Here's how to create the test environment using GitHub's web interface and an IDE:

1. Repository Setup:

* Go to GitHub.com and click "New Repository"
* Name it "code-analysis-test"
* Choose "Public" visibility
* Click "Create Repository"

1. Local Development:

* Open your preferred IDE (like VS Code or IntelliJ)
* Clone the new repository:

git clone <https://github.com/allanbrunobr/code-analysis-test.git>

cd code-analysis-test

1. Create Initial File:

* Create test.py in your IDE
* Add the code: print("hello world")
* Commit and push to main:

git add test.py

git commit -m "Initial commit"

git push origin main

1. Feature Branch Development:

* Create and switch to new branch:

git checkout -b feature/code-analysis

* Modify test.py to: print("hello world from feature branch")
* Commit and push changes:

git add test.py

git commit -m "Update test.py with new code"

git push origin feature/code-analysis

1. Create Pull Request:

* Go to GitHub repository
* Click "Pull Requests"
* Click "New Pull Request"
* Select base:main and compare:feature/code-analysis
* Add title "Add code analysis feature"
* Add description "This PR adds code analysis functionality"
* Click "Create Pull Request"

1. Test Integration:

* Use Google Cloud Console or gcloud CLI to publish the test message:

gcloud pubsub topics publish code-to-analyze --message='{

"name": "bruno",

"email": "allanbruno@gmail.com",

"code": "print(\"hello world from feature branch\")",

"token": "YOUR\_GITHUB\_TOKEN",

"language": "python",

"prompt": "analyze code",

"repository": {

"type": "Github",

"owner": "allanbrunobr",

"repo": "code-analysis-test",

"pull\_request\_number": "1",

"project\_id": "code-analysis-test",

"repo\_slug": "code-analysis-test",

"pull\_request\_id": "1",

"workspace": "code-analysis-test"

}

}'

## Análise do código

Analisando o código fornecido, identifiquei alguns pontos que podem ser melhorados:

### Refatoração

O código em si é simples e funcional, mas pode ser refatorado para melhorar a legibilidade e a manutenibilidade.

Sugiro a alteração do código para:

```python

def print\_hello\_world():

"""Imprime a mensagem 'Olá, mundo!' no console."""

print("Olá, mundo!")

if \_\_name\_\_ == "\_\_main\_\_":

print\_hello\_world()

```

### Princípios SOLID

O código segue o princípio da Responsabilidade Única (SRP), pois possui apenas uma função que imprime a mensagem "Olá, mundo!".

No entanto, não segue os demais princípios SOLID:

\* \*\*O\*\*: O código não possui nenhuma abstração.

\* \*\*L\*\*: O código não apresenta baixo acoplamento, pois a função `print\_hello\_world` está diretamente acoplada à saída do console.

\* \*\*I\*\*: O código não é imutável, pois a função `print\_hello\_world` modifica o estado do console.

\* \*\*D\*\*: O código não é detectável, pois não possui nenhum teste unitário.

\* \*\*P\*\*: O código não é protegido contra alterações, pois não possui nenhuma validação de entrada.

### Segurança

O código não apresenta vulnerabilidades de segurança conhecidas.

### OWASP

O código não segue os fundamentos OWASP, pois não possui nenhuma validação de entrada.

### Considerações Finais

O código é simples e funcional, mas pode ser refatorado e melhorado para se alinhar com os princípios SOLID e OWASP.

Recomendo a adição de testes unitários para garantir a qualidade do código e a implementação de validação de entrada para evitar vulnerabilidades de segurança.