

-m venvApuntes de EGC:

Ejercicio A (GIT)

Intensificación colaborativa

1. Realice un fork de este repositorio con el nombre EGC2324-turno42-"uvus".

Práctica 1: Instalación del sistema base

Crear entorno virtual

```
python -m venv <myenvname>
```

Activar entorno virtual

```
source <myenvname>/bin/activate
```

Desactivar entorno virtual

```
deactivate
```

Git: instalar y configurar

```
sudo apt install git  
git config --global user.name "<tu nombre y apellidos entrecomillados>"  
git config --global user.email <tu email>
```

Git: generar par de claves SSH

```
ssh-keygen -t rsa -b 4096
```

Git: visualizar clave pública y copiar

```
cd ~/.ssh  
cat id_rsa.pub
```

Git: guardar clave pública en GitHub

GitHub -> Settings -> SSH and GPG keys -> New SSH key -> Pegar clave pública (id_rsa.pub)

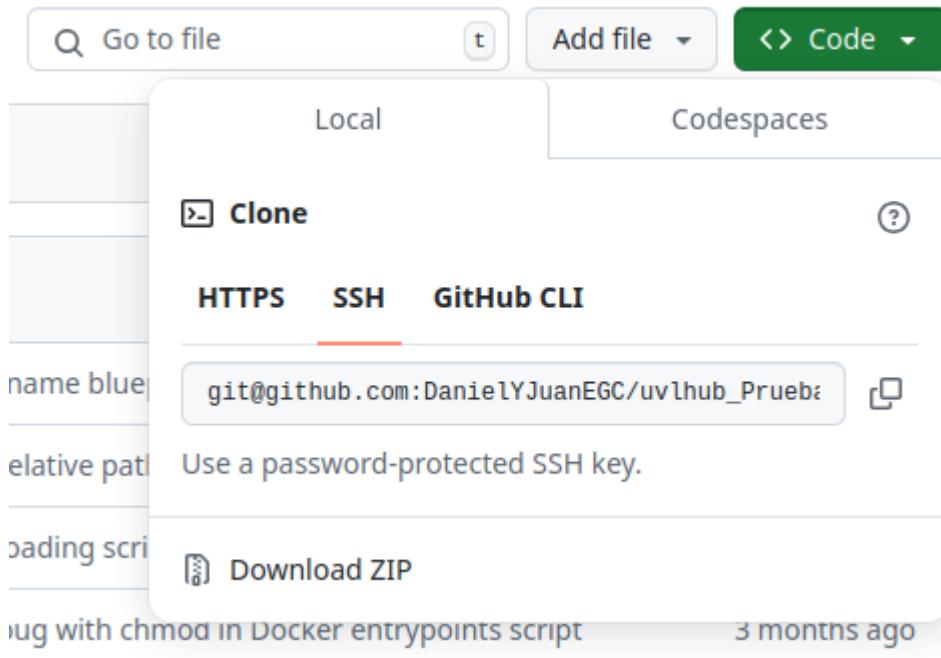
- Clone el repositorio del cual ha hecho el fork.

Clone the repo

ARE YOU A STUDENT OF CONFIGURATION EVOLUTION AND MANAGEMENT (EGC)?

Remember that you have to clone your fork from the subject fork instead of the official one.

```
git clone git@github.com:<YOUR_GITHUB_USER>/uvlhub_practicas.git  
cd uvlhub_practicas
```



- Cree una nueva rama llamada egc_test en el repositorio.
- "Salte" a la rama recien creada.

```
git checkout -b egc_test
```

Este comando crea y “salta” directamente a la rama nueva.

Para subir la rama al repositorio:

```
^C(venv) practica@EGC:~/Escritorio/prueba/uvlhub_prueba$ git push
fatal: The current branch egc test has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin egc_test

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.

(venv) practica@EGC:~/Escritorio/prueba/uvlhub_prueba$ git push --set-upstream origin egc_test
Total 0 (delta 0), reusados 0 (delta 0), pack-reusados 0
remote:
remote: Create a pull request for 'egc_test' on GitHub by visiting:
remote:     https://github.com/Juan-y-Daniel-EGC/uvlhub_prueba/pull/new/egc_test
remote:
To github.com:Juan-y-Daniel-EGC/uvlhub_prueba.git
 * [new branch]      egc_test -> egc_test
rama 'egc_test' configurada para rastrear 'origin/egc_test'.
```

5. En el código de DECIDE del repositorio existe un error. Identifique el error ejecutando en su máquina el código.

Antes de poder ejecutar el código debemos seguir el manual de instalación de la practica1, al llegar a las bases de datos, eliminar si existen las bases de datos y usuario:

sql

 Copiar código

```
DROP DATABASE uvlhubdb;
DROP DATABASE uvlhubdb_test;
```

Para eliminar el usuario:

sql

 Copiar código

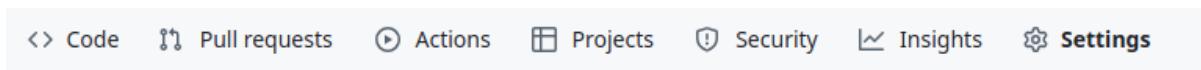
```
DROP USER 'uvlhubdb_user'@'localhost';
```

Después seguir la guía por:

```
CREATE DATABASE uvlhubdb;
CREATE DATABASE uvlhubdb_test;
CREATE USER 'uvlhubdb_user'@'localhost' IDENTIFIED BY 'uvlhubdb_password';
GRANT ALL PRIVILEGES ON uvlhubdb.* TO 'uvlhubdb_user'@'localhost';
GRANT ALL PRIVILEGES ON uvlhubdb_test.* TO 'uvlhubdb_user'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

6. Cree una "issue" en el fork del repositorio para reportar el error según lo visto en clase. 📸

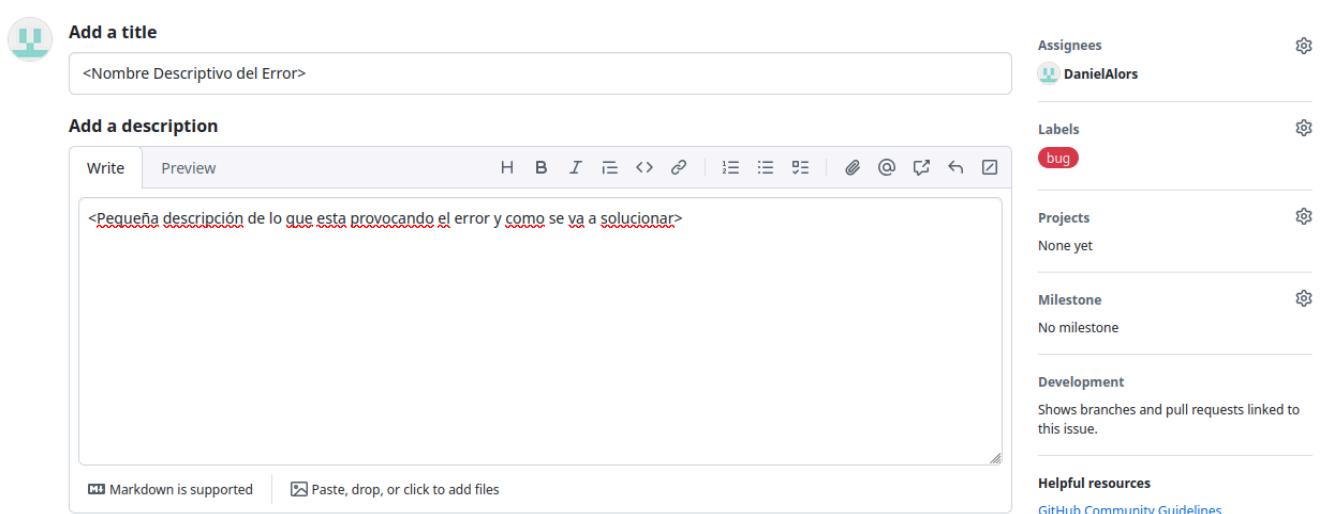
Si no salen directamente las issues en el repositorio hay que activarlas:



The screenshot shows the GitHub repository settings page. The top navigation bar includes links for Code, Pull requests, Actions, Projects, Security, Insights, and Settings. The Settings link is underlined, indicating it is active. Below the navigation is a tab bar with General selected. The main content area is titled 'Features' and contains three sections: Wikis, Issues, and Sponsorships. The Issues section is expanded, showing a sub-section titled 'Get organized with issue templates' with a 'Set up templates' button. The Issues section is currently checked.

- Wikis**
Wikis host documentation for your repository.
- Issues**
Issues integrate lightweight task tracking into your repository. Keep projects on track with issue labels and milestones, and reference them in commit messages.
- Sponsorships**
Sponsorships help your community know how to financially support this repository.

En Issue:



The screenshot shows a new GitHub issue creation form. It includes fields for 'Add a title' (placeholder: <Nombre Descriptivo del Error>) and 'Add a description' (placeholder: <Pequeña descripción de lo que esta provocando el error y como se va a solucionar>). On the right side, there are sections for 'Assignees' (DanielAlors), 'Labels' (bug), 'Projects' (None yet), 'Milestone' (No milestone), 'Development' (Shows branches and pull requests linked to this issue), and 'Helpful resources' (GitHub Community Guidelines). At the bottom, it says 'Markdown is supported' and 'Paste, drop, or click to add files'.

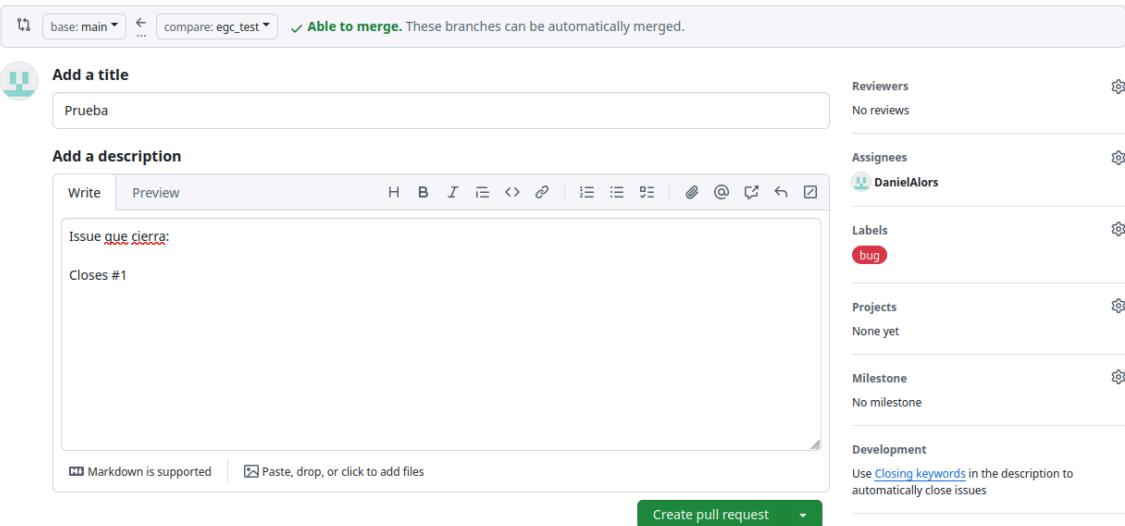
7. Realice las modificaciones necesarias para corregir el error y haga commit de los cambios en la rama egc_test.

Corregir el error correspondiente.

8. Mediante una pull request, fusione en la rama master/main del repositorio los cambios de la rama de egc_test y asocielo a la issue anterior. 

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. [Learn more about diff comparisons here.](#)



The screenshot shows the GitHub interface for creating a pull request. At the top, it says "Able to merge. These branches can be automatically merged." Below that, there are fields for "Add a title" (containing "Prueba") and "Add a description". The description text area contains "Issue que cierra:" and "Closes #1". To the right, there are sections for "Reviewers" (no reviews), "Assignees" (DanielAlors), "Labels" (bug), "Projects" (None yet), "Milestone" (No milestone), and "Development" (instructions to use closing keywords). A "Create pull request" button is at the bottom.

9. Refleje los cambios del repositorio local en el repositorio remoto que creó en el primer paso.

Comandos:

3. Ramas

Comandos básicos

Crear rama

```
git branch <nombre_rama>
```

Saltar a la rama

```
git checkout <nombre_rama>
```

Crear y saltar a la rama (en un solo comando)

```
git checkout -b <nombre_rama>
```



3. Ramas

Comandos básicos

-u = upstream = referencia de seguimiento, es la conexión entre la rama local y la rama remota

Subir rama a repositorio remoto

```
git push -u origin <nombre_rama>
```

Listar ramas locales

```
git branch
```

Listar ramas remotas

```
git branch -r
```

Listar ramas locales y remotas

```
git branch -a
```

3. Ramas

Comandos básicos

Eliminar rama local (si la rama ya ha sido fusionada con otra)

```
git branch -d <nombre_rama>
```

Eliminar rama local (forzar eliminación)

```
git branch -D <nombre_rama>
```

Eliminar rama remota

```
git push origin --delete <nombre_rama>
```

2.9) Crea un commit de nombre “feat: Añade archivo de prueba” pero no lo subas aún al repositorio remoto.

Solución -> `git commit -m "feat: Añade archivo de prueba"`

2.10) ¡Ay! Que nos hemos equivocado, que todo debería ir en inglés. Cambia, entonces, el título del commit por “feat: Add testing file” y cerrando la issue de tu compañero “Testing file (B)” mediante su ID

Solución ->

```
git commit --amend -m "feat: Add testing file. Closes  
#<ID>"
```

4.10) Intenta realizar la fusión de la rama remota con la rama local sin hacer rebase.

Solución ->

```
git pull --no-rebase origin feature/practicandogit_b
```

5.4) Actualiza los cambios remotos y obtén el hash del commit de tu compañero, es decir, “fix: Fix important bug (developer A)”

Solución ->

```
git fetch  
git log --oneline --all (hash del compañero/a B)
```

5.5) Tráete ese commit a tu rama

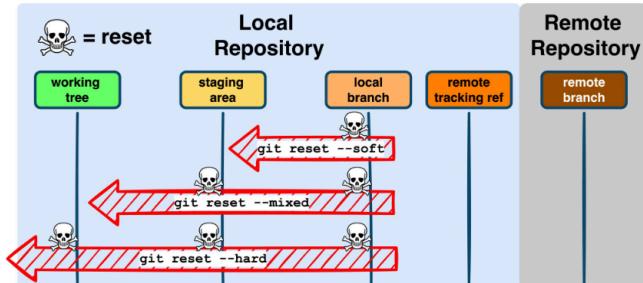
Solución -> `git cherry-pick (hash)`

Usando **git log --oneline --all** obtenemos el hash del cambio que queremos traer y usando la letra “q” salimos sin matar la terminal.

Aplicamos **git cherry-pick (hash obtenido del log)** para traer el cambio sin fusionar toda la rama.

4. Conceptos avanzados de Git

Reseteo



¿ git reset --hard HEAD ?

git reset --soft

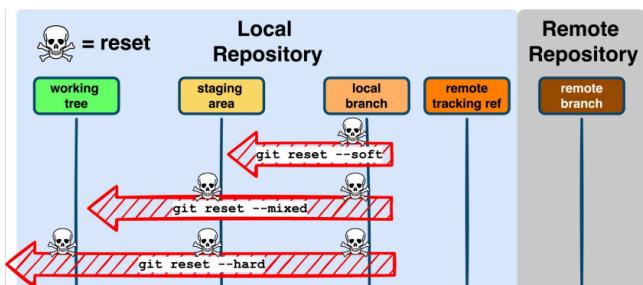
move la rama local al commit anterior (Staging y Working Tree sin cambios)

git reset --mixed (por defecto)

move la rama local al commit anterior y quita los cambios del Staging (Working Tree sin cambios)

git reset --hard

move la rama local al commit anterior y borra los cambios tanto del Staging como del Working Tree



¿ git reset --hard HEAD ?

git reset --soft

move la rama local al commit anterior (Staging y Working Tree sin cambios)

git reset --mixed (por defecto)

move la rama local al commit anterior y quita los cambios del Staging (Working Tree sin cambios)

git reset --hard

move la rama local al commit anterior y borra los cambios tanto del Staging como del Working Tree

TAGS:

Asegúrate de tener un repositorio Git configurado en tu máquina y de que estés en la rama correcta:

git checkout <nombre-de-la-rama>

Crea la Tag:

Para una etiqueta ligera (sin mensaje):

git tag <nombre-de-la-tag>

- Para una etiqueta anotada (recomendado, incluye más información como mensaje y autor):
git tag -a <nombre-de-la-tag> -m "Mensaje descriptivo de la versión"
- Verifica que la Tag fue creada:
git tag

Sube la Tag al repositorio remoto:
git push origin <nombre-de-la-tag>

- Si quieres subir todas las tags locales de una vez:
git push origin --tags

Ejercicio B (GITHUB ACTIONS)

Para usar más de una versión:

```
matrix:
    python-version: ['3.11', '3.12']

Ej:

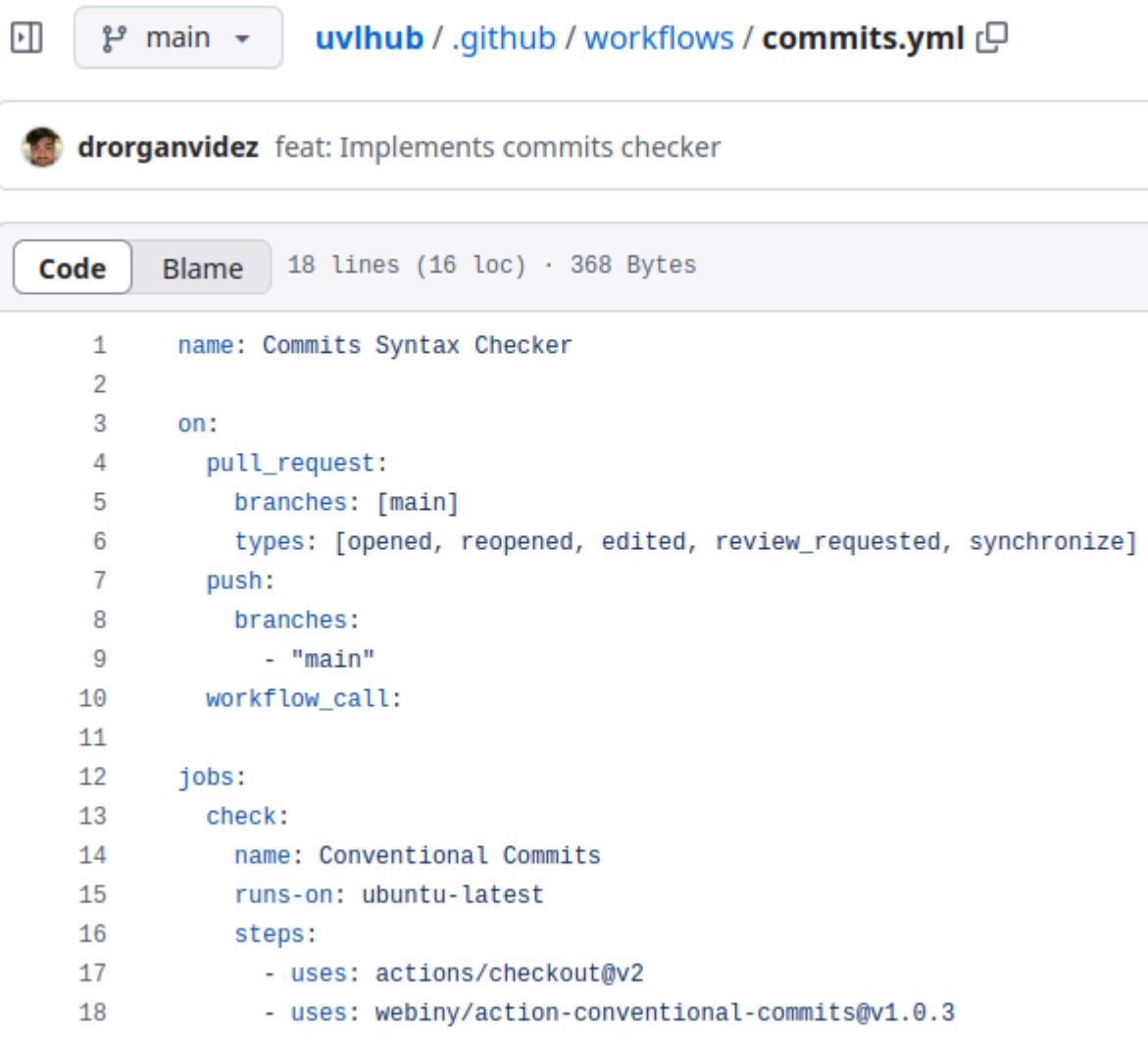
name: Codacy CI
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
jobs:
  build:
    runs-on: ubuntu-latest
    services:
      mysql:
        image: mysql:5.7
        env:
          MYSQL_ROOT_PASSWORD: uvhub_root_password
          MYSQL_DATABASE: uvhubdb_test
          MYSQL_USER: uvhub_user
          MYSQL_PASSWORD: uvhub_password
        ports:
          - 3306:3306
        options: --health-cmd="mysqladmin ping" --health-interval=10s --health-timeout=5s --health-retries=3
    strategy:
      matrix:
        python-version: ['3.11', '3.12']
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Check for outdated dependencies
        run: |
          pip list --outdated
```

EGC Wiki solución ejercicio de Codacy Práctica2

Cree un archivo travis.yml para pasar las pruebas exclusivamente del modulo del modulo store.

```
PHP INSTALLED REQUIREMENTS.YML  
- name: Run Tests  
  env:  
    FLASK_ENV: testing  
    MARIADB_HOSTNAME: 127.0.0.1  
    MARIADB_PORT: 3306  
    MARIADB_TEST_DATABASE: uvhubdb_test  
    MARIADB_USER: uvhub_user  
    MARIADB_PASSWORD: uvhub_password  
  run: |  
    pytest app/modules/ --ignore-glob='*selenium*'  
  
deploy:  
  name: Deploy to Render
```

Se añadiría a la ruta de “pytest” el módulo específico que queremos ejecutar, por ejemplo, para el módulo store, sería: “app/modules/store/”.



drorganvidez feat: Implements commits checker

Code Blame 18 lines (16 loc) · 368 Bytes

```
1 name: Commits Syntax Checker
2
3 on:
4   pull_request:
5     branches: [main]
6     types: [opened, reopened, edited, review_requested, synchronize]
7   push:
8     branches:
9       - "main"
10  workflow_call:
11
12 jobs:
13   check:
14     name: Conventional Commits
15     runs-on: ubuntu-latest
16     steps:
17       - uses: actions/checkout@v2
18       - uses: webiny/action-conventional-commits@v1.0.3
```

Este workflow, llamado **Commits Syntax Checker**, es una acción de GitHub que verifica si los mensajes de commit siguen un formato específico (Convencional Commits). Aquí está explicado por partes:

1. Desencadenadores (on)

El workflow se activa en tres casos:

- **pull_request**: Se ejecuta cuando se realiza una acción relacionada con un pull request en la rama **main**. Estas acciones incluyen:
 - **opened**: Cuando se crea un pull request.
 - **reopened**: Cuando se vuelve a abrir un pull request cerrado.
 - **edited**: Cuando se edita un pull request.
 - **review_requested**: Cuando se solicita una revisión del pull request.
 - **synchronize**: Cuando se hacen nuevos commits al pull request.
- **push**: Se ejecuta cuando hay un **push** directo a la rama **main**.
- **workflow_call**: Permite que este workflow sea llamado desde otros workflows.

2. Trabajo (jobs)

Define un trabajo llamado `check` con las siguientes características:

Nombre y Configuración Básica

- **name: Conventional Commits**: Nombre descriptivo del trabajo.
- **runs-on: ubuntu-latest**: Especifica que el trabajo se ejecutará en un entorno virtual con Ubuntu.

Pasos (steps)

Los pasos del trabajo son:

1. **actions/checkout@v2**:
 - Descarga el contenido del repositorio en el entorno virtual.
 - Esto es necesario para que las siguientes acciones puedan acceder al código.
2. **webiny/action-conventional-commits@v1.0.3**:
 - Una acción que verifica si los mensajes de los commits cumplen con las reglas de los **Commits Convencionales**.
 - Estas reglas generalmente incluyen prefijos como `feat:`, `fix:`, `chore:`, etc., seguidos de una descripción clara.

3. Propósito del Workflow

Este workflow asegura que:

- Todos los mensajes de commit en la rama `main` (ya sea mediante `push` o a través de un pull request) sigan el estándar de **Commits Convencionales**.
- Esto ayuda a mantener un historial de commits consistente y comprensible, lo que facilita el seguimiento de cambios y la generación de changelogs automáticos.

Ejemplo de Commit Convencional

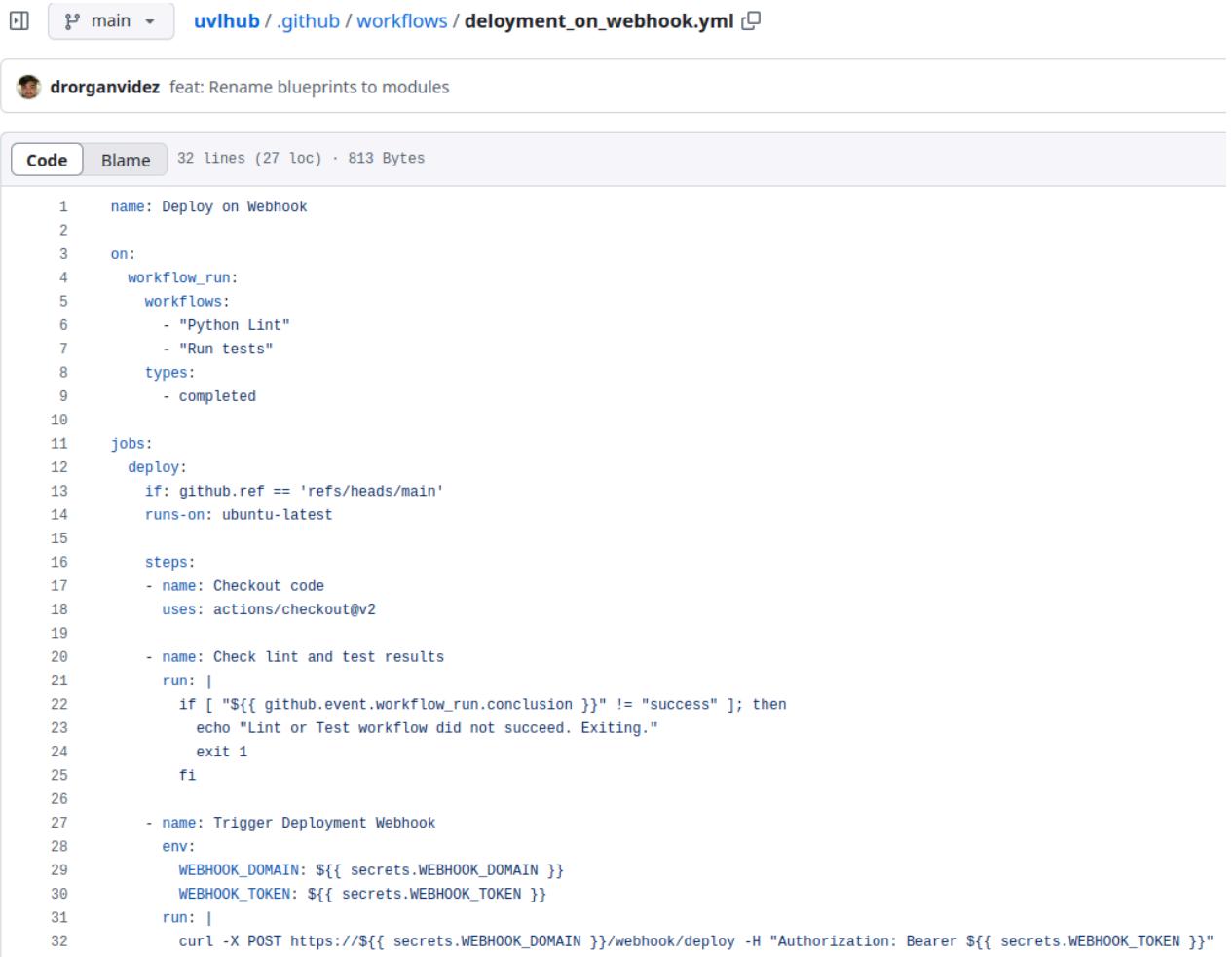
Un mensaje de commit válido podría ser:

`feat: add user authentication`

Esto indica que se añadió una nueva funcionalidad (feature).

Beneficios

- Automatiza la validación de mensajes de commit.
- Mejora la calidad del código al forzar un estándar.
- Aumenta la claridad en la colaboración del equipo.



The screenshot shows a GitHub Actions workflow named "deployment_on_webhook.yml". The workflow has a single job named "deploy". It triggers on the "main" branch and runs on an Ubuntu-latest runner. The job consists of two steps: "Checkout code" and "Check lint and test results". The second step uses a script to check if the workflow run was successful. If it wasn't, it prints an error message and exits. Finally, it triggers a deployment webhook to a specified domain and token.

```
name: Deploy on Webhook
on:
  workflow_run:
    workflows:
      - "Python Lint"
      - "Run tests"
    types:
      - completed
jobs:
  deploy:
    if: github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Check lint and test results
        run: |
          if [ "${{ github.event.workflow_run.conclusion }}" != "success" ]; then
            echo "Lint or Test workflow did not succeed. Exiting."
            exit 1
          fi
      - name: Trigger Deployment Webhook
        env:
          WEBHOOK_DOMAIN: ${{ secrets.WEBHOOK_DOMAIN }}
          WEBHOOK_TOKEN: ${{ secrets.WEBHOOK_TOKEN }}
        run: |
          curl -X POST https://${{ secrets.WEBHOOK_DOMAIN }}/webhook/deploy -H "Authorization: Bearer ${{ secrets.WEBHOOK_TOKEN }}"
```

Este workflow de GitHub Actions, llamado **Deploy on Webhook**, realiza un despliegue automático mediante un webhook si se cumplen las siguientes condiciones:

1. Desencadenadores (on)

El workflow se ejecuta cuando **otros workflows han terminado de ejecutarse**, específicamente los workflows llamados:

- **"Python Lint"**: Verifica el formato y estilo del código.
- **"Run tests"**: Ejecuta las pruebas del código.

Solo se activará cuando estos workflows hayan completado su ejecución, y se evalúa si su conclusión fue exitosa.

El tipo de evento que lo dispara es:

- **completed**: Indica que los workflows mencionados han finalizado, sin importar si su estado fue **success**, **failure**, o cualquier otro.

2. Trabajos (jobs)

Define un solo trabajo llamado `deploy`, con los siguientes elementos:

Condicional de Ejecución

```
if: github.ref == 'refs/heads/main'
```

Este trabajo solo se ejecutará si la rama que se procesó es `main`.

Configuración del Entorno

```
runs-on: ubuntu-latest
```

El trabajo se ejecutará en un entorno virtual con Ubuntu.

Pasos (steps)

El trabajo incluye tres pasos:

Paso 1: Checkout del Código

```
- name: Checkout code
  uses: actions/checkout@v2
```

- Descarga el contenido del repositorio para que esté disponible en el entorno virtual.
 - Esto es esencial para revisar los resultados y realizar operaciones con el código fuente.
-

Paso 2: Verificar Resultados de Lint y Pruebas

```
- name: Check lint and test results
  run: |
    if [ "${{ github.event.workflow_run.conclusion }}" != "success" ]; then
      echo "Lint or Test workflow did not succeed. Exiting."
      exit 1
    fi
```

Este paso revisa si los workflows previos ([Python Lint y Run tests](#)) tuvieron éxito:

- `${{ github.event.workflow_run.conclusion }}`: Es una variable que contiene el resultado (`success`, `failure`, etc.) del workflow que disparó este workflow.
 - Si el resultado **no es success**, el script imprime un mensaje y termina la ejecución con un código de salida `1`, cancelando el despliegue.
-

Paso 3: Despliegue usando un Webhook

```
- name: Trigger Deployment Webhook
  env:
    WEBHOOK_DOMAIN: ${{ secrets.WEBHOOK_DOMAIN }}
    WEBHOOK_TOKEN: ${{ secrets.WEBHOOK_TOKEN }}
  run: |
    curl -X POST https://${{ secrets.WEBHOOK_DOMAIN }}/webhook/deploy -H
    "Authorization: Bearer ${{ secrets.WEBHOOK_TOKEN }}"
```

- Este paso envía una solicitud **POST** a un webhook para iniciar el despliegue.
- Usa variables sensibles (almacenadas en **secrets**) para el dominio del webhook y el token de autenticación:
 - **WEBHOOK_DOMAIN**: Dominio del servicio que maneja el despliegue.
 - **WEBHOOK_TOKEN**: Token de seguridad para autenticar la solicitud.
- **curl**:
 - Realiza la solicitud a la URL: [https://\\${{ secrets.WEBHOOK_DOMAIN }}/webhook/deploy](https://${{ secrets.WEBHOOK_DOMAIN }}/webhook/deploy).
 - Incluye un encabezado de autorización con el token.

3. Propósito del Workflow

Este workflow automatiza el despliegue solo si:

1. Los workflows previos (**Python Lint** y **Run tests**) fueron exitosos.
2. El cambio ocurrió en la rama **main**.

En caso contrario, detiene el despliegue para evitar desplegar código no validado o con fallos.

```
main uvhub/.github/workflows/deployment_on_dockerhub.yml
de Blame 43 lines (35 loc) · 1.5 KB
5
6 # GitHub recommends pinning actions to a commit SHA.
7 # To get a newer version, you will need to update the SHA.
8 # You can also reference a tag or branch, but the action may change without warning.
9
10 name: Publish image in Docker Hub
11
12 on:
13   release:
14     types: [published]
15
16 jobs:
17   push_to_registry:
18     name: Push Docker image to Docker Hub
19     runs-on: ubuntu-latest
20     steps:
21       - name: Check out the repo
22         uses: actions/checkout@v3
23
24       - name: Log in to Docker Hub
25         uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
26         with:
27           username: ${{ secrets.DOCKER_USER }}
28           password: ${{ secrets.DOCKER_PASSWORD }}
29
30       - name: Build and push Docker image
31         run: docker build --build-arg VERSION_TAG=${{ github.event.release.tag_name }} -t drorganvidez/uvlhub:${{ github.event.release.tag_name }} -f docker/images/Dockerfile.prod .
32         env:
33           DOCKER_CLI_EXPERIMENTAL: enabled
34
35       - name: Push Docker image to Docker Hub
36         run: docker push drorganvidez/uvlhub:${{ github.event.release.tag_name }}
37
38       - name: Tag and push latest
39         run: |
40           docker tag drorganvidez/uvlhub:${{ github.event.release.tag_name }} drorganvidez/uvlhub:latest
41           docker push drorganvidez/uvlhub:latest
42         env:
43           DOCKER_CLI_EXPERIMENTAL: enabled
```

Este workflow, llamado **Publish image in Docker Hub**, publica automáticamente una imagen Docker en Docker Hub cuando se publica una nueva versión (release) en el repositorio de GitHub. A continuación, se detalla su estructura y funcionalidad:

1. Desencadenadores (on)

on:

release:

types: [published]

- El workflow se activa cuando se publica una nueva **release** en GitHub.
 - Específicamente, el evento debe ser de tipo **published**, lo que significa que la release ha sido oficialmente publicada (no borrador ni prerelease).
-

2. Trabajos (jobs)

Trabajo: push_to_registry

Configuración Básica

runs-on: ubuntu-latest

- El trabajo se ejecuta en un entorno virtual con Ubuntu.

Pasos (steps)

1. Clonar el Repositorio

```
- name: Check out the repo
  uses: actions/checkout@v3
```

- Descarga el contenido del repositorio en el entorno virtual para que las siguientes acciones puedan acceder al código fuente y a los archivos necesarios.
-

2. Iniciar Sesión en Docker Hub

```
- name: Log in to Docker Hub
  uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
  with:
    username: ${{ secrets.DOCKER_USER }}
    password: ${{ secrets.DOCKER_PASSWORD }}
```

- Usa la acción `docker/login-action` (de un tercero) para iniciar sesión en Docker Hub.
 - Las credenciales (usuario y contraseña) están almacenadas de manera segura en `secrets`:
 - **DOCKER_USER**: Nombre de usuario de Docker Hub.
 - **DOCKER_PASSWORD**: Contraseña de Docker Hub.
-

3. Construir y Publicar la Imagen Docker

```
- name: Build and push Docker image
  run: docker build --build-arg VERSION_TAG=${{ github.event.release.tag_name }} -t
drorganvidez/uvlhub:${{ github.event.release.tag_name }} -f docker/images/Dockerfile.prod .
  env:
    DOCKER_CLI_EXPERIMENTAL: enabled
```

- Construye una imagen Docker con el comando `docker build`.
 - **--build-arg VERSION_TAG=\${{ github.event.release.tag_name }}**: Pasa como argumento el nombre de la etiqueta (tag) de la release publicada.
 - **-t drorganvidez/uvlhub:\${{ github.event.release.tag_name }}**: Asigna un tag a la imagen basada en el nombre de la release.
 - **-f docker/images/Dockerfile.prod**: Especifica el archivo `Dockerfile` ubicado en `docker/images/Dockerfile.prod`.
 - **..**: Define el contexto de construcción como el directorio actual.
- **DOCKER_CLI_EXPERIMENTAL: enabled**: Habilita características experimentales de Docker CLI.

4. Publicar la Imagen Docker en Docker Hub

```
- name: Push Docker image to Docker Hub
  run: docker push drorganvidez/uvlhub:${{ github.event.release.tag_name }}
```

- Sube la imagen construida al repositorio de Docker Hub:
 - **drorganvidez/uvlhub**: Nombre del repositorio en Docker Hub.
 - **\${{ github.event.release.tag_name }}**: La etiqueta asignada a la imagen es el nombre de la release.
-

5. Etiquetar como `latest` y Publicar

```
- name: Tag and push latest
  run: |
    docker tag drorganvidez/uvlhub:${{ github.event.release.tag_name }}
    drorganvidez/uvlhub:latest
    docker push drorganvidez/uvlhub:latest
  env:
    DOCKER_CLI_EXPERIMENTAL: enabled
```

- Etiqueta la imagen publicada como `latest` para que sea accesible como la versión más reciente:
 - **docker tag**: Aplica la etiqueta `latest` a la imagen existente.
 - **docker push**: Publica la imagen con la etiqueta `latest` en Docker Hub.
 - **env**:
 - **DOCKER_CLI_EXPERIMENTAL: enabled**: habilita las **funciones experimentales** en la herramienta de línea de comandos de Docker (Docker CLI). Estas son características nuevas o en prueba que aún no están completamente integradas como funciones oficiales en Docker.

The screenshot shows a GitHub repository page for 'uvlhub'. The URL is 'uvlhub / .github / workflows / lint.yml'. The file is named 'lint.yml' and has 28 lines (22 loc) and 478 Bytes. The code block shows a workflow configuration:

```
1  name: Python Lint
2
3  on:
4    push:
5      branches: [main, develop]
6    pull_request:
7      branches: [main, develop]
8
9  jobs:
10   lint:
11     runs-on: ubuntu-latest
12
13   steps:
14     - uses: actions/checkout@v2
15
16     - name: Set up Python
17       uses: actions/setup-python@v2
18       with:
19         python-version: '3.x'
20
21     - name: Install Dependencies
22       run: |
23         python -m pip install --upgrade pip
24         pip install flake8
25
26     - name: Lint with flake8
27       run: |
28         flake8 app
```

Este workflow, llamado **Python Lint**, verifica la calidad del código Python en el repositorio utilizando la herramienta **Flake8**. Aquí está el desglose de cómo funciona:

1. Desencadenadores (on)

El workflow se activa en dos situaciones:

1. **push**:
 - Cuando hay un **push** a las ramas **main** o **develop**.
2. **pull_request**:
 - Cuando se abre, actualiza o sincroniza un pull request hacia las ramas **main** o **develop**.

Esto garantiza que cualquier cambio en estas ramas o contribución a ellas sea revisado automáticamente.

2. Trabajo (**jobs**)

Nombre del trabajo: `lint`

Configuración básica

`runs-on: ubuntu-latest`

- El trabajo se ejecuta en un entorno virtual con Ubuntu.
-

3. Pasos (**steps**)

El trabajo se compone de los siguientes pasos:

Paso 1: Clonar el repositorio

- `uses: actions/checkout@v2`
 - Descarga el código del repositorio en el entorno virtual, haciéndolo disponible para las operaciones posteriores.
-

Paso 2: Configurar Python

- `name: Set up Python`
 - `uses: actions/setup-python@v2`
 - `with:`
 - `python-version: '3.x'`
 - Configura el entorno con la versión 3.x de Python.
 - Esto asegura que la herramienta **Flake8** y los scripts Python sean compatibles.

Paso 3: Instalar Dependencias

```
- name: Install Dependencies
```

```
  run: |
```

```
    python -m pip install --upgrade pip
```

```
    pip install flake8
```

- Actualiza `pip` a la última versión.
 - Instala **Flake8**, una herramienta que revisa el código Python para detectar problemas de estilo y errores comunes.
-

Paso 4: Ejecutar Linter con Flake8

```
- name: Lint with flake8
```

```
  run: |
```

```
    flake8 app
```

- Ejecuta **Flake8** en el directorio `app` (asumiendo que el código principal está allí).
 - **Flake8** revisa:
 - Conformidad con PEP 8 (la guía de estilo de Python).
 - Errores de sintaxis o problemas comunes.
 - Problemas relacionados con el mantenimiento del código (como funciones o variables no utilizadas).
-

Resultados

1. **Éxito:** Si el código cumple con las reglas definidas por Flake8, el workflow finaliza con éxito.
2. **Error:** Si Flake8 encuentra problemas, el workflow falla y muestra un reporte detallado con los problemas detectados, ayudando a los desarrolladores a corregirlos.

drorganvidez feat: Rename blueprints to modules

Code

Blame 50 lines (42 loc) · 1.18 KB

```
1   name: Run tests
2
3   on:
4     push:
5       branches: [main, develop]
6     pull_request:
7       branches: [main, develop]
8
9   jobs:
10    pytest:
11      runs-on: ubuntu-latest
12
13    services:
14      mysql:
15        image: mysql:5.7
16        env:
17          MYSQL_ROOT_PASSWORD: uvlhub_root_password
18          MYSQL_DATABASE: uvlhubdb_test
19          MYSQL_USER: uvlhub_user
20          MYSQL_PASSWORD: uvlhub_password
21      ports:
22        - 3306:3306
23      options: --health-cmd="mysqladmin ping" --health-interval=10s --health-timeout=5s --health-retries=3
24
25    steps:
26      - uses: actions/checkout@v4
27
28      - uses: actions/setup-python@v5
29        with:
30          python-version: '3.12'
31
32      - name: Prepare environment
33        run: |
34          sed -i '/rosemary @ file:/\\//app/d' requirements.txt
35
36      - name: Install dependencies
37        run: |
38          python -m pip install --upgrade pip
39          pip install -r requirements.txt
40
41      - name: Run Tests
42        env:
43          FLASK_ENV: testing
44          MARIADB_HOSTNAME: 127.0.0.1
45          MARIADB_PORT: 3306
46          MARIADB_TEST_DATABASE: uvlhubdb_test
47          MARIADB_USER: uvlhub_user
48          MARIADB_PASSWORD: uvlhub_password
49        run: |
50          pytest app/modules/ --ignore-glob='*selenium*'
```

Este workflow, llamado **Run tests**, está diseñado para ejecutar pruebas automáticas en un entorno configurado, utilizando `pytest` como marco de pruebas y una base de datos MySQL como servicio. A continuación, se desglosan sus componentes y propósito.

1. Desencadenadores (on)

on:

```
push:  
  branches: [main, develop]  
pull_request:  
  branches: [main, develop]
```

El workflow se activa en dos situaciones:

1. **push**: Cuando hay un `push` a las ramas `main` o `develop`.
2. **pull_request**: Cuando se abre, actualiza o sincroniza un pull request dirigido a `main` o `develop`.

Esto asegura que cualquier cambio en estas ramas o contribución a ellas pase por un conjunto de pruebas antes de ser aceptado.

2. Trabajo (jobs)

Trabajo: `pytest`

Configuración Básica

runs-on: `ubuntu-latest`

- Se ejecuta en un entorno virtual con Ubuntu.
-

3. Servicios

services:

```
mysql:  
  image: mysql:5.7  
  env:  
    MYSQL_ROOT_PASSWORD: uvjhdb_root_password  
    MYSQL_DATABASE: uvjhdb_test  
    MYSQL_USER: uvjhdb_user  
    MYSQL_PASSWORD: uvjhdb_password  
  ports:  
    - 3306:3306
```

```
options: --health-cmd="mysqladmin ping" --health-interval=10s --health-timeout=5s  
--health-retries=3
```

Se configura un contenedor de MySQL (versión 5.7) como servicio para las pruebas:

- **VARIABLES DE ENTORNO (env):**
 - `MYSQL_ROOT_PASSWORD`: Contraseña para el usuario root.
 - `MYSQL_DATABASE`: Nombre de la base de datos de prueba (`uvlhubdb_test`).
 - `MYSQL_USER` y `MYSQL_PASSWORD`: Credenciales para un usuario estándar (`uvlhub_user`).
- **Puertos (ports):**
 - El puerto 3306 del contenedor se expone al mismo puerto en el entorno de ejecución.
- **Opciones de salud (options):**
 - Configura un comando de salud (`mysqladmin ping`) para verificar si el servicio MySQL está disponible antes de proceder.
 - Intervalo: 10 segundos.
 - Tiempo de espera: 5 segundos.
 - Reintentos: 3.

Esto asegura que el servicio de base de datos esté activo antes de que comiencen las pruebas.

4. Pasos (steps)

Paso 1: Clonar el repositorio

- uses: `actions/checkout@v4`

- Descarga el código fuente del repositorio en el entorno virtual.
-

Paso 2: Configurar Python

- uses: `actions/setup-python@v5`

with:

`python-version: '3.12'`

- Configura el entorno con Python 3.12.
-

Paso 3: Preparar el entorno

- name: Prepare environment

run: |

`sed -i '/rosemary @ file:/\\!d' requirements.txt`

- Realiza modificaciones al archivo `requirements.txt`:

- Elimina una línea específica que contiene `rosemary @ file:///app` (posiblemente una dependencia local no necesaria para las pruebas).
-

Paso 4: Instalar dependencias

```
- name: Install dependencies
```

```
run: |
```

```
  python -m pip install --upgrade pip
  pip install -r requirements.txt
```

- Actualiza `pip` a la última versión.
 - Instala las dependencias del proyecto listadas en `requirements.txt`.
-

Paso 5: Ejecutar las pruebas

```
- name: Run Tests
```

```
env:
```

```
  FLASK_ENV: testing
  MARIADB_HOSTNAME: 127.0.0.1
  MARIADB_PORT: 3306
  MARIADB_TEST_DATABASE: uvlhubdb_test
  MARIADB_USER: uvlhub_user
  MARIADB_PASSWORD: uvlhub_password
```

```
run: |
```

```
  pytest app/modules/ --ignore-glob='*selenium*'
```

- Configura variables de entorno necesarias para las pruebas:
 - `FLASK_ENV`: Define el entorno Flask como `testing`.
 - Variables de conexión para MySQL (`MARIADB_HOSTNAME`, `MARIADB_PORT`, etc.).
 - Ejecuta `pytest` con las siguientes características:
 - Ejecuta pruebas localizadas en `app/modules/`.
 - Ignora pruebas relacionadas con `selenium` (usando el flag `--ignore-glob`).
-

Resultados

1. Éxito:

- Si todas las pruebas pasan, el workflow finaliza correctamente.

2. Error:

- Si alguna prueba falla, el workflow se detiene y reporta los errores.

EJERCICIO C (DOCKER)

Error: Error response from daemon: driver failed programming external connectivity on endpoint mariadb_container (ea2122c53ccda9rb3e963ea07436a8c1c85912250a29180a39faa61699c60bd): failed to bind port 0.0.0.0:3306/tcp: Error starting userland proxy: listen tcp4 0.0.0.0:3306: bind: address already in use

```
sudo systemctl stop mariadb
```



Desplegar en Docker:
cp .env.docker.example .env

```
sudo docker compose -f docker/docker-compose.dev.yml up -d
```

Si hay problemas por temas cache y sale todo el rato error 502:
sudo scripts/clean_docker.sh sh

Para pararlo:
sudo docker compose -f docker/docker-compose.dev.yml down -v

Explicación docker/images/**Dockerfile.dev**:

Este archivo `Dockerfile.dev` es un archivo de configuración para construir una imagen de Docker personalizada diseñada específicamente para un entorno de desarrollo. A continuación, se explica paso a paso lo que hace cada parte del archivo:

1. Base de la imagen

```
dockerfile
```

FROM python:3.12-slim

Copiar código

- Utiliza una imagen oficial de Python (versión 3.12 en una versión ligera `slim`) como base.

2. Configuración del entorno

```
dockerfile
```

ENV PIP_ROOT_USER_ACTION=ignore

Copiar código

- Configura una variable de entorno para evitar que `pip` emita advertencias sobre la ejecución como usuario root.

3. Instalación de paquetes necesarios

```
dockerfile Copiar código

RUN apt-get update \
    && apt-get install -y --no-install-recommends mariadb-client \
    && apt-get install -y --no-install-recommends gcc libc-dev python3-dev libffi-dev \
    && apt-get install -y --no-install-recommends curl \
    && apt-get install -y --no-install-recommends bash \
    && apt-get install -y --no-install-recommends openrc \
    && apt-get install -y --no-install-recommends build-essential
```

- Actualiza la lista de paquetes e instala herramientas necesarias como:
 - `mariadb-client`: Cliente para interactuar con bases de datos MariaDB.
 - `gcc`, `libc-dev`, `python3-dev`, `libffi-dev`: Herramientas de desarrollo y compilación para extensiones de Python y otras dependencias.
 - `curl`, `bash`, `openrc`: Utilidades generales.

4. Instalación de Docker CLI

```
dockerfile Copiar código

RUN apt-get install -y --no-install-recommends ca-certificates gnupg lsb-release \
    && curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/trusted.gpg.d/docker.gpg \
    && echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list \
    && apt-get update \
    && apt-get install -y --no-install-recommends docker-ce-cli
```

- Agrega el cliente de Docker CLI para que los contenedores puedan interactuar con Docker.

5. Limpieza

```
dockerfile Copiar código

RUN apt-get clean \
    && rm -rf /var/lib/apt/lists/*
```

- Elimina archivos temporales creados durante la instalación de paquetes para reducir el tamaño de la imagen.

6. Configuración del directorio de trabajo

```
dockerfile
```

 Copiar código

```
WORKDIR /app
```

- Cambia el directorio actual dentro del contenedor a `/app`.

7. Copia y preparación de archivos

```
dockerfile
```

 Copiar código

```
COPY requirements.txt .
```

- Copia el archivo `requirements.txt` al directorio de trabajo en el contenedor.

```
dockerfile
```

 Copiar código

```
RUN find . -type d -name "__pycache__" -exec rm -r {} + && \  
      find . -type f -name "*.pyc" -exec rm -f {} +
```

- Elimina archivos de caché de Python durante la etapa de construcción.

```
dockerfile
```

 Copiar código

```
COPY --chmod=0755 scripts/wait-for-db.sh ./scripts/  
COPY --chmod=0755 scripts/init-testing-db.sh ./scripts/
```

- Copia los scripts `wait-for-db.sh` e `init-testing-db.sh` al directorio `scripts/` dentro del contenedor y les otorga permisos de ejecución.

```
dockerfile
```

 Copiar código

```
COPY rosemary/ ./rosemary  
COPY setup.py ./  
COPY docker/ ./docker
```

- Copia el código fuente (`rosemary/`), el archivo de configuración `setup.py` y otros archivos de Docker al contenedor.

8. Actualización de pip

```
dockerfile
```

 Copiar código

```
RUN pip install --no-cache-dir --upgrade pip
```

- Actualiza `pip` a la versión más reciente.

9. Instalación de dependencias

```
dockerfile
```

 Copiar código

```
RUN pip install -r requirements.txt
```

- Instala las dependencias de Python especificadas en `requirements.txt`.

```
dockerfile
```

 Copiar código

```
RUN pip install -e ./
```

- Instala el paquete local `rosemary` en modo editable.

10. Exposición de puertos

```
dockerfile
```

 Copiar código

```
EXPOSE 5000
```

- Expone el puerto 5000, que es típicamente usado por aplicaciones Flask.

Explicación docker/docker-compose.dev.yml:

Este archivo `docker-compose.dev.yml` define una configuración para levantar un entorno de desarrollo con varios servicios (contenedores) utilizando Docker Compose. A continuación, se explica cada parte del archivo:

1. Definición de servicios

El archivo define tres servicios principales: `web`, `db` y `nginx`.

Servicio web

```
yaml Copiar código

web:
  container_name: web_app_container
  image: drorganvidez/uvlhub:dev
  env_file:
    - ../.env
  expose:
    - "5000"
  depends_on:
    - db
  build:
    context: ../
    dockerfile: docker/images/Dockerfile.dev
  volumes:
    - ..:/app
    - /var/run/docker.sock:/var/run/docker.sock
  command: [ "sh", "-c", "sh /app/docker/entrypoints/development_entrypoint.sh" ]
  networks:
    - uvlhub_network
```

- 1. Nombre del contenedor:** Se crea un contenedor llamado `web_app_container`.
- 2. Imagen base:** Usa la imagen `drorganvidez/uvlhub:dev` como base. Sin embargo, también se incluye una opción de **build** para crear una imagen personalizada si se modifica el `Dockerfile`.
- 3. Archivo de entorno:**
 - Utiliza variables de entorno definidas en `.../.env`.
- 4. Exposición de puerto:**
 - Expone el puerto `5000` (interno), donde probablemente estará la aplicación web.
- 5. Dependencias:**
 - Este servicio depende de `db` (base de datos), lo que asegura que `db` se inicie antes de `web`.
- 6. Build:**
 - Construye la imagen usando el contexto `.../` (directorio raíz) y el archivo `docker/images/Dockerfile.dev`.
- 7. Volúmenes:**
 - Monta el código fuente desde `.../` en `/app` dentro del contenedor (útil para desarrollo en tiempo real).
 - Monta el socket de Docker (`/var/run/docker.sock`) para que este contenedor pueda interactuar con el daemon de Docker del host.
- 8. Comando de inicio:**
 - Ejecuta un script llamado `development_entrypoint.sh`, que podría configurar o iniciar la aplicación en modo desarrollo.
- 9. Red:**
 - Este servicio está conectado a la red `uvlhub_network`.

Servicio db (Base de datos MariaDB)

yaml

Copiar código

```
db:
  container_name: mariadb_container
  env_file:
    - ../../.env
  build:
    context: ../
    dockerfile: docker/images/Dockerfile.mariadb
  restart: always
  ports:
    - "3306:3306"
  volumes:
    - db_data:/var/lib/mysql
  networks:
    - uvlhub_network
```

1. **Nombre del contenedor:** `mariadb_container`.

2. **Archivo de entorno:**

- Usa variables definidas en `../../.env` para configurar la base de datos (por ejemplo, usuario, contraseña).

3. **Build:**

- Construye la imagen con el `Dockerfile.mariadb`.

4. **Reinicio automático:**

- Siempre reinicia el contenedor si falla (`restart: always`).

5. **Puertos:**

- Mapea el puerto `3306` del contenedor al puerto `3306` del host (puerto estándar para MariaDB/MySQL).

6. **Volúmenes:**

- Utiliza un volumen llamado `db_data` para persistir los datos en `/var/lib/mysql` (directorio donde MariaDB almacena los datos).

7. **Red:**

- Conectado a `uvlhub_network`.

Servicio nginx (Servidor web)

yaml

Copiar código

```
nginx:
  container_name: nginx_web_server_container
  image: nginx:latest
  volumes:
    - ./nginx/nginx.dev.conf:/etc/nginx/nginx.conf
    - ./nginx/html:/usr/share/nginx/html
  ports:
    - "80:80"
  depends_on:
    - web
  networks:
    - uvhub_network
```

1. **Nombre del contenedor:** `nginx_web_server_container`.
2. **Imagen base:** Usa la imagen oficial `nginx:latest`.
3. **Volúmenes:**
 - Monta el archivo de configuración de Nginx (`nginx.dev.conf`) en `/etc/nginx/nginx.conf`.
 - Monta contenido estático desde `./nginx/html` en `/usr/share/nginx/html`.
4. **Puertos:**
 - Mapea el puerto `80` del contenedor al puerto `80` del host (puerto estándar HTTP).

5. Dependencias:

- Depende de `web` (el servicio de la aplicación web debe estar activo para que Nginx funcione correctamente).

6. Red:

- Conectado a `uvlhub_network`.
-

2. Definición de volúmenes

yaml

 Copiar código

```
volumes:  
  db_data:
```

- Define un volumen llamado `db_data`, que se utiliza para almacenar los datos de la base de datos de manera persistente.

3. Definición de redes

yaml

 Copiar código

```
networks:  
  uvlhub_network:
```

- Define una red llamada `uvlhub_network`, a la cual se conectan todos los servicios para comunicarse entre sí.

EJERCICIO D (VAGRANT)

Antes de nada borrar los siguientes archivos:

`rm -r uploads`

`rm -r rosemary.egg-info`

`rm app.log*`

Para desplegar Vagrant:

`cp .env.vagrant.example .env`

Ahora pasamos a la carpeta de vagrant:

```
cd vagrant
```

Explicación vagrant/Vagrantfile:

Este archivo `Vagrantfile` configura una máquina virtual (VM) utilizando Vagrant, una herramienta para gestionar entornos de desarrollo virtualizados. A continuación, te explico cada parte del archivo y lo que hace:

1. Cargar variables de entorno desde un archivo `.env`

```
ruby Copiar código
def load_env(file)
  ...
end
```

- **Propósito:** Carga variables de entorno desde un archivo `.env`, las almacena en un hash y las asigna a las variables de entorno del sistema (`ENV`).
 - Si el archivo `.env` no existe, genera un error.
-
- **Ejemplo:** Si el archivo `.env` contiene:

```
env Copiar código
FLASK_APP_NAME=my_app
FLASK_ENV=development
```

Estas variables estarán disponibles como `ENV['FLASK_APP_NAME']` y `ENV['FLASK_ENV']`.

2. Configuración de la máquina virtual con Vagrant

```
ruby Copiar código
Vagrant.configure("2") do |config|
```

- **Vagrant.configure("2"):** Define la configuración de la máquina virtual en formato compatible con Vagrant 2.x.

2.1. Base box

```
ruby  
  
config.vm.box = "ubuntu/jammy64"
```

 Copiar código

- Selecciona la imagen base de la VM, en este caso, `ubuntu/jammy64` (Ubuntu 22.04).
-

2.2. Redirección de puertos

```
ruby  
  
config.vm.network "forwarded_port", guest: 5000, host: 5000  
config.vm.network "forwarded_port", guest: 8089, host: 8089
```

 Copiar código

- Redirige puertos de la VM al host:
 - Puerto 5000: Para una aplicación Flask o similar.
 - Puerto 8089: Para otra herramienta o servicio.
-

2.3. Carpetas sincronizadas

```
ruby  
  
config.vm.synced_folder "../", "/vagrant"
```

 Copiar código

- Sincroniza el directorio padre del `Vagrantfile` en el host (`../`) con el directorio `/vagrant` dentro de la VM. Esto permite compartir archivos entre el host y la VM.

2.4. Provisión con Ansible

```
ruby

config.vm.provision "ansible" do |ansible|
  ansible.playbook = "00_main.yml"
  ansible.extra_vars = {
    flask_app_name: ENV['FLASK_APP_NAME'],
    flask_env: ENV['FLASK_ENV'],
    domain: ENV['DOMAIN'],
    ...
  }
end
```

Copiar código

- Usa Ansible para aprovisionar la VM:

- **Playbook:** `00_main.yml` contiene las tareas para configurar la VM.
- **Variables:** Se pasan variables a Ansible, cargadas previamente desde el archivo `.env`. Estas variables configuran servicios como Flask y MariaDB.

2.5. Configuración de variables de entorno en la VM

```
ruby

config.vm.provision "shell", inline: <<-SHELL
  ENV_FILE="/vagrant/.env"
  ...
SHELL
```

Copiar código

- **Primera provisión de shell:**

- Carga las variables del archivo `.env` en el entorno de la VM.
- Crea un script (`/etc/profile.d/vagrant_env.sh`) para que estas variables estén disponibles al iniciar sesión.

- **Segunda provisión de shell:**

- Asegura que las variables se carguen automáticamente cada vez que se inicie sesión en la VM.

2.6. Configuración del proveedor (VirtualBox)

ruby

 Copiar código

```
config.vm.provider "virtualbox" do |vb|
  vb.memory = "2048"
  vb.cpus = 4
end
```

- Configura la VM para ejecutarse en VirtualBox:
 - **Memoria:** 2048 MB (2 GB).
 - **CPUs:** 4 núcleos.

Explicación vagrant/00_main.yml:

Este archivo `00_main.yml` es un playbook principal de Ansible que actúa como un **índice** para importar y ejecutar otros playbooks de manera organizada. A continuación, te explico cada parte:

1. Importación de otros playbooks

yaml

 Copiar código

```
- import_playbook: 01_setup.yml
- import_playbook: 02_install_mariadb.yml
- import_playbook: 03_mariadb_scripts.yml
- import_playbook: 04_install_dependencies.yml
- import_playbook: 05_run_app.yml
- import_playbook: 06_utilities.yml
```

Cada línea utiliza el módulo `import_playbook` de Ansible, que sirve para incluir y ejecutar otros playbooks en el orden especificado. Esto divide las tareas de aprovisionamiento en pasos más manejables, haciendo el proceso más modular y fácil de mantener.

2. Propósito de cada playbook importado

2.1. 01_setup.yml

- **Propósito:** Configuración inicial del entorno.
 - Podría incluir tareas como:
 - Actualizar paquetes del sistema.
 - Configurar usuarios y claves SSH.
 - Establecer permisos básicos.
-

2.2. 02_install_mariadb.yml

- **Propósito:** Instalación de MariaDB (base de datos).
 - Podría incluir:
 - Instalación del servidor de MariaDB.
 - Configuración inicial (archivos de configuración, usuarios).
 - Habilitación del servicio para que arranque automáticamente.

2.3. 03_mariadb_scripts.yml

- **Propósito:** Ejecución de scripts SQL para configurar la base de datos.
 - Podría incluir:
 - Creación de esquemas y tablas.
 - Inserción de datos iniciales.
 - Configuración de permisos y usuarios específicos para la aplicación.
-

2.4. 04_install_dependencies.yml

- **Propósito:** Instalación de dependencias necesarias para la aplicación.
 - Podría incluir:
 - Instalación de herramientas de desarrollo (por ejemplo, `pip`, `npm`, etc.).
 - Instalación de bibliotecas específicas requeridas por la aplicación Flask o cualquier otro componente.

2.5. `05_run_app.yml`

- **Propósito:** Configuración y ejecución de la aplicación.
 - Podría incluir:
 - Configuración de variables de entorno para Flask.
 - Ejecución del servidor de desarrollo o despliegue de la aplicación en un servidor como Gunicorn.
 - Configuración del proxy inverso si es necesario (por ejemplo, con Nginx).
-

2.6. `06_utilities.yml`

- **Propósito:** Tareas adicionales o utilidades.
 - Podría incluir:
 - Configuración de herramientas auxiliares (monitoreo, registro de logs).
 - Limpieza de archivos temporales o configuraciones sobrantes.
 - Creación de backups o snapshots del entorno.

Explicación `vagrant/01_setup.yml`:

Este archivo `01_setup.yml` es un **playbook de Ansible** que realiza una tarea básica de configuración inicial del sistema en los hosts especificados. A continuación, te explico su estructura y propósito:

1. Definición de los hosts

yaml

 Copiar código

```
- hosts: all
```

- `hosts: all`: Este playbook se ejecutará en todos los hosts definidos en el inventario.
 - Por ejemplo, si tienes varios servidores en tu infraestructura (base de datos, aplicación, etc.), se aplicará a todos.

2. Escalado de privilegios

```
yaml
```

 Copiar código

```
become: true
```

- `become: true` : Ejecuta las tareas con privilegios elevados (similar a usar `sudo` en una terminal). Esto es necesario para realizar cambios en el sistema, como actualizar paquetes.

3. Definición de tareas

```
yaml
```

 Copiar código

```
tasks:  
  - name: Update the system  
    apt:  
      update_cache: yes
```

```
tasks :
```

- Es una lista de tareas que Ansible ejecutará en el orden en que están definidas.

Primera tarea: "Update the system":

- `name` : Descripción de la tarea. En este caso, "Actualizar el sistema".
- `apt` : Usa el módulo `apt` de Ansible, que es específico para sistemas basados en Debian (como Ubuntu).
 - `update_cache: yes` : Actualiza la caché de los paquetes disponibles. Equivale a ejecutar `sudo apt-get update` en la terminal.

Explicación vagrant/02_install_mariadb.yml:

Este archivo `02_install_mariadb.yml` es un playbook de Ansible que instala y configura **MariaDB** en los hosts especificados. A continuación, se detalla lo que hace cada sección del archivo:

1. Definición de hosts y permisos

yaml

 Copiar código

```
- hosts: all  
become: true
```

- `hosts: all`: Aplica las tareas a todos los hosts definidos en el inventario.
- `become: true`: Escala privilegios para ejecutar tareas como administrador (equivalente a `sudo`).

2. Lista de tareas

Tarea 1: Instalar MariaDB Server

yaml

 Copiar código

```
- name: Install MariaDB Server  
apt:  
  name:  
    - mariadb-server  
    - python3-pymysql  
  update_cache: yes
```

- Instala:
 - `mariadb-server`: El servidor MariaDB.
 - `python3-pymysql`: Cliente Python necesario para interactuar con MariaDB desde scripts Python.
- Actualiza la caché de paquetes antes de instalar.

Tarea 2: Iniciar y habilitar el servicio de MariaDB

yaml

 Copiar código

```
- name: Start and enable MariaDB service
  systemd:
    name: mariadb
    state: started
    enabled: yes
```

- Usa el módulo `systemd` para:
 - `state: started` : Asegurarse de que el servicio MariaDB está en ejecución.
 - `enabled: yes` : Habilitar el servicio para que inicie automáticamente con el sistema.

Tarea 3: Verificar si la contraseña de root ya está configurada

yaml

 Copiar código

```
- name: Check if MariaDB root password is already set
  shell: "mysql -u root -p'{{ mariadb_root_password }}' -e 'SELECT 1;''"
  register: mariadb_root_check
  failed_when: mariadb_root_check.rc not in [0, 1]
  changed_when: false
  ignore_errors: true
```

- Usa el módulo `shell` para ejecutar un comando SQL que verifica si la contraseña de root está configurada.
- Guarda el resultado en la variable `mariadb_root_check` :
 - `failed_when` : Define las condiciones para considerar la tarea como fallida.
 - `changed_when` : Marca la tarea como no cambiada.
 - `ignore_errors: true` : Ignora errores para manejar casos en los que la contraseña no está configurada.

Tarea 4: Configurar la contraseña de root si no está configurada

yaml

 Copiar código

```
- name: Set MariaDB root password if not set
  mysql_user:
    login_unix_socket: /run/mysqld/mysqld.sock
    login_user: 'root'
    login_password: ''
    name: 'root'
    password: '{{ mariadb_root_password }}'
    state: present
  when: mariadb_root_check.failed
```

- Usa el módulo `mysql_user` para:
 - Conectar a MariaDB usando el socket UNIX.
 - Configurar la contraseña de root con el valor de `{{ mariadb_root_password }}`.
 - Solo se ejecuta si la contraseña no está configurada (`when: mariadb_root_check.failed`).

Tarea 5: Crear archivo `.my.cnf` para root

yaml

 Copiar código

```
- name: Create .my.cnf for root
  copy:
    dest: /root/.my.cnf
    content: |
      [client]
      user=root
      password={{ mariadb_root_password }}
    owner: root
    mode: '0600'
```

- Crea un archivo `/root/.my.cnf` que almacena las credenciales de root para MariaDB.
- Esto permite ejecutar comandos MySQL sin tener que proporcionar la contraseña explícitamente.

Tarea 6: Crear script SQL

```
yaml Copiar código

- name: Create SQL script
  copy:
    content: |
      CREATE DATABASE IF NOT EXISTS {{ mariadb_database }};
      CREATE DATABASE IF NOT EXISTS {{ mariadb_test_database }};
      CREATE USER IF NOT EXISTS '{{ mariadb_user }}'@'localhost' IDENTIFIED BY '{{ mariadb_password }}';
      GRANT ALL PRIVILEGES ON {{ mariadb_database }}.* TO '{{ mariadb_user }}'@'localhost';
      GRANT ALL PRIVILEGES ON {{ mariadb_test_database }}.* TO '{{ mariadb_user }}'@'localhost';
      FLUSH PRIVILEGES;
  dest: /tmp/setup.sql
```

- Crea un archivo SQL temporal en `/tmp/setup.sql` con las siguientes instrucciones:
 - **Bases de datos:**
 - Crea las bases de datos principales (`{{ mariadb_database }}`) y de prueba (`{{ mariadb_test_database }}`).
 - **Usuario:**
 - Crea el usuario `{{ mariadb_user }}` con la contraseña `{{ mariadb_password }}`.
 - Otorga permisos completos sobre ambas bases de datos.
 - **Flush Privileges:**
 - Asegura que los cambios en permisos sean aplicados.

Tarea 7: Importar el script SQL

```
yaml Copiar código

- name: Import SQL script
  command: bash -c "mysql < /tmp/setup.sql"
```

- Usa el comando `mysql` para ejecutar el script SQL creado en la tarea anterior.

Tarea 8: Eliminar el script SQL temporal

yaml

Copiar código

```
- name: Remove temporary SQL script
  file:
    path: /tmp/setup.sql
    state: absent
```

- Usa el módulo `file` para eliminar el archivo `/tmp/setup.sql` después de usarlo.

Explicación vagrant/03_mariadb_scripts.yml:

El archivo `03_mariadb_scripts.yml` es un **playbook de Ansible** diseñado para manejar scripts relacionados con MariaDB, asegurando que el entorno esté listo y configurado correctamente para trabajar con la base de datos. A continuación, te detallo lo que hace cada parte del archivo:

1. Definición de hosts y permisos

yaml

Copiar código

```
- hosts: all
  become: true
```

- `hosts: all`: Este playbook se ejecutará en todos los hosts especificados en el inventario.
- `become: true`: Escala privilegios para ejecutar tareas como administrador.

2. Definición de variables (vars)

yaml

Copiar código

```
vars:  
  common_environment:  
    FLASK_APP_NAME: "{{ flask_app_name }}"  
    FLASK_ENV: "{{ flask_env }}"  
    DOMAIN: "{{ domain }}"  
    MARIADB_HOSTNAME: "{{ mariadb_hostname }}"  
    MARIADB_PORT: "{{ mariadb_port }}"  
    MARIADB_DATABASE: "{{ mariadb_database }}"  
    MARIADB_TEST_DATABASE: "{{ mariadb_test_database }}"  
    MARIADB_USER: "{{ mariadb_user }}"  
    MARIADB_PASSWORD: "{{ mariadb_password }}"  
    MARIADB_ROOT_PASSWORD: "{{ mariadb_root_password }}"  
    WORKING_DIR: "{{ working_dir }}"
```

- Define un conjunto de variables bajo `common_environment` que serán utilizadas como entorno (`environment`) en las tareas.
- Estas variables son probablemente heredadas desde el archivo `.env` cargado previamente en el proceso.

3. Lista de tareas

Tarea 1: Establecer permisos para `wait-for-db.sh`

yaml

Copiar código

```
- name: Set permissions for wait-for-db.sh  
  file:  
    path: "{{ working_dir }}/scripts/wait-for-db.sh"  
    mode: '0755'  
    state: file  
  environment: "{{ common_environment }}"
```

- Usa el módulo `file` para:
 - Localizar el script `wait-for-db.sh` en el directorio `scripts/` dentro del `working_dir`.
 - Cambiar sus permisos a `0755` (lectura, escritura y ejecución para el propietario; lectura y ejecución para otros).
 - Asegurar que el archivo esté presente (`state: file`).
- Se utiliza el entorno definido en `common_environment`.

Tarea 2: Establecer permisos para `init-testing-db.sh`

```
yaml  
  
- name: Set permissions for init-testing-db.sh  
  file:  
    path: "{{ working_dir }}scripts/init-testing-db.sh"  
    mode: '0755'  
    state: file  
  environment: "{{ common_environment }}"
```

 Copiar código

- Similar a la tarea anterior, pero aplica los mismos cambios al script `init-testing-db.sh`.

Tarea 3: Esperar a que MariaDB esté listo

```
yaml  
  
- name: Wait for MariaDB to be ready  
  shell: |  
    {{ working_dir }}scripts/wait-for-db.sh  
  args:  
    executable: /bin/bash  
  environment: "{{ common_environment }}"
```

 Copiar código

- Ejecuta el script `wait-for-db.sh`, que probablemente verifica si el servicio MariaDB está listo para aceptar conexiones.
- Usa el módulo `shell` con el intérprete de comandos `/bin/bash`.
- Pasa las variables de entorno definidas en `common_environment`.

Tarea 4: Inicializar la base de datos

```
yaml
- name: Initialize the database
  shell: |
    {{ working_dir }}scripts/init-testing-db.sh
  args:
    executable: /bin/bash
  environment: "{{ common_environment }}"
```

Copiar código

- Ejecuta el script `init-testing-db.sh`, que probablemente realiza tareas como:
 - Configurar datos iniciales en la base de datos.
 - Crear tablas, usuarios, o bases de datos adicionales para pruebas.
- Al igual que en la tarea anterior, usa `/bin/bash` como intérprete y pasa las variables de entorno.

Explicación `vagrant/04_install_dependencies.yml`:

El archivo `04_install_dependencies.yml` es un **playbook de Ansible** que configura Python 3.12, un entorno virtual, e instala las dependencias necesarias para el proyecto. A continuación, se explica cada parte del archivo y su propósito:

1. Definición de hosts y permisos

```
yaml
- hosts: all
  become: true
```

Copiar código

- `hosts: all`: Este playbook se ejecutará en todos los hosts definidos en el inventario.
- `become: true`: Escala privilegios para ejecutar tareas con permisos administrativos.

2. Lista de tareas

Tarea 1: Agregar el repositorio `deadsnakes` para Python 3.12

yaml

Copiar código

```
- name: Add deadsnakes PPA for installing Python 3.12
  apt_repository:
    repo: ppa:deadsnakes/ppa
    state: present
    update_cache: yes
```

- Usa el módulo `apt_repository` para:
 - Agregar el repositorio `deadsnakes`, que contiene versiones recientes de Python no disponibles en los repositorios oficiales.
 - Actualizar la caché de paquetes (`update_cache: yes`) después de agregar el repositorio.

Tarea 2: Instalar Python 3.12 y dependencias

yaml

Copiar código

```
- name: Update the system and install Python 3.12 and dependencies
  apt:
    name:
      - python3.12
      - python3.12-venv
      - mariadb-client
    state: present
```

- Usa el módulo `apt` para instalar:
 - `python3.12`: El intérprete de Python 3.12.
 - `python3.12-venv`: Herramientas para crear entornos virtuales en Python.
 - `mariadb-client`: Cliente de MariaDB para interactuar con la base de datos.

Tarea 3: Instalar pip y setuptools para Python 3.12 desde el origen

yaml

Copiar código

```
- name: Install pip and setuptools for Python 3.12 from source
  shell: |
    wget https://bootstrap.pypa.io/get-pip.py -O /tmp/get-pip.py
    python3.12 /tmp/get-pip.py
  args:
    executable: /bin/bash
```

- Descarga el instalador de `pip` desde su fuente oficial (`get-pip.py`).
- Ejecuta el script con Python 3.12 para instalar:
 - `pip`: El gestor de paquetes de Python.
 - `setuptools`: Herramientas para empaquetar e instalar proyectos de Python.

Tarea 4: Actualizar pip y setuptools

yaml

Copiar código

```
- name: Upgrade pip and setuptools for Python 3.12
  shell: |
    python3.12 -m pip install --upgrade pip setuptools
  args:
    executable: /bin/bash
```

- Usa `pip` para actualizar:
 - `pip`: Asegura la versión más reciente.
 - `setuptools`: Mantiene las herramientas de empaquetado actualizadas.

Tarea 5: Configurar un entorno virtual

yaml

 Copiar código

```
- name: Set up the Python 3.12 virtual environment
  command: python3.12 -m venv {{ working_dir }}/vagrant_venv
  args:
    creates: "{{ working_dir }}/vagrant_venv/bin/activate"
```

- Usa el módulo `command` para:
 - Crear un entorno virtual en el directorio `{{ working_dir }}/vagrant_venv`.
 - La opción `creates` evita que esta tarea se ejecute si el archivo `activate` ya existe, optimizando el tiempo de ejecución.

Tarea 6: Activar el entorno virtual e instalar dependencias

yaml

 Copiar código

```
- name: Activate the virtual environment and install dependencies
  shell: |
    source {{ working_dir }}/vagrant_venv/bin/activate
    pip install --upgrade pip
    cd {{ working_dir }}
    pip install -r requirements.txt
    pip install -e ./
  args:
    executable: /bin/bash
```

- Usa el módulo `shell` para realizar las siguientes acciones en el entorno virtual:
 1. **Activar el entorno virtual:** `source {{ working_dir }}/vagrant_venv/bin/activate`.
 2. **Actualizar pip:** Asegura que `pip` esté actualizado dentro del entorno virtual.
 3. **Cambiar al directorio de trabajo:** `cd {{ working_dir }}`.
 4. **Instalar dependencias:**
 - Instalar paquetes listados en `requirements.txt`.
 - Instalar el proyecto en modo editable (`pip install -e ./`).

Explícame vagrant/05_run_app.yml:

El archivo `05_run_app.yml` es un **playbook de Ansible** diseñado para ejecutar una aplicación Flask configurada previamente. A continuación, se explica cada sección del archivo y su propósito:

1. Definición de hosts y permisos

```
yaml
  - hosts: all
    become: true
```

- `hosts: all`: Este playbook se ejecutará en todos los hosts definidos en el inventario.
- `become: true`: Escala privilegios para ejecutar tareas con permisos administrativos.

2. Variables (`vars`)

```
yaml
vars:
  common_environment:
    FLASK_APP_NAME: "{{ flask_app_name }}"
    FLASK_ENV: "{{ flask_env }}"
    DOMAIN: "{{ domain }}"
    MARIADB_HOSTNAME: "{{ mariadb_hostname }}"
    MARIADB_PORT: "{{ mariadb_port }}"
    MARIADB_DATABASE: "{{ mariadb_database }}"
    MARIADB_TEST_DATABASE: "{{ mariadb_test_database }}"
    MARIADB_USER: "{{ mariadb_user }}"
    MARIADB_PASSWORD: "{{ mariadb_password }}"
    MARIADB_ROOT_PASSWORD: "{{ mariadb_root_password }}"
    WORKING_DIR: "{{ working_dir }}"
```

- Define un conjunto de variables que representan el entorno común necesario para la ejecución de las tareas.
- Estas variables se heredan probablemente de un archivo `.env` cargado anteriormente.

3. Lista de tareas

Tarea 1: Agregar un webhook a .moduleignore

yaml

Copiar código

```
- name: Add webhook to .moduleignore
  shell: echo "webhook" > {{ working_dir }}.moduleignore
  args:
    executable: /bin/bash
    environment: "{{ common_environment }}"
```

- Usa el módulo `shell` para agregar una línea con el texto `webhook` al archivo `.moduleignore` en el directorio de trabajo (`{{ working_dir }}`).
 - **Propósito:** Esto puede ser útil para ignorar ciertos módulos o archivos específicos durante las operaciones del proyecto (como generación de paquetes o despliegue).

Tarea 2: Configurar la base de datos con Rosemary

yaml

Copiar código

```
- name: Set database with Rosemary
  shell: |
    source {{ working_dir }}vagrant_venv/bin/activate
    cd {{ working_dir }}
    flask db upgrade
    rosemary db:seed -y --reset
  args:
    executable: /bin/bash
    environment: "{{ common_environment }}"
```

- Usa el módulo `shell` para ejecutar comandos de Flask y Rosemary en el entorno virtual:
 1. **Activar el entorno virtual:** `source {{ working_dir }}vagrant_venv/bin/activate`.
 2. **Cambiar al directorio de trabajo:** `cd {{ working_dir }}`.
 3. **Actualizar la base de datos:**
 - `flask db upgrade`: Aplica las migraciones de la base de datos para actualizar su esquema.
 4. **Sembrar datos en la base de datos:**
 - `rosemary db:seed -y --reset`: Inserta datos iniciales en la base de datos y la reinicia si es necesario.

Tarea 3: Ejecutar la aplicación Flask

```
yaml Copiar código  
  
- name: Run Flask application  
  shell: |  
    source {{ working_dir }}vagrant_venv/bin/activate  
    cd {{ working_dir }}  
    nohup flask run --host=0.0.0.0 --port=5000 --reload --debug > app.log 2>&1 &  
  args:  
    executable: /bin/bash  
  environment: "{{ common_environment }}"  
  async: 1  
  poll: 0
```

- Usa el módulo `shell` para iniciar la aplicación Flask en modo asíncrono:
 1. Activar el entorno virtual: `source {{ working_dir }}vagrant_venv/bin/activate`.
 2. Cambiar al directorio de trabajo: `cd {{ working_dir }}`.
 3. Iniciar la aplicación Flask:
 - `nohup flask run`:
 - `--host=0.0.0.0`: Permite el acceso desde cualquier IP (no solo localhost).
 - `--port=5000`: Usa el puerto 5000 para la aplicación.
 - `--reload`: Permite recargar automáticamente los cambios en desarrollo.
 - `--debug`: Habilita el modo de depuración.
 - Redirige la salida a un archivo de log (`app.log`).
 4. Modo asíncrono:
 - `async: 1`: Ejecuta la tarea en segundo plano.
 - `poll: 0`: No espera a que la tarea finalice.

Explicación vagrant/06_utilities.yml:

El archivo `06_utilities.yml` es un **playbook de Ansible** que se utiliza para agregar configuraciones de utilidad al entorno del usuario, en este caso al archivo `.bashrc` del usuario `vagrant`. A continuación, se detalla lo que hace:

1. Definición de hosts y permisos

yaml

 Copiar código

```
- hosts: all
  become: true
```

- `hosts: all`: Este playbook se aplica a todos los hosts definidos en el inventario.
- `become: true`: Escala privilegios para ejecutar las tareas como administrador.

2. Lista de tareas

Tarea: Agregar comandos al archivo .bashrc

yaml

Copiar código

```
- name: Add commands to .bashrc
  lineinfile:
    path: /home/vagrant/.bashrc
    line: '{{ item }}'
    create: yes
  with_items:
    - 'source {{ working_dir }}vagrant_venv/bin/activate'
    - 'cd {{ working_dir }}'
  when: ansible_user == 'vagrant'
```

1. Descripción (name):

- Agrega comandos al archivo .bashrc del usuario vagrant .

2. Módulo utilizado: lineinfile :

- path : Define el archivo donde se agregarán las líneas (/home/vagrant/.bashrc).
- line : Especifica la línea a agregar al archivo. En este caso, se utiliza {{ item }} para iterar sobre las líneas definidas en with_items .
- create: yes : Si el archivo no existe ↓ creará automáticamente.

3. Lista de elementos (with_items):

- source {{ working_dir }}vagrant_venv/bin/activate :
 - Agrega un comando para activar automáticamente el entorno virtual de Python al iniciar sesión.
- cd {{ working_dir }}:
 - Agrega un comando para cambiar al directorio de trabajo (working_dir) automáticamente al iniciar sesión.

4. Condición (when):

- ansible_user == 'vagrant' : La tarea solo se ejecuta si el usuario que ejecuta Ansible es vagrant .

EJERCICIO E (RENDER)

Intensificación técnica

- Realice los cambios necesarios para desplegar DECIDE en Render mediante el ciclo de integración y despliegue continuos.
- Haga commit y push de los cambios realizados.

CI

The screenshot shows two browser windows. The left window is titled 'Installing Render — Mozilla Firefox' and displays the configuration for a GitHub app. It asks to install on the organization 'Daniel\JuanEGC' for repositories owned by the resource owner or selected repositories. It lists permissions: Read access to Dependabot alerts, administration, code, members, metadata, and organization hooks; and Read and write access to actions, checks, commit statuses, deployments, environments, issues, pull requests, repository hooks, and workflows. It also includes user permissions for email addresses. The right window is the 'Render' dashboard under 'Settings > Services'. It shows a list of services: 'control-check-2-g3-DanielAlors' (Oct 16, 2024), 'control-check-1-g3-DanielAlors' (Dec 20, 2023), and 'control-check-1-g2-DanielAlors' (Nov 13, 2023). It includes sections for 'AUTHORIZED ACCOUNTS & ORGS' (listing DanielAlors, benjimrfl, gii-is-DPI, and rubpergar) and 'Configure in GitHub'.

CD

The screenshot shows the 'Settings' tab in the Render service configuration. It includes sections for 'Events', 'Logs', 'Metrics', 'Disks', 'Environment', 'Shell', 'Previews', and 'Jobs'. The main configuration area has sections for 'Pre-Deploy Command' (optional, useful for database migrations), 'Auto-Deploy' (disabled by default), and 'Deploy hook' (private URL for triggering deployment).

Auto-deploy=NO

Deploy hook copiar y poner en github→Tu repositorio→Settings→Secrets and variables→Actions→new repository secret (Poner nombre y en descripción pegar el deploy hook)

Render flujo de trabajo de despliegue continuo

En el `.github/workflows` carpeta que tienes que añadir lo siguiente `render.yml`.

```
name: Deploy to Render

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  deploy:
    name: Deploy to Render
    runs-on: ubuntu-latest
    steps:
      - name: Check out the repo
        uses: actions/checkout@v4

      - name: Deploy to Render
        env:
          deploy_url: ${{ secrets.RENDER_DEPLOY_HOOK_URL }}
        run:
          curl "$deploy_url"
```

> **1.-** Permitir que se realice el despliegue solo si un *pull request* ha sido aprobado y mergeado

> **2.-** Permitir que se realice el despliegue solo si todas las pruebas han pasado

> **3.-** Realizar un despliegue solo cuando se crea un *tag* en GitHub

Ejercicios CD

```
name: Deploy to Render
on:
  push:
    tags:
      - 'v*'
  pull_request:
    branches:
      - main
jobs:
  testing:
    name: Run Tests
    runs-on: ubuntu-latest
    services:
      mysql:
        image: mysql:5.7
        env:
          MYSQL_ROOT_PASSWORD: uvlhub_root_password
          MYSQL_DATABASE: uvlhubdb_test
          MYSQL_USER: uvlhub_user
          MYSQL_PASSWORD: uvlhub_password
        ports:
          - 3306:3306
        options: --health-cmd="mysqladmin ping" --health-interval=10s --health-timeout=5s --health-retries=3
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: '3.12'
      - name: Install dependencies
        run:
          python -m pip install --upgrade pip
          pip install -r requirements.txt
      - name: Run Tests
        env:
          FLASK_ENV: testing
          MARIADB_HOSTNAME: 127.0.0.1
          MARIADB_PORT: 3306
          MARIADB_TEST_DATABASE: uvlhubdb_test
          MARIADB_USER: uvlhub_user
          MARIADB_PASSWORD: uvlhub_password
        run:
          pytest app/modules/ --ignore-glob='*selenium*'
  deploy:
    name: Deploy to Render
    needs: testing
    runs-on: ubuntu-latest
    steps:
      - name: Check out the repo
        uses: actions/checkout@v4
      - name: Deploy to Render
        env:
          deploy_url: ${{ secrets.RENDER_DEPLOY_HOOK_URL }}
        run:
          curl "$deploy_url"
```