GITLAB CI/CD

L.I. Hugo Cuéllar Martínez

Dirección General de Cómputo y de Tecnologías de Información y Comunicación, UNAM

hugo.cuellar@comunidad.unam.mx

GitLab CI/CD pasa sus construcciones en una cosa que se llama GitLab Runners. Estos Runners son máquinas virtuales aisladas que corren pasos predefinidos a través de la API, GitLab CI API. Esta herramienta independiente, permite que los proyectos circulan a través de las construcciones de pipelines más rápido, comparada a cuando se corren en una sola instancia.

Entorno:

- 4cpu, 15gb de ram.
- Sistema Operativo: CentOS Linux 7.
- GitLab Community Edition 14.0.1.
- Docker 19.03.12.

GitLab Runner

GitLab Runner es el proyecto de código abierto usado para correr trabajos y mandar los resultados de regreso a GiLab. Son usados en conjunción con GitLab CI, el servicio de código abierto de integración continúa incluido en GiLab el cual coordina los trabajos.

Crear Runners

Requisitos para crear los Runners.

Para crear un nuevo Runner es necesario tener instalado en nuestro servidor donde esta GitLab.

- GITLab Runner: https://docs.gitlab.com/runner/install/ esta herramienta es la que nos va a permitir ejecutar.

- Docker si se va a ocupar contendores en nuestro pipeline. Pudiendo instalar las herramientas docker-compose y docker-machine (En este caso solo está instalado Docker en el servidor). Y el usuario gitlab-runner debe tener permisos para poder crear, eliminar y listar los contenedores de Docker.
- Si se va a copiar o necesitamos acceder a otro servidor debemos configurar las llaves ssh para poder comunicarse entre los servidores.

Una vez instalado revisamos que el comando gitlab-runner esté disponible como se muestra a continuación.

```
[gitlab-runner@mercurio ~]$ gitlab-runner
Runtime platform                                    arch=amd64 os=linux pid=8823 revision=738bbe5a version=13.3.1
NAME:
   gitlab-runner - a GitLab Runner

USAGE:
   gitlab-runner [global options] command [command options] [arguments...]

VERSION:
   13.3.1 (738bbe5a)

AUTHOR:
   GitLab Inc. <support@gitlab.com>

COMMANDS:
     exec                execute a build locally
     list                List all configured runners
     run                 run multi runner service
     register            register a new runner
     install             install service
     uninstall           uninstall service
     start               start service
     stop                stop service
     restart             restart service
     status              get status of a service
     run-single          start single runner
     unregister          unregister specific runner
     verify              verify all registered runners
     artifacts-downloader  download and extract build artifacts (internal)
     artifacts-uploader  create and upload build artifacts (internal)
     cache-archiver      create and upload cache artifacts (internal)
     cache-extractor     download and extract cache artifacts (internal)
     cache-init          changed permissions for cache paths (internal)
     health-check        check health for a specific address
     read-logs           reads job logs from a file, used by kubernetes executor (internal)
     help, h             Shows a list of commands or help for one command
```

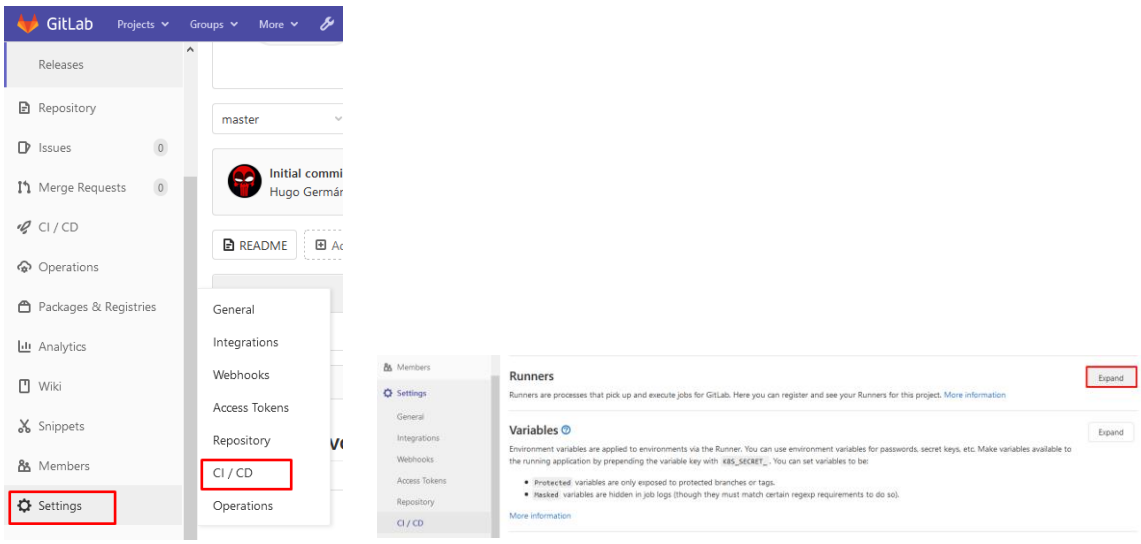
Creación de un repositorio y configuración de Runners

Hay dos tipos de Runners, los específicos y los compartidos.

Runners Específicos:

Son para usarse en un solo proyecto y se crean de la siguiente manera:

Creamos o seleccionamos un repositorio en el menú de «Settings» –> «CI/CD» –> «Runner Settings» en esta parte podemos elegir si crear un nuevo Runner para nuestro proyecto.



Debemos ir a la sección «Specific Runners» vamos a encontrar la URL y el token para registrar los nuevos Runners asociados al repositorio que son necesarios para poder ejecutar las distintas tareas que definamos en el pipeline.

Una vez que tenemos localizada esta información. Vamos a acceder al terminal de la máquina donde esté corriendo GitLab.

Ahora para registrar nuestro primer Runner vamos a ejecutar:

```
[gitlab-runner@mercurio ~]$ gitlab-runner register
```

Este comando nos va a lanzar una serie de preguntas:

- **Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com)**: contestamos con la URL que nos proporcionó el paso anterior (http://mercurio.dcaa.unam.mx/).
- **Please enter the gitlab-ci token for this runner**: contestamos con el token que nos proporcionó el paso anterior (sSxC…tj8x).
- **Please enter the gitlab-ci description for this runner**: es la descripción del Runner. Se recomienda poner el tipo de runner.
- **Please enter the gitlab-ci tags for this runner (comma separated)**: le asociamos un tag, el cual se va a utilizar para determinar el runner en los jobs. Por ejemplo, Docker, Shell, sync ; podría ser cualquier otro descriptivo.

- **Please enter the executor: ssh, docker+machine, docker-ssh+machine, kubernetes, docker, parallels, virtualbox, docker-ssh, shell**: seleccionamos el tipo de runner.
- **Please enter the Docker image (eg. ruby:2.1)**: si elegimos Docker nos pide la imagen base del contenedor ejemplo alpine:latest

El Runner tipo Docker nos va a permitir definir imágenes de docker que serán las encargadas de ejecutar los distintos comandos en su interior. Esto tiene la ventaja de que no necesitamos instalar en la máquina ningún runtime de Java, Maven o NodeJS, solo tenemos que utilizar las imágenes apropiadas a cada tarea.

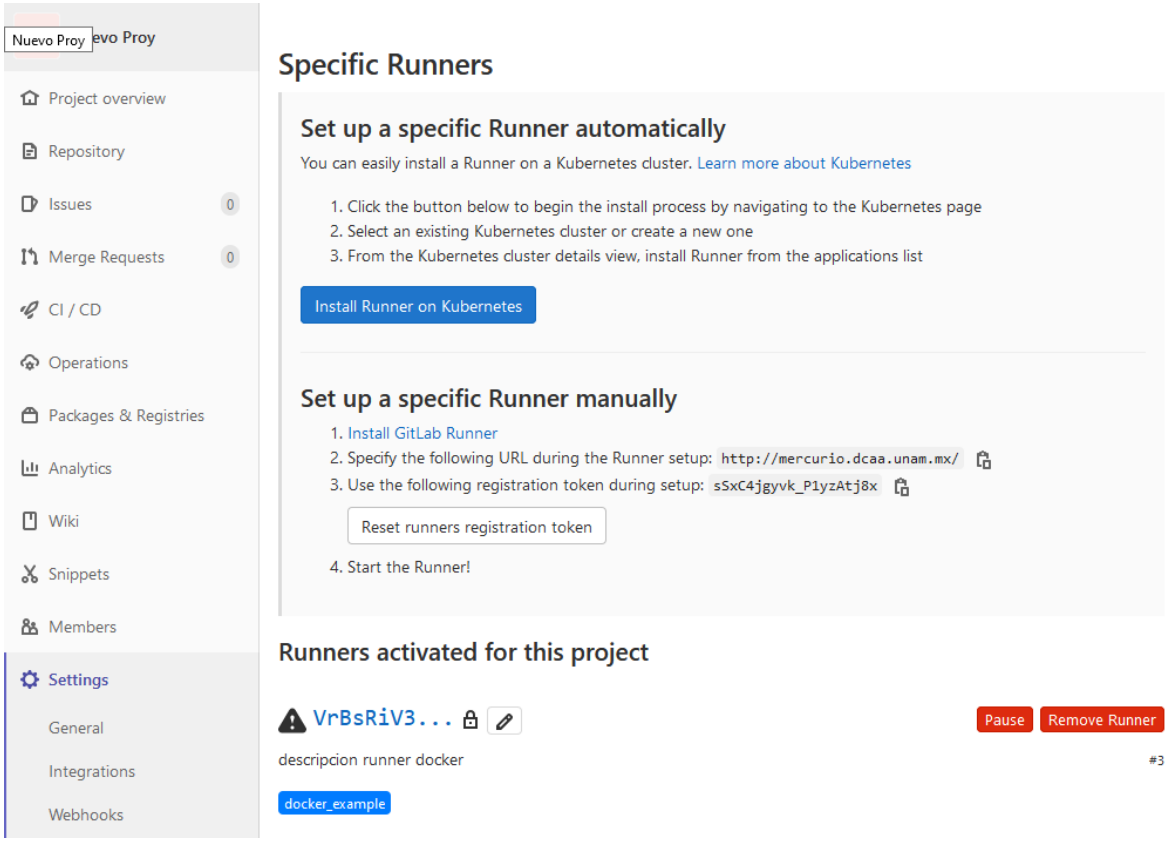
El Runner de tipo Shell nos va a dar la flexibilidad de poder ejecutar cualquier tipo de tarea que pudiéramos ejecutar en la shell de la máquina, por ejemplo, ejecutar docker, docker-compose, o comandos como wget, etc.

Podemos registrar tantos runners como queramos.

```
[gitlab-runner@mercurio ~]$ gitlab-runner register
Runtime platform                                    arch=amd64 os=linux pid=9328 revision=738bbe5a version=13.3.1
WARNING: Running in user-mode.
WARNING: The user-mode requires you to manually start builds processing:
WARNING: $ gitlab-runner run
WARNING: Use sudo for system-mode:
WARNING: $ sudo gitlab-runner...

Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
http://mercurio.dcaa.unam.mx/
Please enter the gitlab-ci token for this runner:
sSxC4jgyvk_P1yzAtj8x
Please enter the gitlab-ci description for this runner:
[mercurio.dcaa.unam.mx]: descripcion runner docker
Please enter the gitlab-ci tags for this runner (comma separated):
docker_example
Registering runner... succeeded                     runner=sSxC4jgy
Please enter the executor: docker, parallels, docker-ssh+machine, custom, docker-ssh, shell, ssh, virtualbox, docker+machine, kubernetes:
docker
Please enter the default Docker image (e.g. ruby:2.6):
alpine:latest
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!
```

Si todo es correcto, al refrescar la página, veremos que aparece una lista con los Runners instalados.

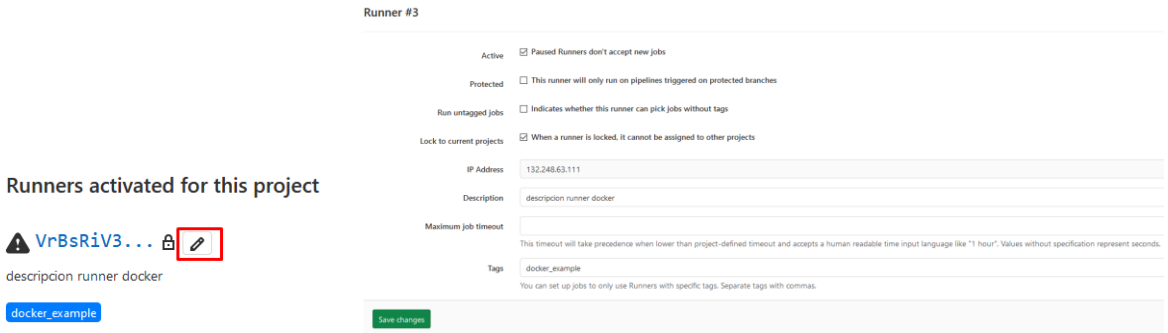


Runners Compartidos:

Para utilizar un Runner compartido en el proyecto podemos ir a la sección << Available specific Runners >> ahí se mostrarán todos los Runners compartidos que están disponibles, para nuestro proyecto.

Activar y configurar Runners

Si queremos activar o configurar el Runner podemos acceder desde el botón de editar



Ahí se podrán editar y activar otras configuraciones como

- **Active:** si el Runner estará en pausa o activo.
- **Protected:** solo se ejecutará en ramas protegidas.
- **Run untagged Jobs:** Se ejecuta en trabajos sin etiquetas
- **Lock to current projects:** Bloquea para que no se puede asignar a otros proyectos.
- **IP Address:** Es la direccion IP donde se va a ejecutar el Runner.
- **Description:** Descripcion del Runner.
- **Maximum job timeout:** Tiempo máximo de espera para ejecutar el Runner.
- **Tags:** Etiquetas específicas para ejecutar el Runner separadas por coma (,).

Dentro del archivo /etc/gitlab-runner/config.toml podemos definir otros parámetros de configuración más avanzados para cada uno de los Runners, aquí se encuentran todos los posibles:
https://docs.gitlab.com/runner/configuration/advanced-configuration.html

Teniendo ya instalado y habilitado el Runner ahora el siguiente paso es crear nuestro archivo para realizar las tareas automatizadas.

Configurar CI/CD en nuestro proyecto

Podemos acceder a la configuración de CI/CD de nuestro proyecto en el menú de Settings -> CI/CD
Ahí podemos revisar la estrategia para obtener el repositorio (Git Strategy):

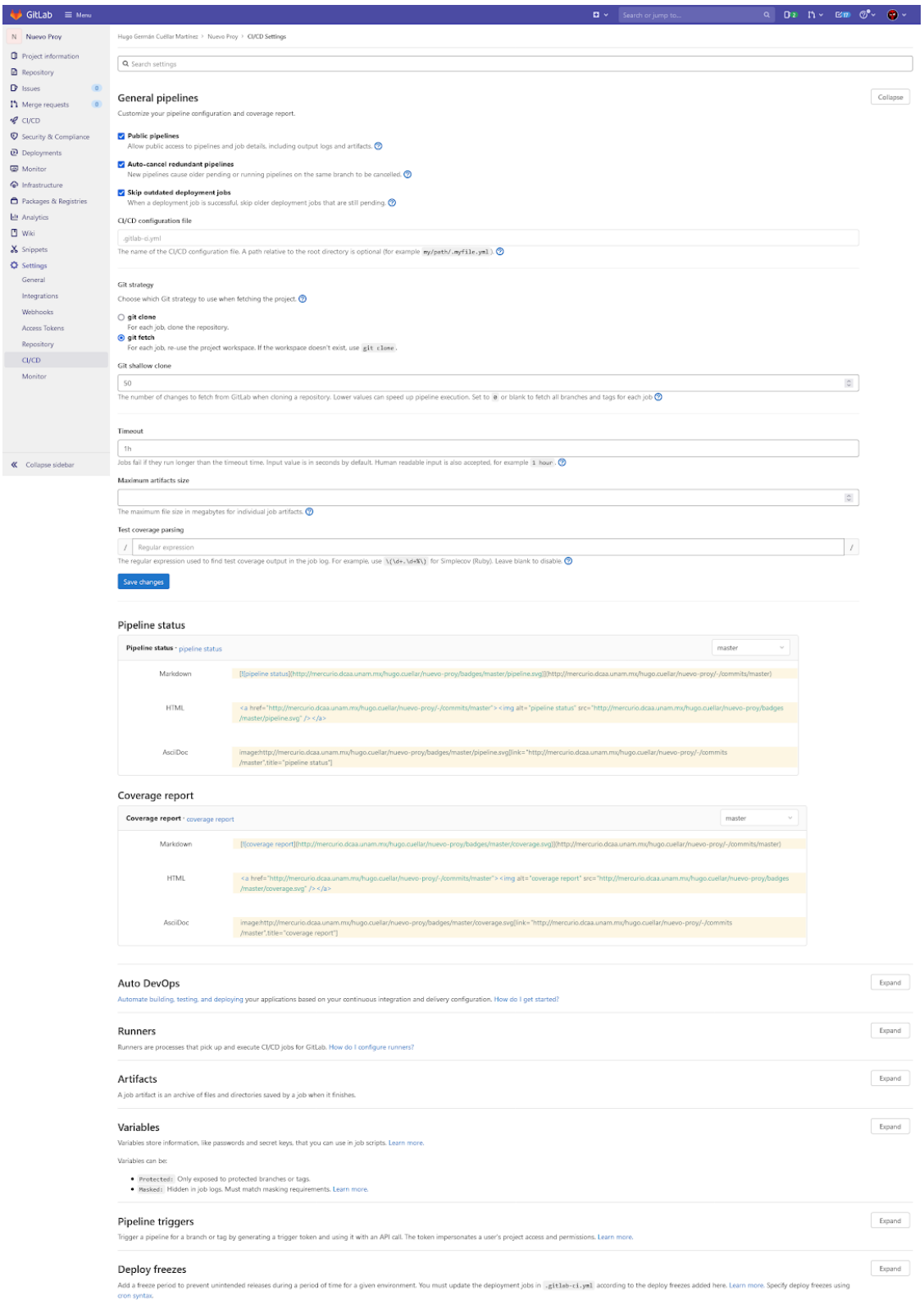
Git Clone: Clona el repositorio desde cero para cada trabajo, asegurándose de que la copia de trabajo local sea siempre impecable; más lento.

Git Fetch: (predeterminado) Reutiliza la copia de trabajo local (clona si no existe). Esto se recomienda, especialmente para repositorios grandes. Es rápido, pero en ocasiones cuando se ejecuta el trabajo manda errores.

También se puede especificar el directorio y nombre del archivo de configuración

de CI/CD en nuestro proyecto por default se encuentra en la raíz con nombre

“.gitlab-ci.yml” lo único que no se puede cambiar es la extensión yml.


GitLab
Menu
Search or jump to...
Nuevo Proy
Project information
Repository
Issues
Merge requests
CI/CD
Security & Compliance
Deployments
Monitor
Infrastructure
Packages & Registries
Analytics
Wiki
Snippets
Settings
General
Integrations
Webhooks
Access Tokens
Repository
CI/CD
Monitor
Collapse sidebar
Hugo Germán Cuéllar Martínez > Nuevo Proy > CI/CD Settings
Search settings
General pipelines
Collapse
Customize your pipeline configuration and coverage report.
Public pipelines
Allow public access to pipelines and job details, including output logs and artifacts.
Auto-cancel redundant pipelines
New pipelines cause older pending or running pipelines on the same branch to be cancelled.
Skip outdated deployment jobs
When a deployment job is successful, skip older deployment jobs that are still pending.
CI/CD configuration file
.gitlab-ci.yml
The name of the CI/CD configuration file. A path relative to the root directory is optional (for example my/path/.myfile.yml).
Git strategy
Choose which Git strategy to use when fetching the project.
git clone
For each job, clone the repository.
git fetch
For each job, re-use the project workspace. If the workspace doesn't exist, use git clone .
Git shallow clone
50
The number of changes to fetch from GitLab when cloning a repository. Lower values can speed up pipeline execution. Set to 0 or blank to fetch all branches and tags for each job
Timeout
1h
Jobs fail if they run longer than the timeout time. Input value is in seconds by default. Human readable input is also accepted, for example 1 hour.
Maximum artifacts size
The maximum file size in megabytes for individual job artifacts.
Test coverage parsing
Regular expression
The regular expression used to find test coverage output in the job log. For example, use \(\d+.\d+%\) for Simplecov (Ruby). Leave blank to disable.
Save changes
Pipeline status
Pipeline status · pipeline status
master
Markdown
[![pipeline status](http://mercurio.dcaa.unam.mx/hugo.cuellar/nuevo-proy/badges/master/pipeline.svg)](http://mercurio.dcaa.unam.mx/hugo.cuellar/nuevo-proy/-/commits/master)
HTML

AsciiDoc
image:http://mercurio.dcaa.unam.mx/hugo.cuellar/nuevo-proy/badges/master/pipeline.svg[link="http://mercurio.dcaa.unam.mx/hugo.cuellar/nuevo-proy/-/commits /master",title="pipeline status"]
Coverage report
Coverage report · coverage report
master
Markdown
[![coverage report](http://mercurio.dcaa.unam.mx/hugo.cuellar/nuevo-proy/badges/master/coverage.svg)](http://mercurio.dcaa.unam.mx/hugo.cuellar/nuevo-proy/-/commits/master)
HTML

AsciiDoc
image:http://mercurio.dcaa.unam.mx/hugo.cuellar/nuevo-proy/badges/master/coverage.svg[link="http://mercurio.dcaa.unam.mx/hugo.cuellar/nuevo-proy/-/commits /master",title="coverage report"]
Auto DevOps
Expand
Automate building, testing, and deploying your applications based on your continuous integration and delivery configuration. How do I get started?
Runners
Expand
Runners are processes that pick up and execute CI/CD jobs for GitLab. How do I configure runners?
Artifacts
Expand
A job artifact is an archive of files and directories saved by a job when it finishes.
Variables
Expand
Variables store information, like passwords and secret keys, that you can use in job scripts. Learn more.
Variables can be:
Protected: Only exposed to protected branches or tags.
Masked: Hidden in job logs. Must match masking requirements. Learn more.
Pipeline triggers
Expand
Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates a user's project access and permissions. Learn more.
Deploy freezes
Expand
Add a freeze period to prevent unintended releases during a period of time for a given environment. You must update the deployment jobs in .gitlab-ci.yml according to the deploy freezes added here. Learn more. Specify deploy freezes using cron syntax.

Para usar el CI/CD en GitLab necesitamos crear un archivo llamado ".gitlab-ci.yml" en la raíz del repositorio o como se haya especificado en el punto anterior, donde contenga las configuraciones a utilizar.

http://132.248.63.111/help/ci/pipelines/settings

En la siguiente liga se pueden observar las diferentes configuraciones del archivo.

https://docs.gitlab.com/14.0/ee/ci/yaml/README.html

Este archivo contiene:

- Los scripts que desea ejecutar
- Archivos de configuración y plantillas que se desean incluir
- Dependencias y cachés
- Comandos que se desean ejecutar
- Ubicaciones donde implementar la aplicación
- Si los scripts se ejecutan automática o manualmente

Palabras clave para definir el comportamiento:

Keyword	Description
after_script	Override a set of commands that are executed after job
allow_failure	Allow job to fail. A failed job does not cause the pipeline to fail
artifacts	List of files and directories to attach to a job on success
before_script	Override a set of commands that are executed before job
cache	List of files that should be cached between subsequent runs
coverage	Code coverage settings for a given job
dependencies	Restrict which artifacts are passed to a specific job by providing a list of jobs to fetch artifacts from
environment	Name of an environment to which the job deploys
except	Control when jobs are not created
extends	Configuration entries that this job inherits from
image	Use Docker images
include	Include external YAML files

Keyword	Description
inherit	Select which global defaults all jobs inherit
interruptible	Defines if a job can be canceled when made redundant by a newer run
needs	Execute jobs earlier than the stage ordering
only	Control when jobs are created
pages	Upload the result of a job to use with GitLab Pages
parallel	How many instances of a job should be run in parallel
release	Instructs the runner to generate a release object
resource_group	Limit job concurrency
retry	When and how many times a job can be auto-retried in case of a failure
rules	List of conditions to evaluate and determine selected attributes of a job, and whether or not it's created
script	Shell script that is executed by a runner
secrets	The CI/CD secrets the job needs
services	Use Docker services images
stage	Defines a job stage
tags	List of tags that are used to select a runner
timeout	Define a custom job-level timeout that takes precedence over the project-wide setting
trigger	Defines a downstream pipeline trigger
variables	Define job variables on a job level
when	When to run job

1. Para empezar a realizar nuestro archivo empezamos con la etiqueta stages. Esta etiqueta se utiliza para definir etapas que contienen grupos de trabajos.
   El orden de los stages define el orden de ejecución de los trabajos:
   - Los trabajos de la misma etapa se ejecutan en paralelo.
   - Los trabajos de la etapa siguiente se ejecutan después de que los trabajos de la etapa anterior se completen correctamente.

```
stages:
    - desarrollo
    - pruebas
    - produccion
```

2. Creamos el trabajo (job) que va a realizar si se ejecutan varios trabajos se necesita ponerle una etiqueta, en el ejemplo se crean las etiquetas deploy-desarrollo y deploy-pruebas.

```
deploy-desarrollo:

      instrucciones

deploy-pruebas:

      instrucciones
```

3. Ponemos nuestras instrucciones de que se va a realizar en nuestro trabajo aquí podemos decir en qué etapa se ejecuta, en qué rama va a realizar la ejecución, si es manual o automática, y todas las instrucciones que queramos realizar.

```
deploy-desarrollo:

  stage: desarrollo

  script:
        - echo "Imprimir algo en la consola de CI/CD"
        - ssh user@132.248.0.0 'ejecutar algo en una ip diferente'
        - ejecutar algo mas en el servidor
        - php -v
# si hay un cambio en esta rama se ejecutara este trabajo.
  only:
        - rama
```

Ejemplo de un GItlab CI/CD

```
stages:
    - desarrollo
    - produccion
deploy-desarrollo:
    stage: desarrollo
    script:
        - rsync -vv -rz --checksum . user@host:/home/user/public_html/
        - ssh user@host 'bash /home/user/ls -l'
    except:
        - master
deploy-produccion:
    stage: produccion
    script:
        - echo "Copiando al servidor de produccion"
        - rsync -vv -rz --checksum . user2@host-prod:/var/html/
        - ssh user2@host-prod '/var/html/composer install'
        - ssh user2@host-prod 'php /var/html/artisan config:cache'
    when: manual
    only:
        - master
```

En este ejemplo creamos dos etapas una de desarrollo y otra para producción:

- En la etapa de desarrollo se ejecutará una sincronización de carpetas en un servidor remoto con la instrucción rsync, también se ejecutará un listado de directorio en el servidor remoto con ssh, cuando detecte un cambio en cualquier rama que no sea master.
- En la etapa de producción se ejecutarán las siguientes instrucciones:
    - Una impresión a la consola Copiando al servidor de producción
    - La sincronización de carpetas a un servidor remoto
    - La ejecución del comando composer install en el servidor remoto en la ruta /var/html
    - La ejecución del comando php artisan config:cache en el servidor remoto en la ruta /var/html

  Solo se va a ejecutar cuando se detecte un cambio en la rama master y de forma manual se mostrará un botón de play para ejecutar.

Referencias

Chrissie Buchanan. (2019). A quick guide to GitLab CI/CD pipelines. 28/06/2021, de GitLab Inc Sitio web: https://about.gitlab.com/blog/2019/07/12/guide-to-ci-cd-pipelines/

Dov Hershkovitch. (2021). Meet Pipeline Editor, your one-stop shop for building a CI/CD pipeline. 28/06/2021, de GitLab Inc Sitio web: https://about.gitlab.com/blog/2021/02/22/pipeline-editor-overview/

GitLab. (2021). GitLab CI/CD. 28/06/2021, de GitLab Inc Sitio web: https://docs.gitlab.com/ee/ci/

CRÉDITOS

DGTIC-UNAM

Elaborado por: L.I. Hugo Germán Cuéllar Martínez

Revisión técnica: L.I. Karla Alejandra Fonseca Márquez

Revisión de estilo: Pamela Valdés Reséndiz

Autorización de publicación: Dra. Marcela J. Peñaloza Báez

Dirección de Colaboración y Vinculación, 2021