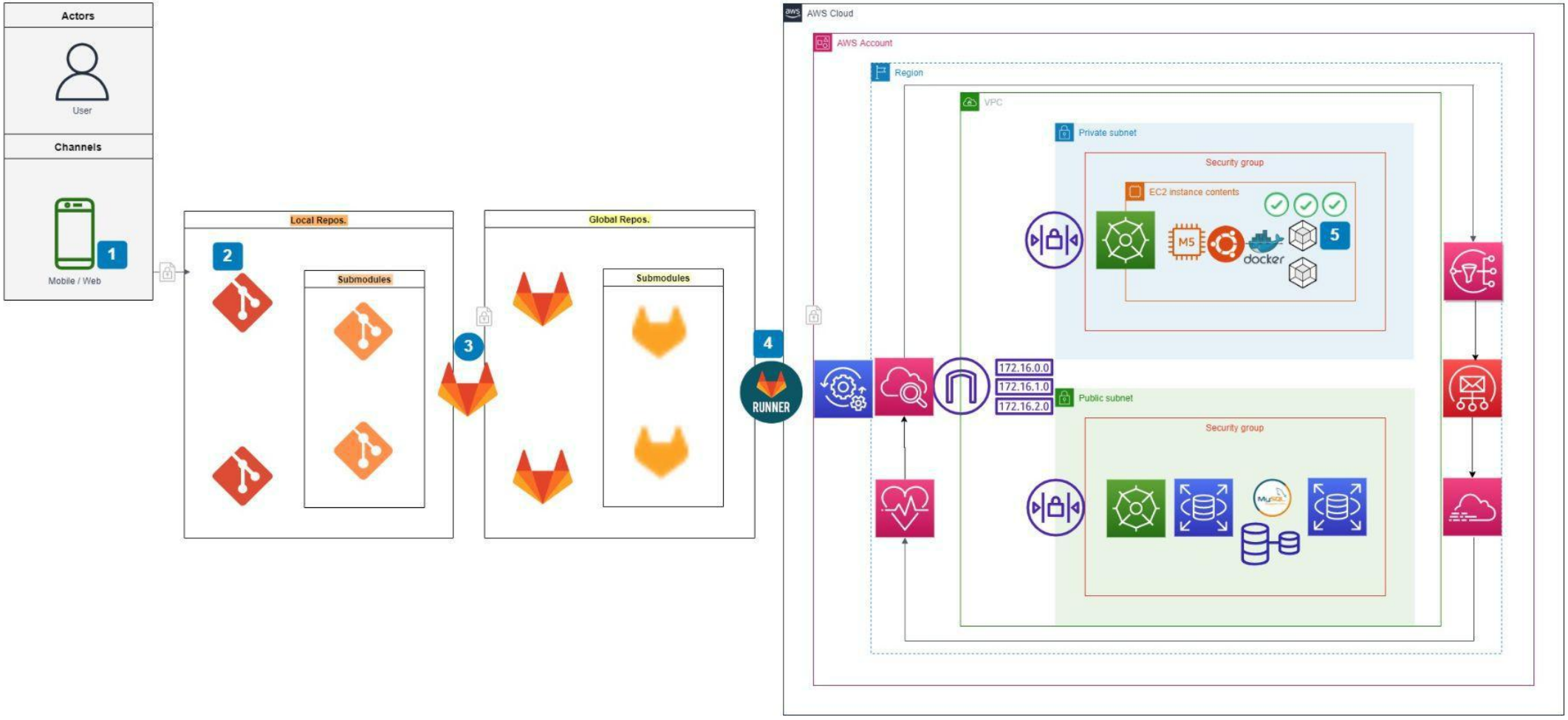
Servicios Independientes.

Multi Servicios. etecnic
Git Submodules.



Realizado por Oscar Macias.
Fecha: 19-01-2023.

Flujo de integración continua.

1. Acceso del usuario a los repositorios locales.
   Requisitos:
   - Cuenta GitLab.
   - Configuración de acceso SSH por Token.
   - Acceso a lectura de repositorios por proyecto.

2. Lanzamientos de cambios sobre los repositorios git.
   - Acceso de escritura sobre repositorios nuevos por proyecto.

3. Acceso a los repositorios globales.

4. Se ejecuta el runner del pipeline para hacer la construcción del pipeline.

   Requisitos:

   - Instalar GitLab CI en servidor Ubuntu.

```
$ yes | sudo apt-get install curl openssh-server ca-certificates postfix
```

La siguiente sentencia instalará dentro del servidor los paquetes requeridos para la conexión, descarga y comunicación por medio de los servicios de acceso a sitios WEB, Conexión segura por consola y Correo electrónico.

```
$ curl -sS https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash
```

La siguiente sentencia descargara dentro del servidor el paquete requerido para la preinstalación por medio de los servicios de acceso a sitios WEB. Luego filtra la salida instalando el mismo.

```
$ sudo EXTERNAL_URL="http://gitlabce.example.com" apt-get install gitlab-ce
```

La siguiente sentencia declarará dentro del servidor una variable requerida para la instalación por medio de los servicios de acceso a sitios WEB.

5. Comienza la integración continua sobre los proyectos y sub módulos dentro del entorno.

Requisitos:

- Instalar GitLab Runner en el servidor Ubuntu.

```
# Linux x86-64
$ sudo curl -L --output /usr/local/bin/gitlab-runner
"https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64"
```

La siguiente sentencia descargara dentro del servidor el paquete requerido, según la arquitectura y distro del sistema operativo actual, para la instalación por medio de los servicios de acceso a sitios WEB. Finaliza la descarga sobre la ruta absoluta indicada para ello en la salida del comando

```
$ dpkg -i gitlab-runner_<arch>.deb
```

La siguiente sentencia instalará dentro del servidor el paquete requerido, según la arquitectura y distro del sistema operativo actual.

```
$ sudo chmod +x /usr/local/bin/gitlab-runner
```

La siguiente sentencia establece privilegios de usuario sobre ejecución dentro del servidor sobre la ruta absoluta donde finalizó y contiene el resultado del binario descargado desde el sitio WEB de repositorios en AWS.

```
$ sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash
```

La siguiente sentencia creará dentro del servidor un usuario requerido, con su respectiva ruta absoluta de unidad personal, igual estableciendo la consola por defecto a utilizar dentro del sistema operativo.

```
$ sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner
```

La siguiente sentencia instalará dentro del servidor el paquete requerido, estableciendo el usuario requerido, con su respectiva ruta absoluta de unidad personal de trabajo, igual estableciendo la consola por defecto a utilizar dentro del sistema operativo.

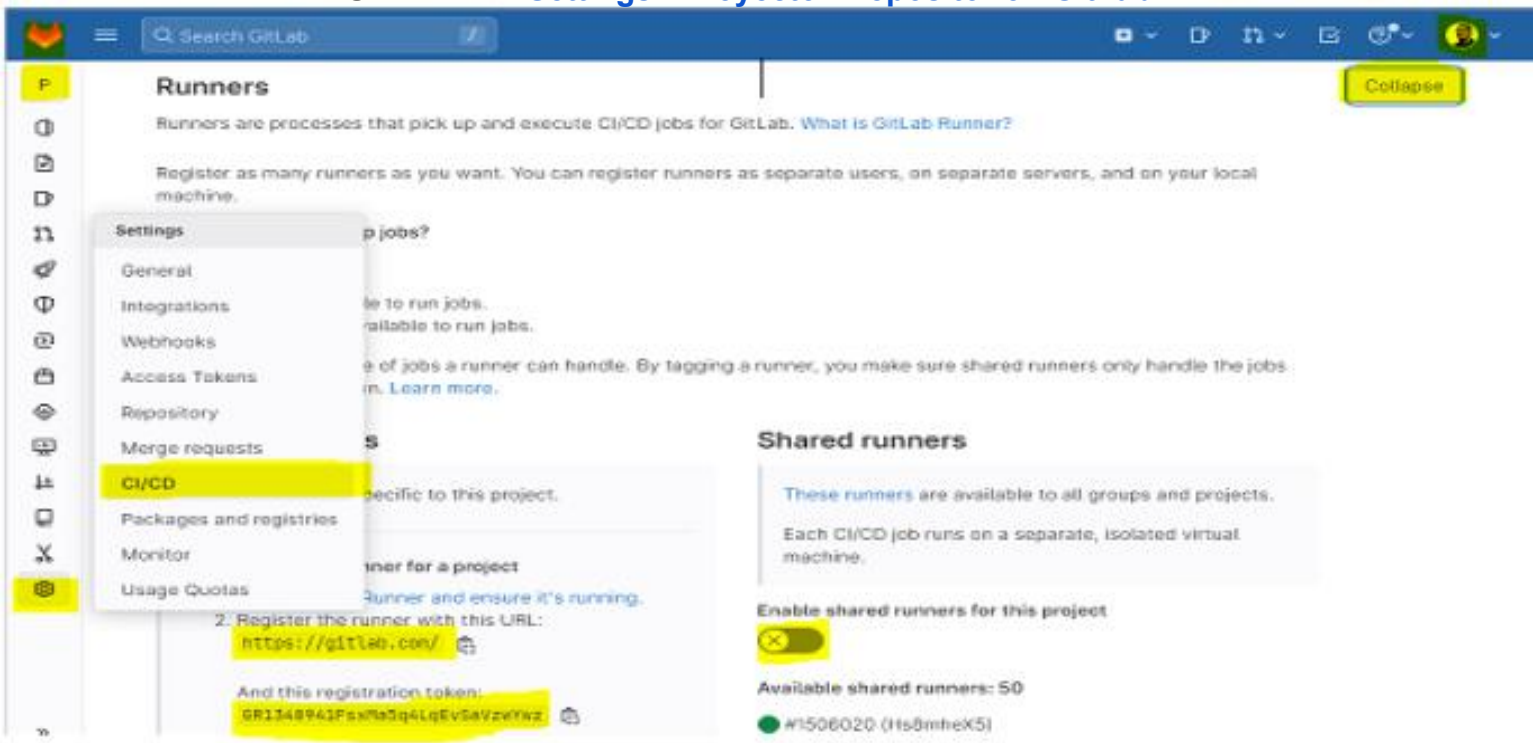
```
$ sudo gitlab-runner start
```

La siguiente sentencia inicia dentro del servidor como super usuario el servicio requerido, estableciendo el alta del demonio dentro del sistema operativo.

- Registrar un Runner para GitLab en el servidor Ubuntu.

Registro sobre la consola WEB de GitLab.

URL: CI/CD Settings · Proyecto / Repositorio · GitLab



La siguiente URL nos permite acceder a la configuración CI/CD dentro de GitLab correspondiente al actual repositorio del proyecto. Aquí evidenciamos los valores requeridos para continuar con el correcto registro del corredor dentro del servidor.
Por defecto y con pago están activos los corredores públicos de GitLab, **se debe desactivar los mismos para que no prioricen a nuestras canalizaciones, generando peticiones de suscripción de servicio, deteniendo la ejecución de la misma.**

Registro sobre el servidor Ubuntu.

```
    $ sudo -E gitlab-runner register -h
\ --config /tmp/test-config.toml \ --
non-interactive \
--url https://gitlab.com \
--registration-token __REDACTED__
\ --name test-runner \
--tag-list test \
--locked \ --
paused \ --
executor shell /
```

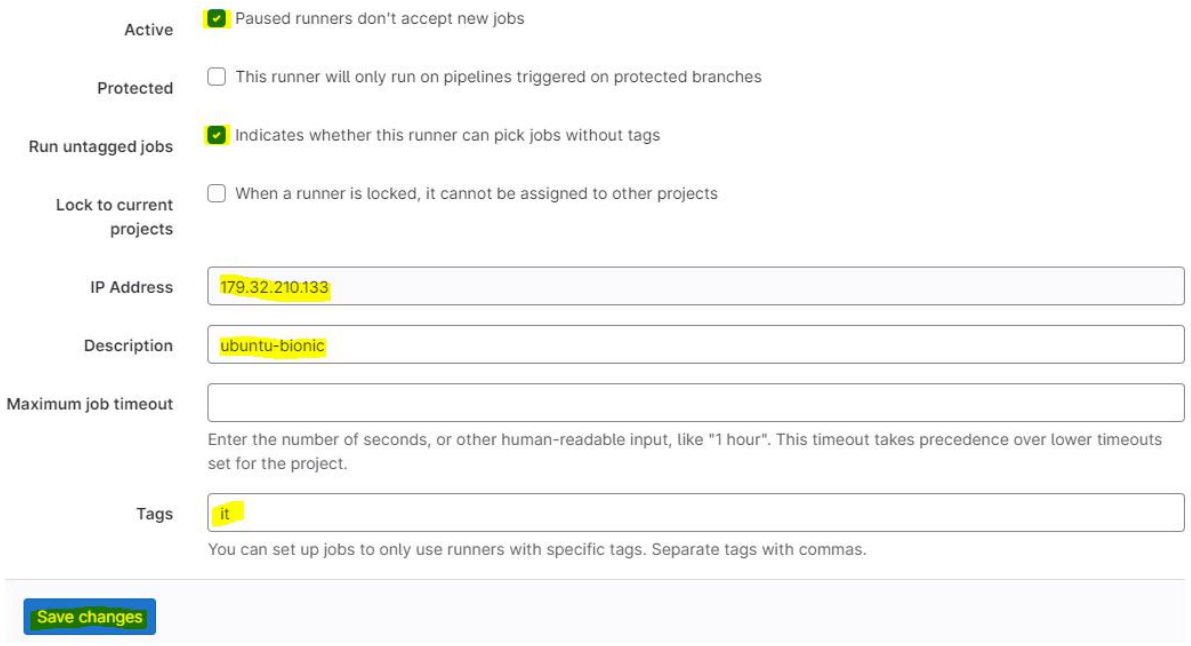
La siguiente sentencia registra dentro del servidor el corredor requerido, estableciendo dentro de la distro del sistema operativo actual, el archivo de configuración, la url y el token de registro para el acceso al sitio WEB, nombre y etiqueta de identificación del corredor. Finalizando con el tipo de ejecutor para la salida del comando.

```
        $ cat > /tmp/test-config.template.toml << EOF
concurrent = 1
check_interval = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "test-runner"
  url = "https://gitlab.com"
  token = "__REDACTED__"
  executor = "shell"
  [runners.cache]
    [runners.cache.s3]
    [runners.cache.gcs]
EOF
```

La siguiente sentencia imprime dentro del archivo de configuración toml, sobre el servidor dicho corredor requerido, establecido en la distro del sistema operativo actual. Finalizando a la salida del archivo de configuración.

Configuración sobre la consola WEB de GitLab.

URL: Edit · #19712554 (RUNNER) · Proyecto / Repositorio · GitLab



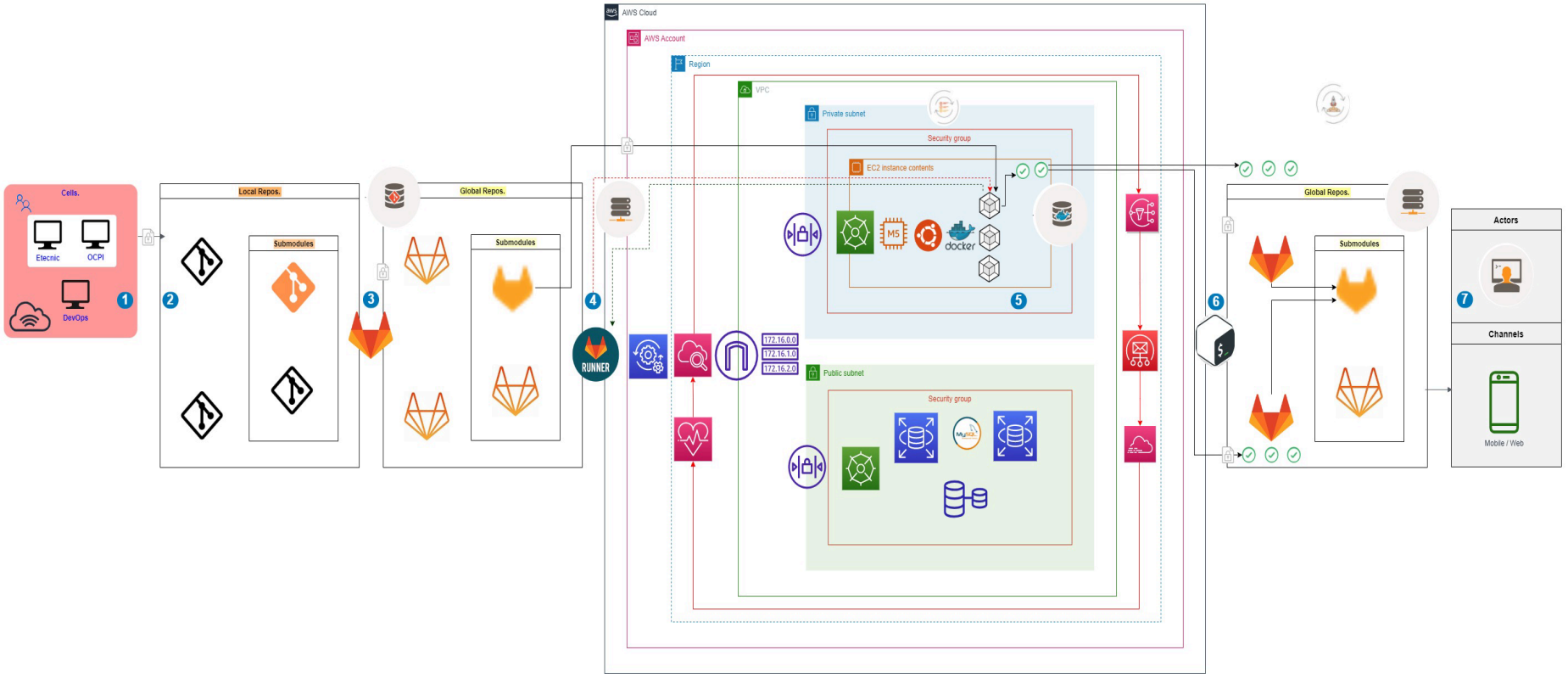
La siguiente URL nos permite acceder a editar la configuración del corredor CI/CD dentro de GitLab correspondiente al actual repositorio del proyecto. Aquí evidenciamos los valores requeridos para continuar con el correcto registro del corredor dentro del servidor.

Por defecto no vienen activadas las funciones tal cual la imagen. Para evitar errores con el corredor de GitLab, **se debe activar la función que corra trabajos sin etiquetas.**

Pipeline sobre cambios en un sub módulo.

Al subir un cambio en cualquier submódulo en común como sub repositorio con dos o más proyectos, desencadenará una canalización hija, la cual actualizará todos los proyectos dependientes según la rama del entorno o ambiente solicitado con los cambios generados. Con ello se hace uso del principio de microservicios independizando cualquier cambio mantenimiento sobre cada repositorio global.

Multi Servicios. etecnic
Pipelines CI/CD.



6. Ejecuta el trigger para continuar con la integración continua.

Código Fuente de Pipeline para Sub Módulos: .gitlab-ci.yml

```yaml
variables:
    TEST_VAR: "Begin - Update all git submodule in Etecnic projects."
```

Declaramos las variables globales, para uso de todos los Jobs.

```yaml
stages:
    - Build
    - Triggers
```

Establecemos las etapas correspondientes al flujo de la canalización.

```yaml
Build-job1:
    stage: Build
    script:
        - echo $TEST_VAR
```

Creamos un trabajo con su respectiva etapa y el guión con las sentencias a ejecutar.

```yaml
Trigger-job1:
    stage: Triggers
    when: on_success
    trigger:
        project: pruebas-it/proyecto_a
        branch: main
```

Creamos un desencadenador que cada vez que sea satisfactorio el trabajo, llame a la canalización hija que corresponde al proyecto remoto sobre la rama del ambiente o entorno correspondiente.

Código Fuente de Pipeline para Proyectos: .gitlab-ci.yml

```
variables:
    CHANGES: $(git status --porcelain | wc -l)
```

Declaramos una variable global que pueda evaluar una sentencia en BASH donde la salida de un comando, se filtre con el número correspondiente de líneas impresas.

```
job2:
    stage: test
    extends: .test-dev
    only:
        variables: [ $CI_PIPELINE_SOURCE == "push”]
```

Declaramos dentro de un trabajo una función con una extensión de código. Donde evaluamos solamente un evento que sea igual la variable predefinida por GitLab CI al estado del push, con ello, se ejecute la porción de código mapeada.

```
workflow:
    stage: deploy
    extends: .deploy-dev
    rules:
    - if: $CI_COMMIT_BRANCH && $CI_OPEN_MERGE_REQUESTS && $CI_PIPELINE_SOURCE == "push"
      when: never
    - when: always
```

Establecemos un flujo de trabajo que administra varios eventos y reglas, las cuales son explícitas por defecto. Debido a ello bloqueamos las validaciones que no deseamos que se ejecuten alguna vez. Para las demás declaramos que se ejecuten siempre.

```
.test-dev:
    script: exit
```

Función declarada para ejecutarse en el momento que el evento valide y permita la acción.

```
.deploy-dev:
    before_script:
```

Establecemos una serie de sentencias a recorrer antes de llegar al guión principal de la función.

- Acceso por SSH en el Runner hacia GitLab desde el servidor Ubuntu.

```
        - 'which ssh-agent || (apt-get update -y && apt-get install openssh-client -y)'
        - eval $(ssh-agent -s)
```

Las siguientes sentencias evalúan si existe el paquete instalado dentro del servidor. Dado el caso de no encontrarlo, procederá a actualizar los paquetes actuales y binarios del sistema. Junto a ello procederá instalar la dependencia para la conexión segura por consola que requerimos para autenticarnos dentro de la canalización.

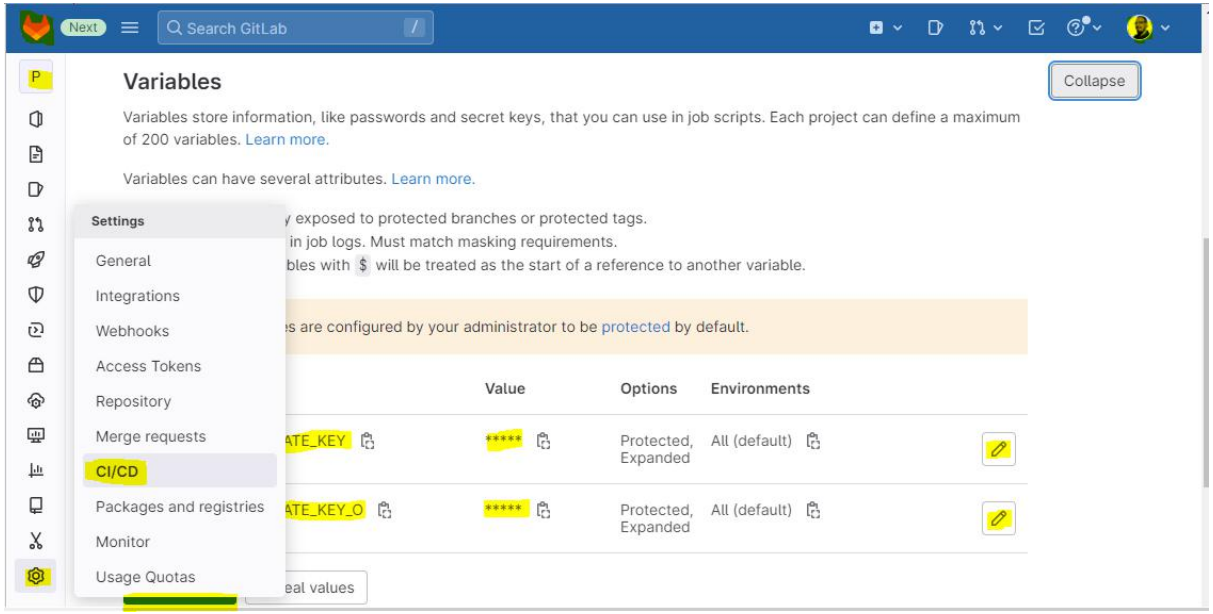
Registro sobre el servidor Ubuntu.

```
        $ cat > .ssh/id_rsa << EOF
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAyuhctMkzmCctJ4eLHF2zxFm58U5uWbNqRvCsTlmriEUDZX23Ii1G
oCRWmOwLSZ33zpSM26KBV0X/eFV6P3yuxkoQhtlfcDFvpyKOSVfgpN+dWRqdyKAPV2xQl2
d2tg+BYzdtAcAnFm99yRkJ4a1qNGy33JieO54JxjpIpfrQImNH8Ljyoe+g+egjBPMqn6Hw
klS9V/ne5Too6cw9DOFjCamGwyt1pgTvaoWGVUxCJqhZ7oQmdWcV3yl9Z2vc0K0T7wPgdv
TWLrswwVAJqaeP/0CAYH6reOHZYo/X9ed6XjdMXE+M33yzmEizKLJs64FewuwUmBxOC38m
zkc/qC5OUJHlESceO4/IVxGDS/nRUBOsrbzXpjPaF36DXZjaaTqVAz1fdkz2V4LL3l99JU
-----END OPENSSH PRIVATE KEY-----
EOF
```

La siguiente sentencia imprime dentro del archivo de configuración la llave privada con cifrado de RSA sobre el servidor. Dicha llave se requiere como bastión y secreto en el acceso a la comunicación segura por consola desde el corredor hacia el sitio WEB de GitLab. Finalizando a la salida del archivo de configuración.

Configuración sobre la consola WEB de GitLab.

URL: CI/CD Settings · Proyecto / Repositorio · GitLab



La siguiente URL nos permite acceder a editar la configuración de las variables CI/CD dentro de GitLab correspondiente al actual repositorio del proyecto. Aquí podemos agregar tanto variables globales protegidas, como secretos de valores sensibles, como la llave privada para los valores, como secretos requeridos dentro del guion del trabajo para continuar con la correcta autenticación desde el corredor dentro del servidor.

```
      - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add - > /dev/null
      - ssh -T git@gitlab.com
      - mkdir -p ~/.ssh
      - chmod 700 ~/.ssh
      - ssh-keyscan "$CI_SERVER_HOST" >> ~/.ssh/known_hosts
      - chmod 644 ~/.ssh/known_hosts
```

Las siguientes sentencias evalúan las credenciales que requerimos para autenticarnos dentro de la canalización, para uso de la conexión segura por consola desde el corredor.

```
      - git config --global user.name "${GITLAB_USER_NAME}"
      - git config --global user.email "${GITLAB_USER_EMAIL}"
```

Las siguientes sentencias evalúan las credenciales que requerimos para autenticarnos dentro de la canalización, para uso de la cuenta actual de GIT desde el corredor.

```
      - url_host=$(echo "${CI_REPOSITORY_URL}" | sed -e 's|https\?://gitlab-ci-token:.*@|ssh://git@|g')
```

La siguiente sentencia establece una variable local dentro del corredor que pueda hacer uso de la clonación del repositorio desde GitLab por medio de una cuenta por conexión segura, mas no desde una web con certificado.

```
    script:
     - git checkout main
```

La siguiente sentencia establece que sea para este caso la rama principal. Quien recibirá los cambios.

```
    - git config remote.origin.fetch "+refs/heads/*:refs/remotes/origin/*"
    - git fetch origin
    - git config pull.rebase false
    - git stash; git fetch origin; git reset --hard origin/main
```

Las siguientes sentencias restablecen los cambios pendientes. Con ello poder continuar con la subida de cambios.

```
    - git pull origin main
    - git submodule update --remote
```

Las siguientes sentencias actualizan los cambios globales dentro del repositorio local del corredor. Con ello poder continuar con los nuevos en los submódulos del mismo.

```
    - echo 1 >>Update.txt
```

La siguiente sentencia crea un cambio local dentro del corredor como pivote ante un registro. Con ello poder subir un nuevo cambio.

```
    - git add -A
    - git commit -m "$COMMIT"
```

Las siguientes sentencias confirman los nuevos cambios locales y de los submódulos dentro del corredor.

```
    - git push ${url host} -o ci.skip
```

La siguiente sentencia sube los cambios al repositorio global apuntando a la url por conexión segura, estableciendo una salida CI que no desencadene canalizaciones luego de subir los mismos.

7. Continuamos con el build de la Integración continua, creando un pull request desde ambientes bajos hacia el entorno de producción.

Código Fuente de Pipeline para Proyectos desde la rama Staging del repositorio: .gitlab-ci.yml

```
build-job1:
    stage: Build
    tags:
        - it
    script:
        - echo $TEST_VAR
        - mkdir -p /home/gitlab-runner/.ssh
    only:
        - staging
```

Creamos una nueva rama a partir de la rama Main o Master para entornos de QA desde Test y Dev, donde se crea trabajo siguiendo el branching strategy de GitLab – Flow.

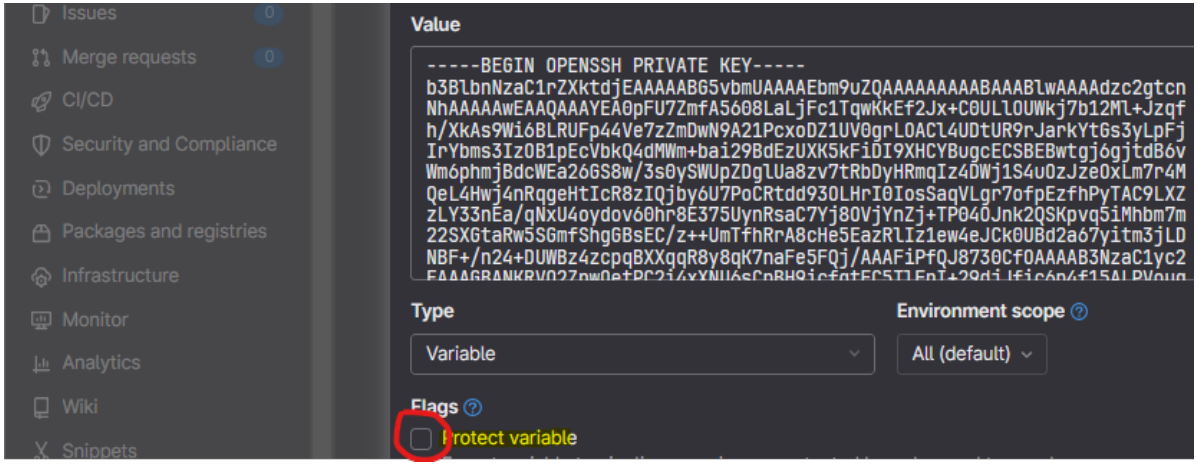
- Acceso por SSH en el Runner hacia GitLab desde el servidor Ubuntu desde la rama Staging.

URL: CI/CD Settings · Proyecto / Variables · GitLab

```
20 $ echo "$SSH_UBUNTU_KEY" | tr -d '\r' | ssh-add - > /dev/null
21 Error loading key "(stdin)": error in libcrypto
```

```
70 $ git push ${url_host} "$BRANCH" -o ci.skip
71 Host key verification failed.
72 fatal: Could not read from remote repository.
73 Please make sure you have the correct access rights
74 and the repository exists.
76 Cleaning up project directory and file based variables
78 ERROR: Job failed: exit status 1
```

NO se debe proteger la variable de la llave publica ante ramas que no son protegidas, ni se han estipulado por defecto.



8. Comienza el reléase con el scripting de deploy de configuraciones desde Capistrano (Gemas – Ruby on rails) hacia los contenedores por Docker**,** donde se va a clonar el repositorio del PR hacia el Merge Request, finalizando la canalización.

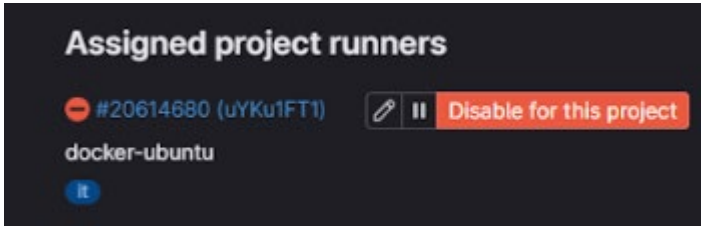
Troubleshooting.

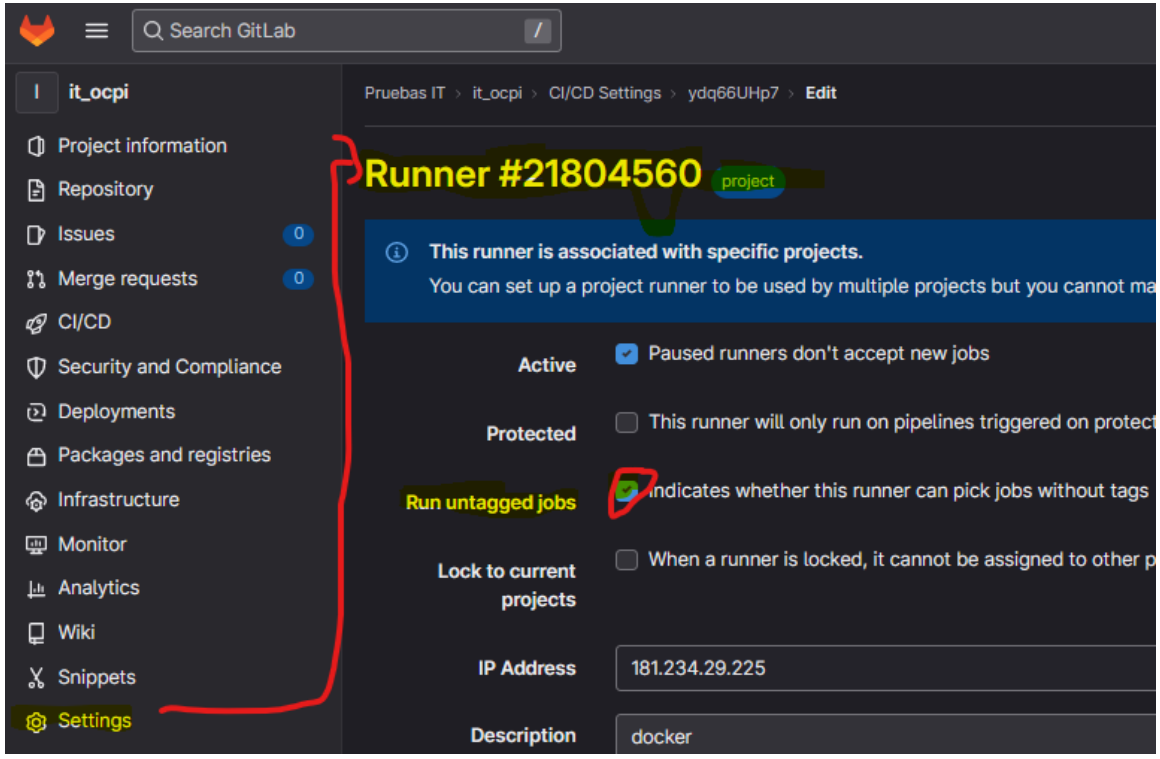
Muchos errores dentro de la canalización se producen por **omitir los tokens de acceso, igual no haber seteado dentro de las variables de CI/CD los secretos de claves sensibles a terceros o no estar activo el corredor.** Los errores con comandos BASH **se deben evidenciar que existan como binarios del sistema para el corredor de consola**. Igual los errores con la API de GIT se deben por **no hacer un pull limpio mediante stash o fetch, igual un push no apuntado al proyecto con carga de nuevos cambios a enviar, como no especificar la salida CI para evitar bucles infinitos.** Para errores en una etapa o trabajo, **establecer bien los flujos y reglas con condicionales a validar.**

Para hacer uso de canalizaciones en la plataforma GitLab con corredores libres, a través de WSL se requiere **usar una versión de un subsistema en Ubuntu no superior a la versión relanzada de distro Linux en kernel "18.04"**. Debido a que **las demás versiones generan un conflicto a la hora de instalar el paquete de conexión segura por consola.**

Desde Ubuntu **no se recomienda ejecutar el parámetro de correr sobre el corredor de GitLab**. Debido a que, al ejecutarse en segundo plano, entrega la salida activa, la cual queda impresa por monitoreo y no permite avanzar en la sentencia de nuevos comandos. **Para ello debemos hacer uso de una nueva terminal, la cual esta conexión se sumará al consumo de memoria RAM y procesamiento en simultáneo con el corredor activo.**

Desde un contenedor Docker, al registrar un corredor, **se debe habilitar manualmente el permiso de correr cualquier pipeline sin etiqueta desde GitLab**. Debido a que, al registrarse se establece un cache que no levanta el cambio en la plataforma de manera automática, pasiva o activa. **Para ello debemos hacer uso de este paso hasta que se vea reflejado la activación del mismo.**





Deuda Técnica.

Desde GitLab, ya **no se permite registrar ningún corredor on premise o privado hacia la plataforma CI**. Debido a que, al lanzar la nueva release “17.0”, por temas de seguridad, agilidad y administración se omite ese túnel dentro de la integración continua. **Para ello debemos hacer uso de herramientas terceras Open Source (Travis CI, Azure DevOps, etc.) o continuar con la versión pro de GitLab CI, para el uso de los runners públicos dentro de la capa gratuita y costos consumidos.**

Se evidencia sí con una versión legada o deprecada se lograba continuar con el servicio, pero la API publica ya no reconoce los motores que suministran los intérpretes dentro de los runners.