiene que tener obligatoriamente la keyword “script”

Lo subimos a nuestro repositorio de gitlab , recordad añadid una clave ssh si no la habéis añadido antes y queréis hacerlo mediante línea de comandos.

** Existe una herramienta en nuestra pestaña llamada “CI Lin” que permite , introduciendo el texto del fichero .gitlab-ci.yml . Si comprobamos el ejemplo :

Parameter	Value
Test Job - supertrabajo1	<pre>apt-get update -qq && apt-get install -y -qq sqlite3 libsqlite3-dev nodejs ruby -v which ruby gem install bundler --no-ri --no-rdoc bundle install --jobs \$(nproc) "\${FLAGS[@]}" bundle exec supertrabajo1</pre> <p>Tag list: Only policy: Except policy: Environment: When: on_success</p>
Test Job - supertrabajo2	<pre>apt-get update -qq && apt-get install -y -qq sqlite3 libsqlite3-dev nodejs ruby -v which ruby gem install bundler --no-ri --no-rdoc bundle install --jobs \$(nproc) "\${FLAGS[@]}" bundle exec supertrabajo2</pre> <p>Tag list: Only policy: Except policy: Environment: When: on_success</p>

Ahora , para que se ejecuten estos trabajos , tendremos que configurar e instalar gitlab runner en otra maquina .

Gitlab Runner :

se recomienda instalar gitlab runner en otra maquina , conectandolo (registrandolo) con la instancia de gitlab.

******Como en este caso , usaremos para las pruebas de integración continua docker executor, con lo cual , se tiene que instalar docker.

Podemos instalar el runner fácilmente añadiendo el repositorio

```
curl -L
https://packages.gitlab.com/install/repositories/runner/gitlab-ru
nner/script.deb.sh | sudo bash
```

a partir de debian stretch la prioridad del gitlab runner de los repositorios oficiales es más alta , con lo cual, si queremos usar uno más novedoso , tendremos que cambiar las preferencias

```
cat > /etc/apt/preferences.d/pin-gitlab-runner.pref <<EOF
Explanation: Prefiero el paquete del repositorio de los chicos de
gitlab
Package: gitlab-runner
Pin: origin packages.gitlab.com
Pin-Priority: 1001
EOF
```

Instalamos el runner propiamente dicho

```
apt-get install gitlab-runner
```

Ahora , configuraremos el gitlab runner en nuestro dominio :

Nos vamos a nuestra cuenta de gitlab , y en **Settings > CI/CD** obtenemos el token necesario que se ha creado automáticamente

Enable Auto DevOps
The Auto DevOps pipeline configuration will be used when there is no `.gitlab-ci.yml` in the project.

Disable Auto DevOps
An explicit `.gitlab-ci.yml` needs to be specified before you can begin using Continuous Integration and Delivery.

Instance default (disabled)
Follow the instance default to either have Auto DevOps enabled or disabled when there is no project specific `.gitlab-ci.yml`.

You need to specify a domain if you want to use Auto Review Apps and Auto Deploy stages.

Runner token

The secure token used by the Runner to checkout the project

[Reveal value](#)

Lo copiamos y accedemos por terminal a la máquina que tiene instalado gitlab-runner

```
gitlab-runner register
```

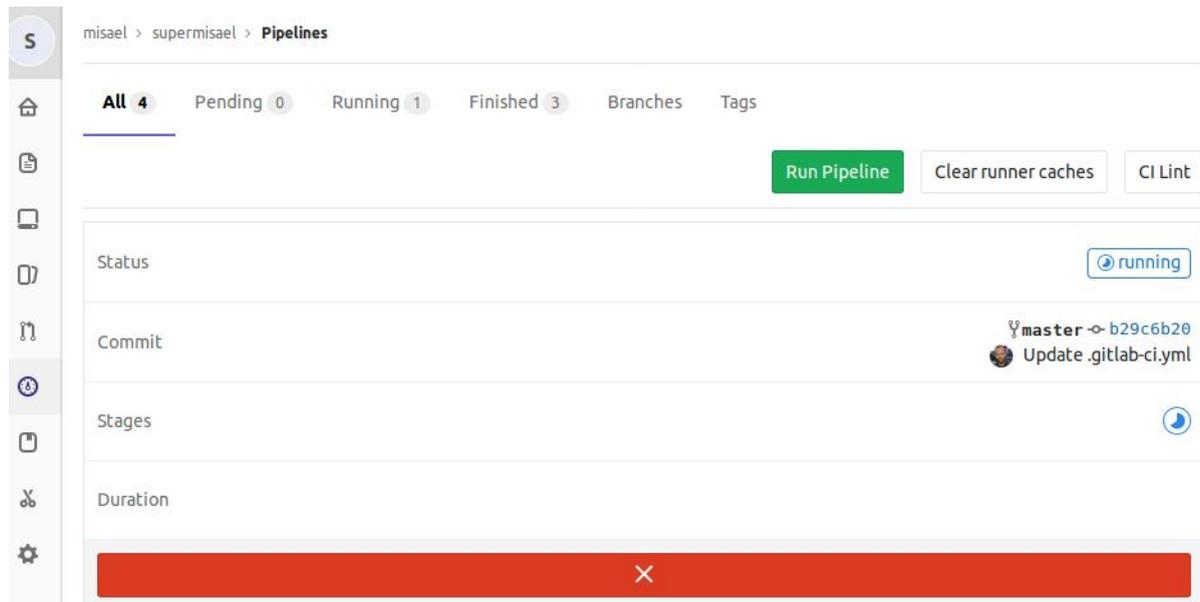
añadimos nuestra url de dominio y nuestro token y los datos que se nos piden

```
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://supergitlab.misael.gonzalonazareno.org
Please enter the gitlab-ci token for this runner:
sKV4yKXAJPcFNfv6io X
Please enter the gitlab-ci description for this runner:
[raul]: superrunner
Please enter the gitlab-ci tags for this runner (comma separated):
supertag
whether to run untagged builds [true/false]:
[false]:
Whether to lock the Runner to current project [true/false]:
[true]:
Registering runner... succeeded runner=sKV4yKXA
Please enter the executor: docker, shell, ssh, kubernetes, docker-ssh, parallels, virtualbox, docker+machine, docker-ssh+machine:
docker
Please enter the default Docker image (e.g. ruby:2.1):
alpine:latest
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!
root@raul:/etc/ssl/certs#
```

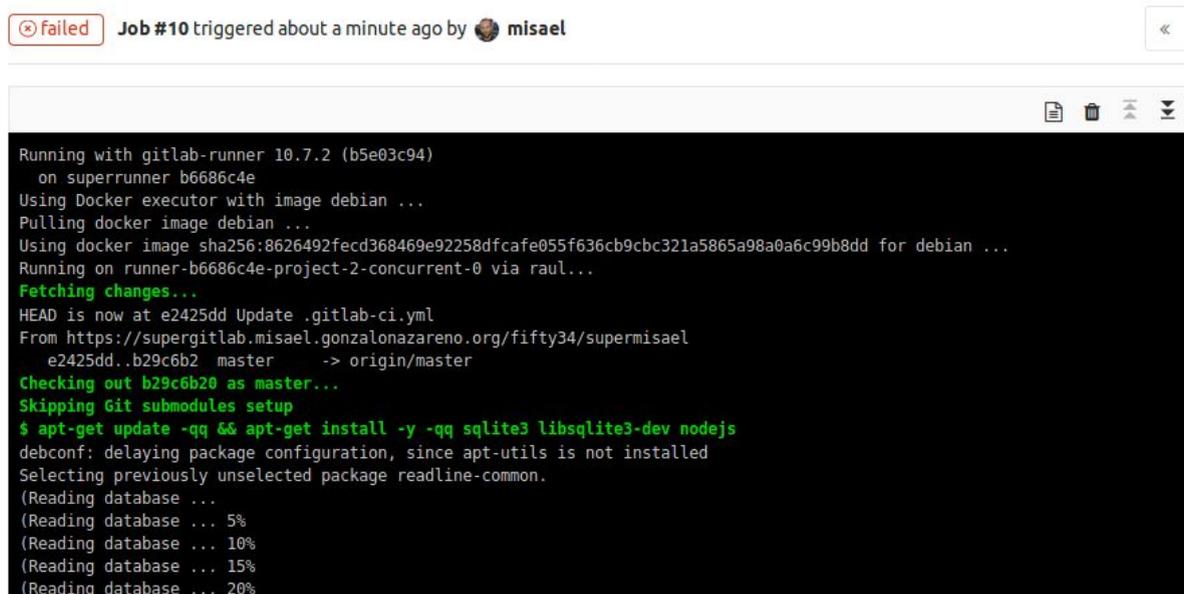
con esto , ya hemos registrado nuestro runner .

**** Importante :** tenemos que modificar nuestro `.gitlab-ci.yml` de forma que contenga el atributo `tag` (ya que al registrar el runner he especificado que solo ejecute los que tengan ese tag y que use una imagen específica ,con lo cual , puede fallar)

Ahora , en la pestaña “pipelines “ vemos los trabajos pendientes :



y si nos vamos a la subpestaña “jobs” podemos observar que está ocurriendo dentro del contenedor de las pruebas :

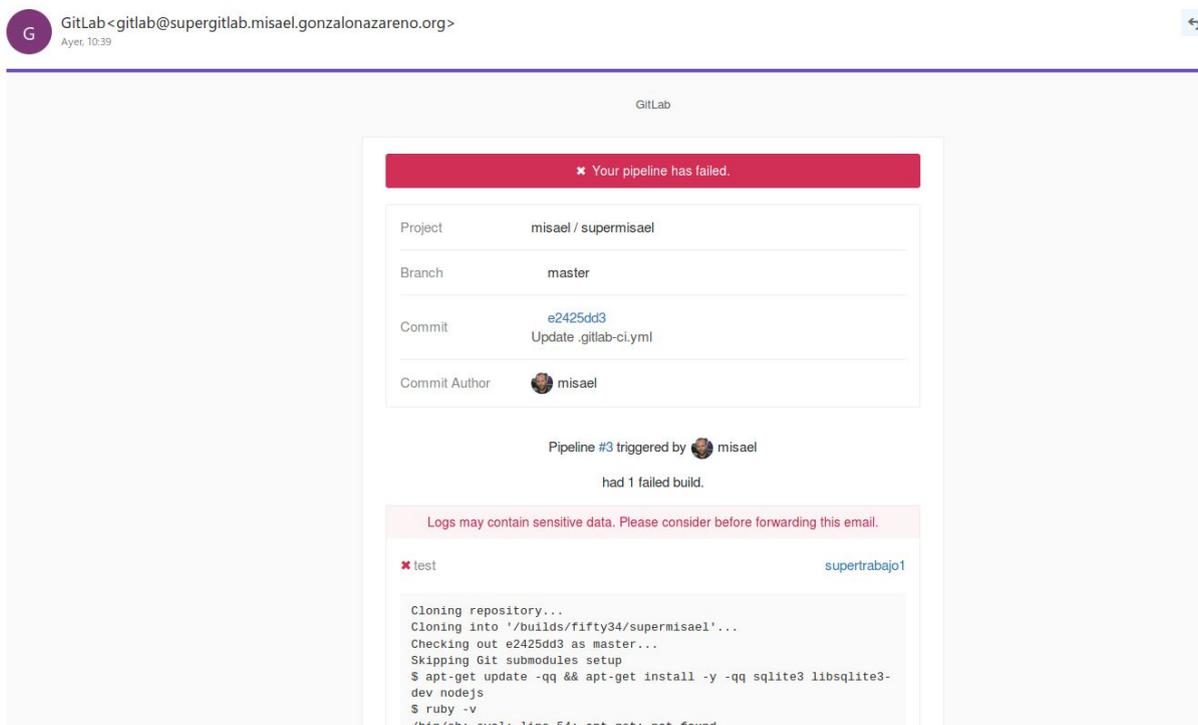


En mi caso , ha fallado porque el comando que he especificado en el fichero .gitlab-CI.yml no existe con la imagen especificada .

```
Processing triggers for libc-bin (2.24-11+deb9u3) ...
Processing triggers for ca-certificates (20161130+nmu1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
$ ruby -v
/bin/bash: line 58: ruby: command not found
ERROR: Job failed: exit code 1
```

también me ha enviado un correo

supermisael | Pipeline #3 has failed for master | e2425dd3



el otro “trabajo” que pertenecía al mismo fichero no se ha ejecutado y está en “pending” esperando para ser recogido por un runner habilitado para ello .

Mencionar que hay varios tipos de runners , los específicos para cada proyecto que ha sido este ejemplo , o los shared runners , que son runners compartidos por la mayoría y que solo un administrador puede lograr crear y asignar , como siempre a determinados proyectos y usuarios . También está los runners de grupos , aplicando a estos únicamente .

Lo bueno de instalarlo en otra maquina es que cualquier desarrollador experimentado puede habilitar su integración continua sin supervisión del administrador , configurando el mismo su runner en su propia máquina .

Gitlab Pages

Primero , veremos como habilitar la funcion de gitlab Pages en nuestro dominio , luego , repasaremos lo que tendrá que hacer un usuario para publicar una web html con contenido estático.

Gitlab pages hace uso de un demonio escrito en GO que ejecuta HTTP y puede escuchar en una dirección IP específica o bajo un dominio con certificados propios .

Existen varias maneras de habilitar Gitlab Pages , dependiendo si queremos crearlo en nuestro mismo dominio de gitlab con/sin HTTPS o si queremos hacerlo en otro dominio diferente con/sin HTTPS .

Nosotros habilitaremos Gitlab Pages en nuestro mismo dominio sin HTTPS

Primero , necesitaremos añadir un “wildcard DNS” a nuestro servicio DNS

```
; Instead, copy it, edit named.conf, and use that copy.
$TTL      86400
@         IN      SOA     raul.misael.gonzalonazareno.org. root.misael.gonza
                        10           ; Serial
                        604800        ; Refresh
                        86400         ; Retry
                        2419200       ; Expire
                        86400 )       ; Negative Cache TTL
;
@         IN      NS     raul.misael.gonzalonazareno.org.
@         IN      MX     10 raul.misael.gonzalonazareno.org.
$ORIGIN   misael.gonzalonazareno.org.
raul     IN      A      10.0.0.5
alberto  IN      A      10.0.0.8
josedomingo IN A 10.0.0.16
supergitlab IN A 172.22.200.57
*.superpaginas IN CNAME supergitlab
```

esto hará , que cuando busquemos cualquier dominio
“*.superpaginas.misael.gonzalonazareno.org” señale a nuestra instancia de gitlab

**En mi caso toca esperar a que la cache de mi servidor DNS de papión se reinicie.

Como vemos nuestro DNS nos corresponde correctamente

```
root@raul:/home/debian# dig holamundo.superpaginas.misael.gonzalonazareno.org
; <<> DiG 9.10.3-P4-Debian <<> holamundo.superpaginas.misael.gonzalonazareno.org
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: NOERROR, id: 41093
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;holamundo.superpaginas.misael.gonzalonazareno.org. IN A

;; ANSWER SECTION:
holamundo.superpaginas.misael.gonzalonazareno.org. 86400 IN CNAME supergitlab.misael.gonzalonazareno.org.
supergitlab.misael.gonzalonazareno.org. 86400 IN A 172.22.200.57
```

editamos nuestro fichero de configuracion de gitlab con lo siguiente

```
pages_external_url
"https://superpaginas.misael.gonzalonazareno.org"
```

y reiniciamos la instancia de gitlab y el servicio runnit

```
gitlab-ctl reconfigure
gitlab-ctl restart gitlab-pages
service gitlab-runsvdir restart
```

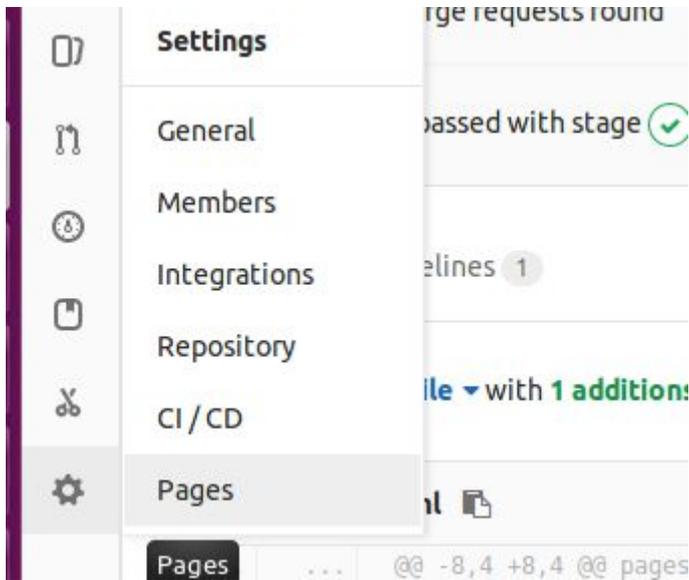
Ahora , nos vamos a nuestro repositorio y creamos una rama llamada "misael.io" y ahi subimos nuestro código html y un .gitlab-CI.yml como el siguiente :

```
image: alpine:latest

pages:
  stage: deploy
  script:
    - echo 'Nothing to do...'
  artifacts:
    paths:
      - public
  only:
    - misael.io
```

* el campo "only" sirve para especificar las ramas que atenderán los cambios
Podemos ver como va nuestra página como si de un jobs de integración continua se tratara.

Una vez desplegado el trabajo correctamente nos vamos a **project settings > Pages**



y nos aparecerá la url donde se ha desplegado nuestra página html .

Pages

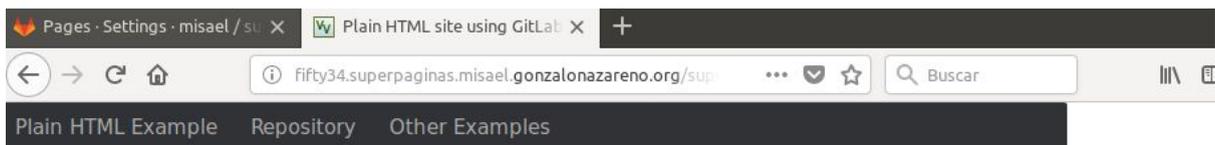
With GitLab Pages you can host your static websites on GitLab. Combined with the power of GitLab CI and the help of GitLab Runner you can deploy static pages for your individual projects, your user or your group.

Access pages

Congratulations! Your pages are served under:

<http://fifty34.superpaginas.misael.gonzalonazareno.org/supermisael>

Si accedemos :



Hello World!

This is a simple plain-HTML website on GitLab Pages, without any fancy static site generator.

Administrando nuestra instancia de Gitlab

Todo lo visto anteriormente ha sido en cuestión de instalaciones y funcionalidades a la hora de escalar nuestra instancia de GitLab para ofrecer mejores servicios o funcionalidades extras. Pero ¿cómo lo administramos realmente?

Bueno, al realizar la instalación mediante el paquete Omnibus, Gitlab se ha instalado automáticamente, con lo cual, no sabemos nada de sus ficheros de configuración, sus servicios, sus componentes etc.... En esta sección de la documentación hallaremos alguna de las cosas más fundamentales a la hora de mantener activa nuestra (super) instancia de gitlab**.

**Basado en este metodo de instalacion

1. Mantenimiento

Backups :

```
sudo gitlab-rake gitlab:backup:create
```

- Los backups se sitúan en el patch descrito en el fichero `/etc/gitlab/gitlab.rb`
- También se pueden subir a un cloud de almacenaje externo directamente o a sistemas de ficheros compartidos remotamente
- Para guardar la configuración de ficheros de configuración, como mínimo, tendrás que hacer un backup manual de `/etc/gitlab/gitlab.rb` (fichero de configuración) y `/etc/gitlab/gitlab-secrets.json` (clave para descifrar runners y usuarios doblemente autenticados)
- Para restaurar, necesitas tener activa una instancia de gitlab con la misma versión, y haber hecho una reconfiguración del sistema al menos una vez.

```
cp ./gitlab_copia{fecha} /var/opt/gitlab/backups

sudo gitlab-ctl stop unicorn
sudo gitlab-ctl stop sidekiq
# Verificar
sudo gitlab-ctl status
#restaurar
sudo gitlab-rake gitlab:backup:restore BACKUP={{fecha}}
#reiniciar
```

```
sudo gitlab-ctl restart
#comprobar
sudo gitlab-rake gitlab:check SANITIZE=true
```

Checks de integridad :

-Esto revisa uno por uno cada uno de los repositorios para comprobar su integridad

```
sudo gitlab-rake gitlab:git:fsck
```

-Esto revisa el "bind_dn" y la password de los usuarios de LDAP

```
sudo gitlab-rake gitlab:ldap:check
```

Limpieza :

-Esto limpia todos los repositorios que no se encuentren referenciados en la base de datos de la instancia. **PUEDE LLEVAR PÉRDIDAS DE DATOS**

```
sudo gitlab-rake gitlab:cleanup:dirs
```

Información de la instancia :

-Muestra información de cada componente y su version

```
sudo gitlab-rake gitlab:env:info
```

-Muestra informacion del fichero de configuracion

```
sudo gitlab-rake gitlab:check
```

-Restaurar authorized_keys y la caché de redis

```
sudo gitlab-rake gitlab:shell:setup
sudo gitlab-rake cache:clear
```

[más info](#)

2. Reiniciar gitlab y configuración

-Reiniciar la instancia de gitlab entera

```
sudo gitlab-ctl restart
```

-Reiniciar algun componente especifico de gitlab

```
sudo gitlab-ctl restart {{nginx}}
```

-Comprobar el estado de los servicios

```
sudo gitlab-ctl status
```

-Reconfigurar instancia de gitlab

```
sudo gitlab-ctl reconfigure
```

-Reiniciar demonio de interfaz web (cosecha propia)

```
sudo service gitlab-runsvdir restart
```

3. Mejorar rendimiento

-Como no es necesariamente importante para la administración , adjunto enlace a la documentación oficial .

[Rendimiento](#)

4. Componentes

-A la hora de entender mejor gitlab y saber por donde mirar por si alguna vez falla , conviene revisar los componentes de estos y dar una breve explicacion (tambien un enlace a su correspondiente documentación por si falla , acceder desde esta documentación directamente, donde se encontrara disponible más información).

1. Gitaly :

Es el servicio que se conecta a nuestros repositorios y provee el acceso a estos a los componentes que lo utilizan

[Gitaly](#)

2. Gitlab-Workhorse :

Cliente que acepta peticiones web , se llama así por su funcionamiento junto con el servidor web usado en Gitlab **Unicorn** que procesa las peticiones en Ruby

[Breve Historia](#)

3. LogRotate

Rotación de Logs

[LogRotate](#)

4. Node-exporter y Postgres-exporter y Redis-exporter

Permite medir los recursos de las máquinas como ram,disco usado,etc...

[activarlo](#)

También permite medir métricas de Postgres

[activarlo](#)

También permite medir métricas de la base de datos Redis

[activarlo](#)

5. Prometheus

Por defecto gitlab activa la integración con Prometheus para la recolección de métricas

[Prometheus](#)

6. Registry

Servicio extra añadido que permite guardar nuestras imágenes docker

[Registry](#)

7. Sidekiq

Proceso corriendo en segundo plano, protagonista de correr los trabajos de forma asíncrona

[Sidekiq](#)

Podemos ver todos nuestros componentes de gitlab y su estado ejecutando el comando

```
gitlab-ctl status
```

```
root@josedomingo:~# gitlab-ctl status
run: gitaly: (pid 27773) 485087s; run: log: (pid 17632) 488956s
run: gitlab-monitor: (pid 27784) 485087s; run: log: (pid 17625) 488956s
run: gitlab-workhorse: (pid 27813) 485086s; run: log: (pid 17628) 488956s
run: logrotate: (pid 12227) 2675s; run: log: (pid 17631) 488956s
run: nginx: (pid 28847) 484979s; run: log: (pid 17627) 488956s
run: node-exporter: (pid 27839) 485085s; run: log: (pid 17622) 488956s
run: postgres-exporter: (pid 27851) 485084s; run: log: (pid 17615) 488956s
run: postgresql: (pid 27903) 485084s; run: log: (pid 17617) 488956s
run: prometheus: (pid 27949) 485083s; run: log: (pid 17633) 488956s
run: redis: (pid 27958) 485083s; run: log: (pid 17624) 488956s
run: redis-exporter: (pid 27964) 485083s; run: log: (pid 17629) 488956s
run: registry: (pid 27972) 485083s; run: log: (pid 17621) 488957s
run: sidekiq: (pid 27997) 485080s; run: log: (pid 17619) 488957s
run: unicorn: (pid 28021) 485079s; run: log: (pid 17630) 488957s
root@josedomingo:~# █
```

5. Logs

-La parte más importante para ver cómo va nuestra instancia.

-Para las instalaciones omnibus, podemos acceder al log general de la instancia

```
# Tail all logs; press Ctrl-C to exit
sudo gitlab-ctl tail

# Drill down to a sub-directory of /var/log/gitlab
sudo gitlab-ctl tail gitlab-rails

# Drill down to an individual file
sudo gitlab-ctl tail nginx/gitlab_error.log
```

-Se pueden especificar distintas rutas **donde almacenar** los logs , así como **establecer las rotaciones** de estos mediante el fichero de configuración y habilitar **log forwarding**, todo esto en el fichero /etc/gitlab/gitlab.rb [más información](#).

-Estos son los logs que existen de diferentes funciones para profundizar mas

1. Production.log

Contiene información sobre todas las peticiones al servidor, también existe una variante en formato json .

2. Api_json .log

Contiene información de las peticiones dirigidas a la API

3. application.log

Contiene los datos realizados en la aplicación, como usuarios creados, proyectos etc

4. Githost.log

Contiene los accesos fallidos de gitlab hacia los repositorios git, normalmente útil solo para los desarrolladores

5. Sidekiq.log

Contiene información de los procesos que ejecuta sidekiq en segundo plano y que pueden llevar bastante tiempo

6. Gitlab-shell.log

Contiene los logs de los comandos git usados por gitlab para, por ejemplo, proporcionar acceso por ssh a los repositorios

7. unicorn_stderr.log

Es el log del servidor Unicorn

8. Repocheck.log

Contiene información de cuando un repositorio es chequeado (véase [mantenimiento](#))

9. Reconfigure logs

son los logs que contienen las veces que se ha reconfigurado gitlab

10. Sidekiq_exporter.log

Si prometheus y sidekiq-exporter están activados, se creará este fichero de log acerca de la recolección de métricas

-En este enlace se especifica más información acerca de estos diferentes ficheros de logs, y su ubicación exacta en nuestro sistema

[Logs Documentación](#)