

# Canalización GitLab CI. Ejecutar secuencia de comandos a través de SSH al servidor remoto

VIERNES. 18 DE JUNIO DE 2021- 8 MINUTOS

DESARROLLADORES GITLAB SSH CI / CD



Foto de [Jantine Doornbos](#) en [Unsplash](#)

Cuando piensa en implementar en un servidor remoto, SSH es el primer protocolo de red que le viene a la mente. Agregar GitLab CI/CD en la parte superior le permitirá aprovechar la automatización. Para usar la canalización de CI/CD de GitLab junto con las conexiones SSH, primero es necesario configurar GitLab y me gustaría mostrarle cómo configurarlo y ejecutar un script simple.

## Requisito previo

- cuenta GitLab
- Servidor remoto (estoy usando la máquina virtual Linux de Azure)

## Agenda

1. Crear nuevo proyecto de GitLab
2. Crear y agregar claves SSH
3. Crear y ejecutar canalización de CI/CD de GitLab

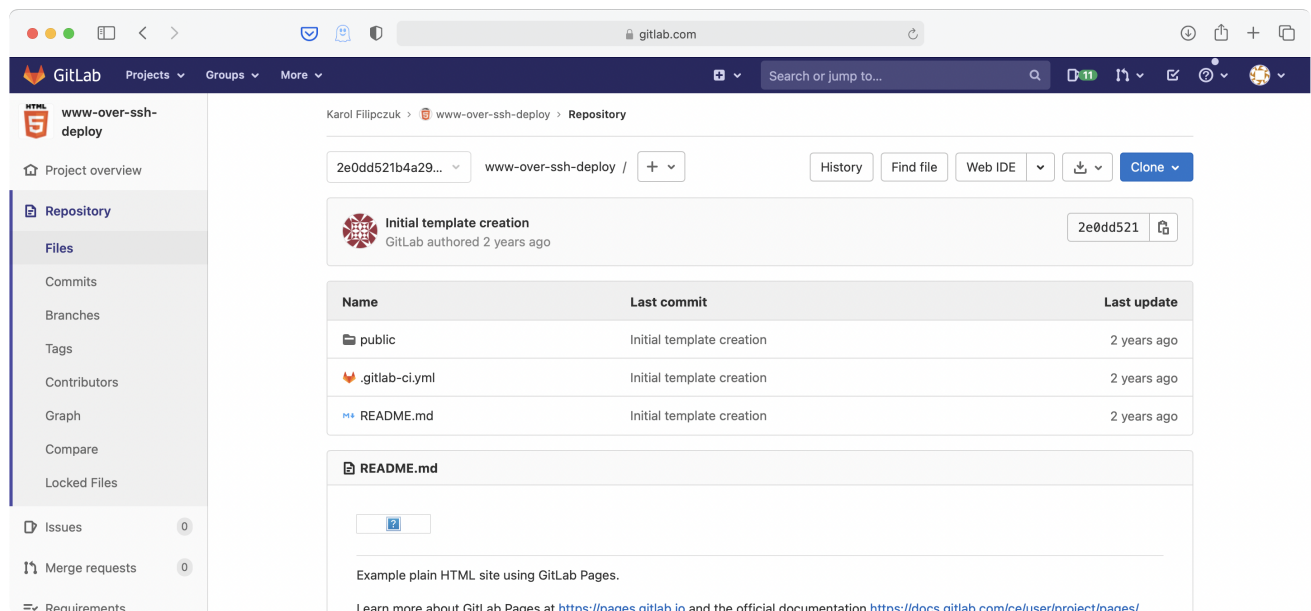
## Crear nuevo proyecto de GitLab

Como primer paso crearemos el proyecto GitLab.

Inicie sesión en GitLab y navegue hasta New project -> Create from template ->

Pages/Plain HTML -> Use template. Dale un nombre de proyecto y presiona Create project.

Esto creará un proyecto html simple y simple.



Páginas/Proyecto HTML simple

La plantilla creó el archivo README.md, .gitlab-ci.yml directorio inicial y público con archivos index.html y .style.css

## Crear y agregar claves SSH

Ya tenemos un proyecto de ejemplo, ahora necesitamos crear claves SSH. Se utilizarán para conectarse a nuestro servidor remoto. Cada vez que se ejecuta la canalización de CI/CD de GitLab, se utiliza GitLab Runner.

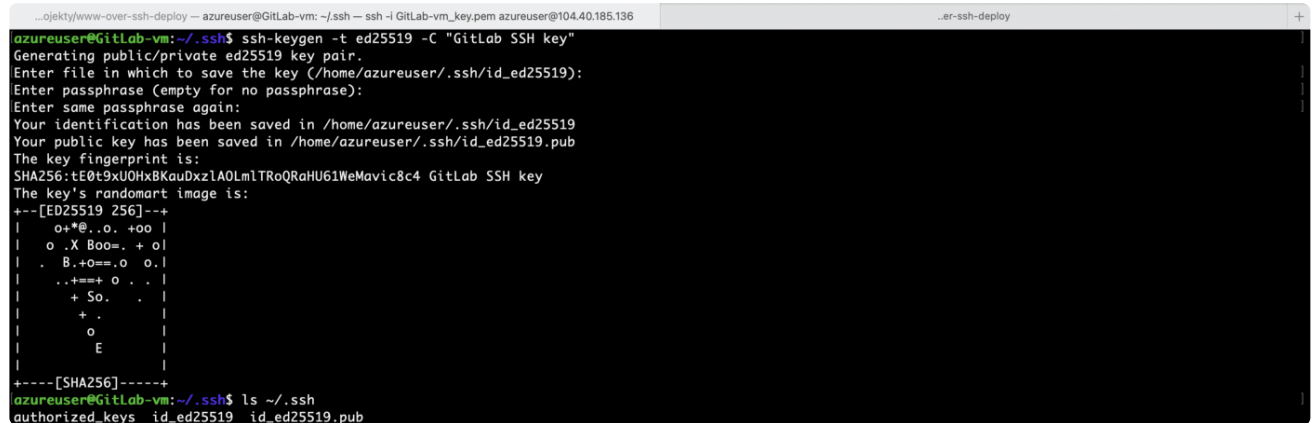
GitLab Runner es una aplicación cuya tarea es ejecutar trabajos en la canalización de CI/CD de GitLab. Usted mismo puede instalar GitLab Runner en su infraestructura o

puede aprovechar Shared Runners mantenido por GitLab. Tienes 400 minutos al mes gratis desde GitLab. Usaremos Shared Runner, ya que son fáciles de usar y no se necesita configuración para nuestro ejemplo. Necesitamos configurar las claves SSH de manera que el trabajo ejecutado por Shared Runner pueda acceder a nuestro servidor remoto a través de una conexión SSH.

## Crear clave SSH

Puede crear una nueva clave SSH en cualquier entorno, incluso en su entorno local. Cuando cree una nueva clave SSH, recibirá dos claves: privada y pública. Es importante que GitLab tenga una clave privada y su servidor remoto tenga una clave pública. Es por eso que no importa dónde cree las claves, solo importa compartirlas en consecuencia con GitLab y el servidor remoto.

Tengo una máquina virtual Linux en Azure y la usaré para esta publicación. Crearé una nueva clave ssh usando la máquina virtual. La recomendación de GitLab es crear la clave SSH tipo ED25519, que es más segura que RSA. Para crear una nueva clave, ejecute `ssh-keygen -t ed25519 -C "GitLab SSH key"`. El texto después de `-c` opción es un comentario y puede cambiarlo.



```

...ojekty/www-over-ssh-deploy -- azureuser@GitLab-vm: ~/ssh -- ssh -i GitLab-vm_key.pem azureuser@104.40.185.136
...er-ssh-deploy
+
azureuser@GitLab-vm:~/ssh$ ssh-keygen -t ed25519 -C "GitLab SSH key"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/azureuser/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/azureuser/.ssh/id_ed25519
Your public key has been saved in /home/azureuser/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:tE0t9xUOHxBKauDxz1A0Lm1TRoQRaHU61WeMavic8c4 GitLab SSH key
The key's randomart image is:
+--[ED25519 256]--+
|  o+*E..o. +oo |
|  o.X Boo=. + ol
|  . B.+o=.o o.l
|  ..+==+ o . . |
|  + So. . |
|  + . |
|  o |
|  E |
+-----[SHA256]-----+
azureuser@GitLab-vm:~/ssh$ ls ~/.ssh
authorized_keys  id_ed25519  id_ed25519.pub

```

Crear clave SSH ed25519

La clave se creará en el directorio predeterminado que para Linux es `/home/<user>/.ssh`. No especifique la frase de contraseña, de lo contrario será engorroso para la canalización de CI/CD de GitLab. Debería tener dos archivos nuevos en el `.ssh` directorio:

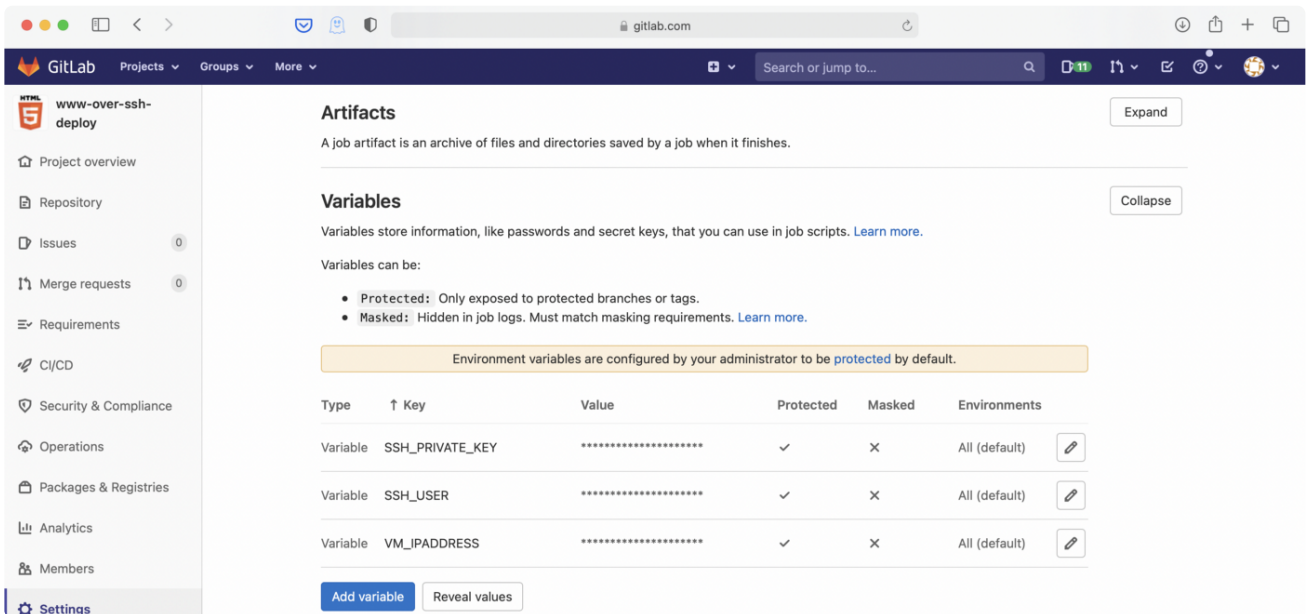
- `id_ed25519`- llave privada
- `id_ed25519.pub`- Llave pública

## Agregar clave privada como variable de GitLab

Copie el contenido de la clave privada y vuelva al proyecto GitLab. Navegar a **Settings** -> **CI/CD** -> **Variables** -> **Expand** -> **Add Variable**. La variable de GitLab es un par clave-valor. Asigne un nombre a la clave `SSH_PRIVATE_KEY` pegue la clave privada en el campo de valor. Haga clic en **Add Variable**.

Agregue dos variables más:

- `SSH_USER`— nombre del usuario en el servidor remoto
- `VM_IPADDRESS`— Dirección IP del servidor remoto

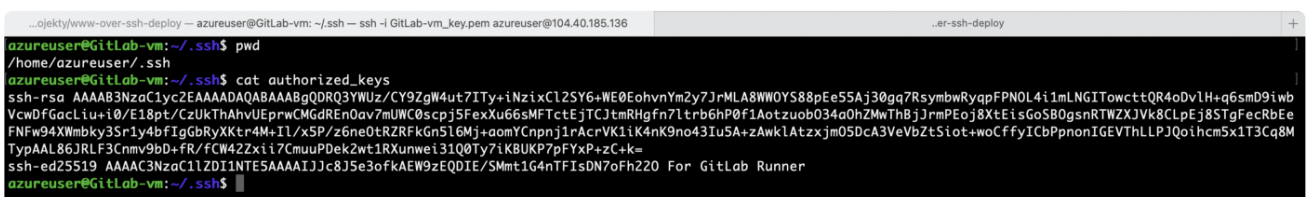


Variables añadidas

## Agregar clave pública al servidor remoto

Copie el contenido de la clave pública y vuelva al servidor remoto. Inicie sesión como el mismo usuario que ha especificado en `SSH_USER` la variable de GitLab. Si aún no tienes este usuario, es hora de crearlo.

Navegar a `/home/<username>/.ssh`. Si el directorio `.ssh` no existe, créelo. Pegue la clave pública en `authorized_keys` el archivo. Si no tiene un `authorized_keys` archivo, créelo. Aquí hay una captura de pantalla de mi VM (que eliminé antes de publicar, por lo que ahora son inútiles).



Configuración de llaves autorizadas

# Crear y ejecutar canalización de CI/CD de GitLab

Es hora de crear la canalización de CI/CD de GitLab. Queremos lograr dos objetivos usando SSH: registrar el nombre de host del servidor remoto y crear un archivo de ejemplo en el directorio de inicio del usuario.

La canalización está definida `.gitlab-ci.yml` tenemos dos opciones para crear/editar:

1. Directamente en el proyecto GitLab en el navegador web, podemos editar `.gitlab-ci.yml` confirmar cambios
2. Clone el repositorio, edite `.gitlab-ci.yml` en su editor de código favorito, confirme los cambios y envíelo a GitLab

Iré con la opción número 2, es una forma más adecuada de manejar `.gitlab-ci.yml`.

Puede clonar el repositorio usando el comando `git clone <repo_address>` y la dirección del repositorio que puede encontrar en el repositorio de GitLab haciendo clic en el `clone` botón.

Después de la clonación, abra los archivos ya existentes `.gitlab-ci.yml` que se crearon como parte de la plantilla de páginas/HTML sin formato.

```
1  image: alpine:latest
2
3  pages:
4    stage: deploy
5    script:
6      - echo 'Nothing to do...'
7  artifacts:
8    paths:
9      - public
10  only:
11    - master
```

`.gitlab-ci-ssh.yml` hosted with ❤ by GitHub

[view raw](#)

`.gitlab-ci.yml` original

Necesitamos agregar una `before_script` sección y una sección de actualización `script`.

```
1  image: alpine:latest
2
3  pages:
4    stage: deploy
5    before_script:
6      - 'command -v ssh-agent >/dev/null || ( apk add --update openssh )'
7      - eval $(ssh-agent -s)
8      - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -
9      - mkdir -p ~/.ssh
```

```

10 - chmod 700 ~/.ssh
11 - ssh-keyscan $VM_IPADDRESS >> ~/.ssh/known_hosts
12 - chmod 644 ~/.ssh/known_hosts
13 script:
14 - ssh $SSH_USER@$VM_IPADDRESS "hostname && echo 'Welcome!!!' > welcome.txt"
15 artifacts:
16   paths:
17     - public
18   only:
19     - master

```

.gitlab-ci-ssh-v2.yml hosted with ❤ by GitHub

[view raw](#)

.gitlab-ci.yml final

.gitlab-ci.yml define la canalización. Utiliza la imagen de la ventana acoplable alpine:latest para ejecutar trabajos definidos en la canalización. Solo tenemos un trabajo pages.

El trabajo se ejecuta en stage: deploy. No definimos ninguna etapa, pero tenemos 5 etapas predeterminadas para usar: .pre, build, test, deploy, .post. No importa en nuestro caso, ya que nuestra canalización en este momento es simple y no requiere configurar etapas.

Luego tenemos before\_script que se explica por sí mismo y se ejecutará antes script del comando. Explicuemos el guión línea por línea:

- command -v ssh-agent > /dev/null || (apk add --update openssh)— comprueba si ssh-agent ya está instalado y, si no, instálelo
- eval \$(ssh-agent -v)— inicia el agente ssh
- echo "\$SSH\_PRIVATE\_KEY" | tr -d '\n' | ssh-add — agrega la clave privada ssh almacenada en la variable SSH\_PRIVATE\_KEY al almacén del agente
- mkdir -p ~/.ssh & chmod 700 ~/.ssh — crea .ssh un directorio y asigna los permisos correctos
- ssh-keyscan \$VM\_IPADDRESS >> ~/.ssh/known\_hosts — verifica la clave pública en el servidor remoto usando la dirección IP almacenada en la VM\_IPADDRESS variable y la agrega a los hosts conocidos. Está protegiendo del ataque de intermediarios y es necesario para trabajar, de lo contrario, el trabajo fallará.
- chmod 644 ~/.ssh/known\_hosts — asignar los permisos correctos
- Para obtener más información, consulte los documentos de GitLab [aquí](#)

scriptes donde se define nuestro código real para ejecutar. Simplemente queremos imprimir el nombre de host en el registro de trabajo y luego crear un archivo de ejemplo en el host remoto.

ssh \$SSH\_USER@\$IP\_ADDRESS "hostname && echo 'Welcome!!!' > welcome.txt" se conectará a



través de SSH como el usuario especificado en `SSH_USER` la variable al servidor remoto, luego ejecutará el comando `hostname` que imprimirá el nombre del host y repetirá el `Welcome!!!` archivo `welcome.txt` que se creará en el servidor remoto en `SSH_USER` el directorio de inicio.

artifacts especificar qué artefactos usar en la implementación. No lo estamos usando en nuestro ejemplo.

only especifique que el trabajo solo debe ejecutarse si se inserta algún cambio en `master` la rama del repositorio.

Después de realizar los cambios, debemos confirmarlos y enviarlos al repositorio.

```

...ojekty/www-over-ssh-deploy -- azureuser@GitLab-vm: ~/ssh -- ssh -i GitLab-vm_key.pem azureuser@104.40.185.136
+ www-over-ssh-deploy git:(master) * git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitlab-ci.yml

no changes added to commit (use "git add" and/or "git commit -a")
+ www-over-ssh-deploy git:(master) * git add .gitlab-ci.yml
+ www-over-ssh-deploy git:(master) * git commit -m "GitLab pipeline via SSH"
[master b838445] GitLab pipeline via SSH
Committer: Karol Filipczuk <karolfilipczuk@MacBook-Air-Karol.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

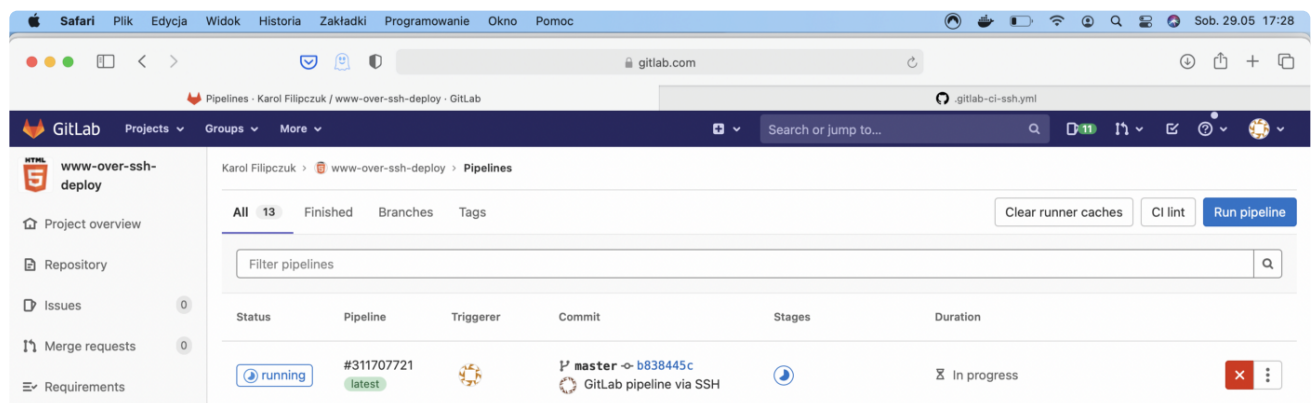
1 file changed, 1 insertion(+), 1 deletion(-)
+ www-over-ssh-deploy git:(master) git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 322 bytes | 322.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/filip5114/www-over-ssh-deploy.git
4977496..b838445 master -> master

```

*Agregar, confirmar y enviar cambios al repositorio*

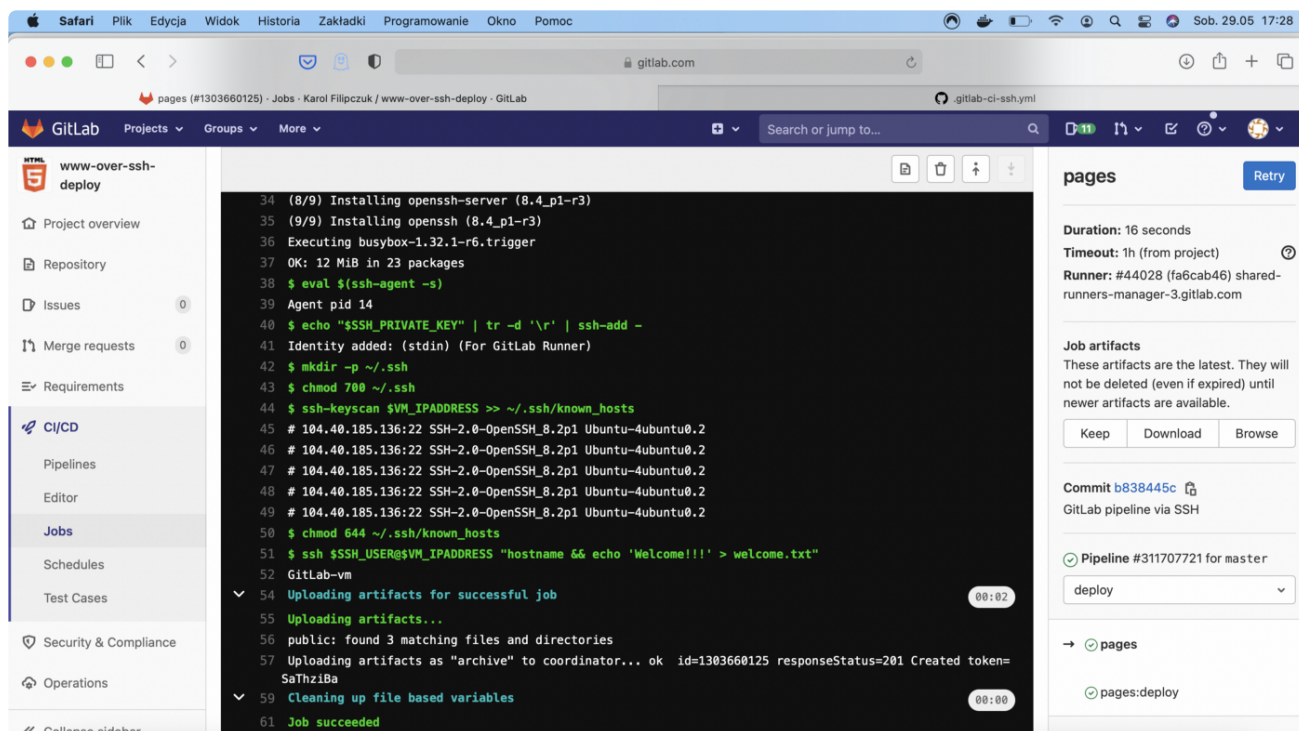
Una vez que el cambio se inserta en la rama maestra, se activará GitLab CI/CD.

Navegue hasta `CI/CD` -> `Pipelines` y debería ver la canalización en estado de ejecución.



Canalización en ejecución

Haga clic en él y haga clic en el trabajo `pages` para ver los registros.



### Registros de trabajo de Pipeline

El trabajo debería terminar rápidamente, en mi caso tardó 16 segundos. La última línea muestra que el trabajo se ejecutó correctamente. La línea 51 muestra `script` parte de `.gitlab-ci.yml` en la línea 52 podemos ver el nombre de host del servidor remoto, que es exactamente lo que queríamos lograr. Comprueba tu servidor remoto, lo encontrarás `welcome.txt` allí.

¡Eso es todo! Creamos con éxito un nuevo proyecto de GitLab, configuramos la conexión SSH al servidor remoto y creamos una canalización simple de GitLab CI/CD para ejecutar el script a través de SSH al servidor remoto.

Gracias por leer.

[Canalización GitLab CI. Cree una imagen acoplable en el trabajo de canalización. »](#)

### Artículos Relacionados

- [¿Cómo utilizar datos confidenciales de forma segura en el lanzamiento de EC2 con Secret Manager?](#)
- [Implementar desde Gitlab a AWS EC2](#)
- [Canalización GitLab CI. Cree una imagen acoplable en el trabajo de canalización.](#)






karol filiczuk  
DevOps y la nube

Twittear [Compartir](#)

Mantengámonos en contacto

dirección de correo

Suscribir

Karol Filipczuk © 2022   
Tema índigo de Kopplin