PYTHON E GITHUB ACTIONS PER AUTOMATIZZARE IL TUO BLOG

Andrea Grillo @ PyCon Italia 2022

ANDREA GRILLO
@andregri

andrea.grillo96@gmail.com

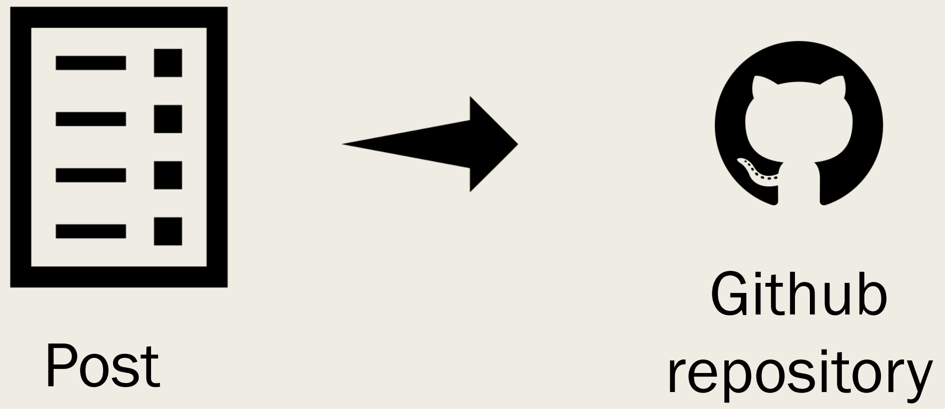
https://github.com/andregri

Python

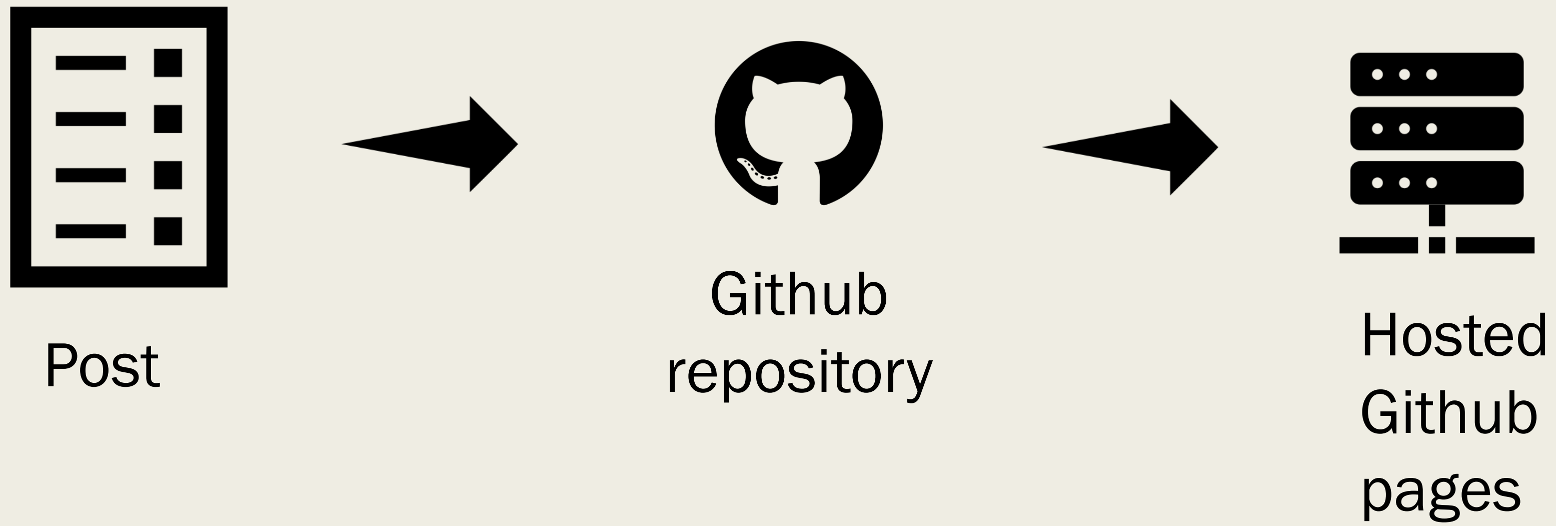
DevOps

Cloud

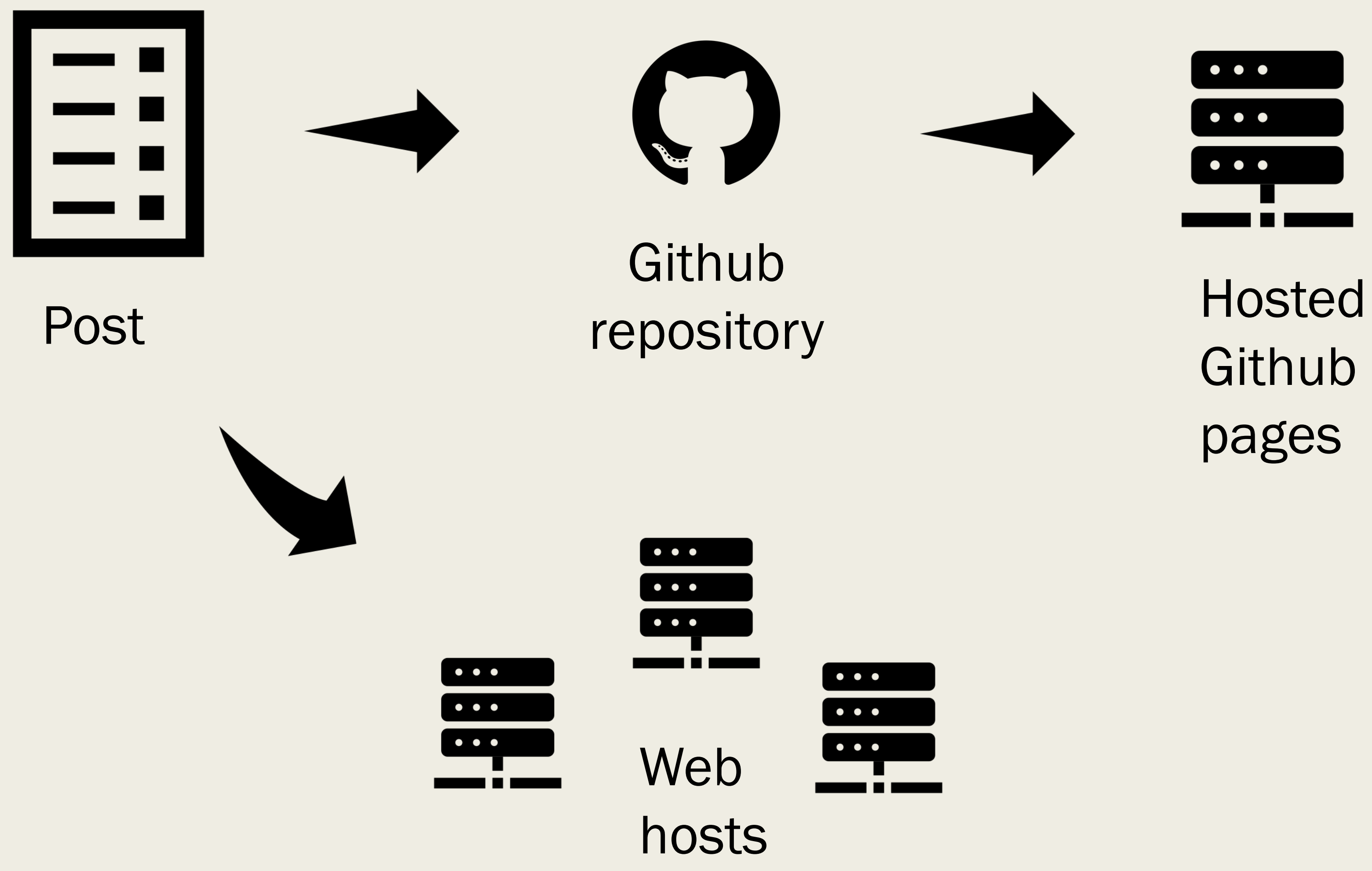
Il problema



Il problema

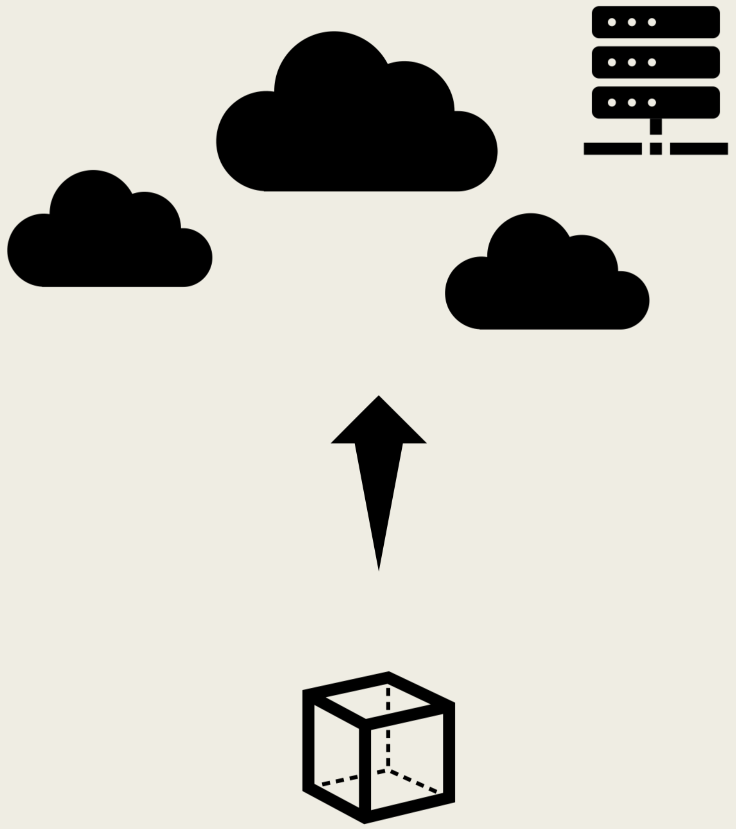


Il problema



Casi reali

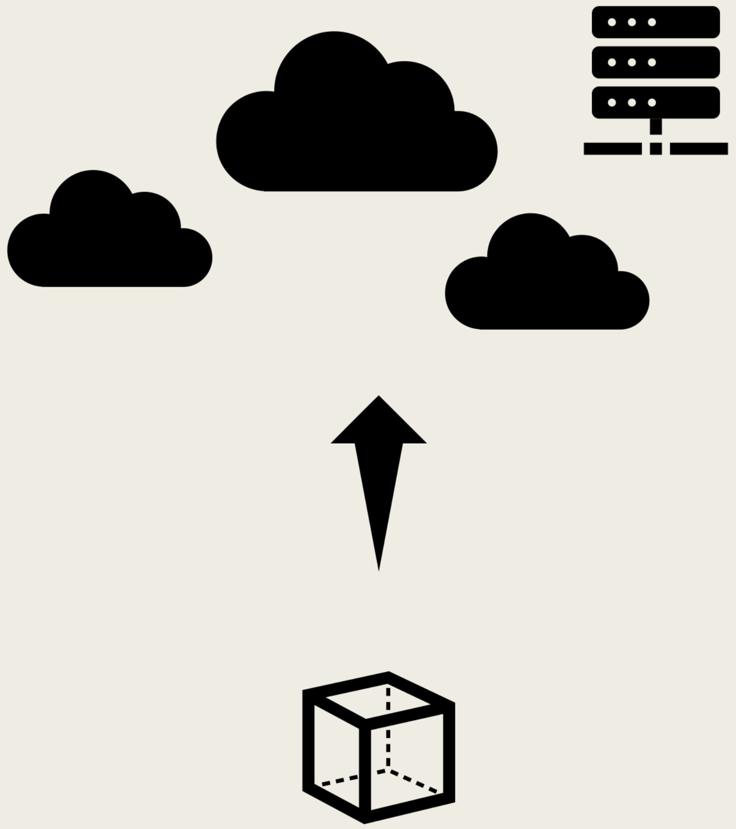
Automatico



Casi reali

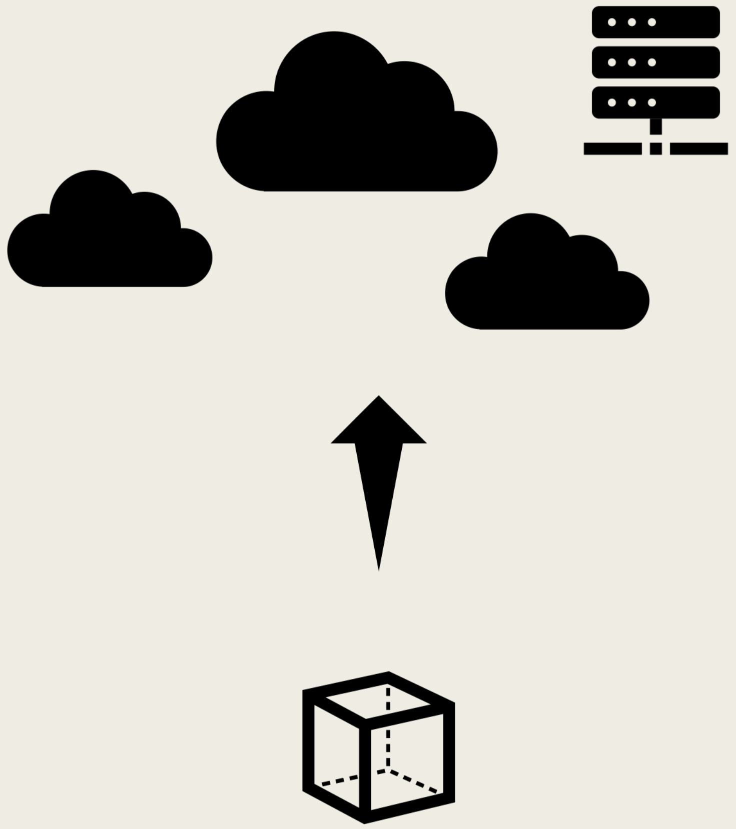
Automatico

Disaster recovery



Casi reali

- Automatico
- Disaster recovery
- Push vs Pull



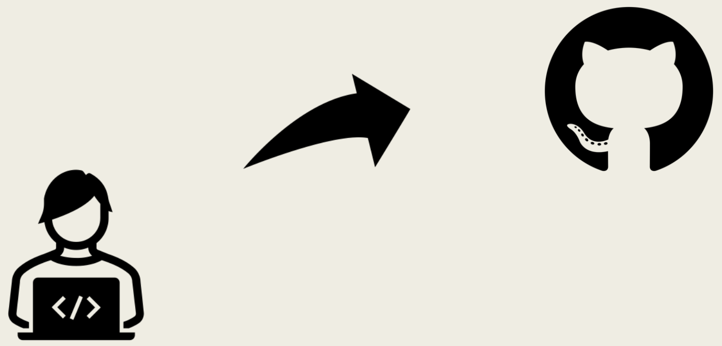
TRE POSSIBILI SOLUZIONI

SOLUZIONE #1

1. **Web scraping** della pagina web

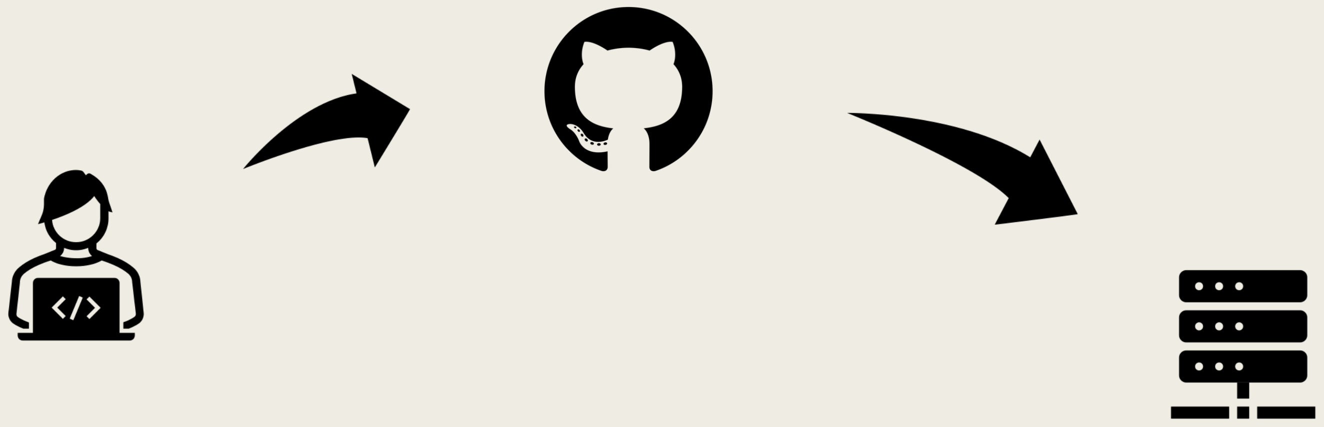
SOLUZIONE #1

1 **Web scraping** della pagina web



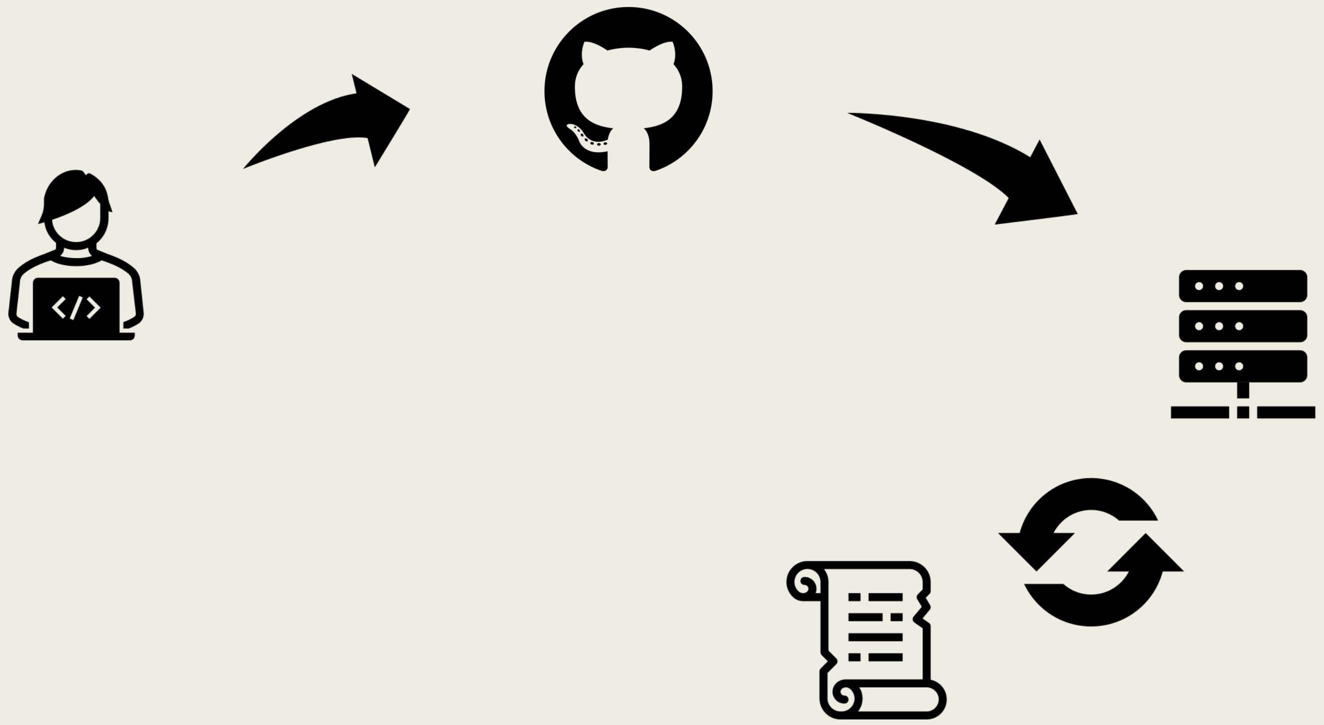
SOLUZIONE #1

1 **Web scraping** della pagina web



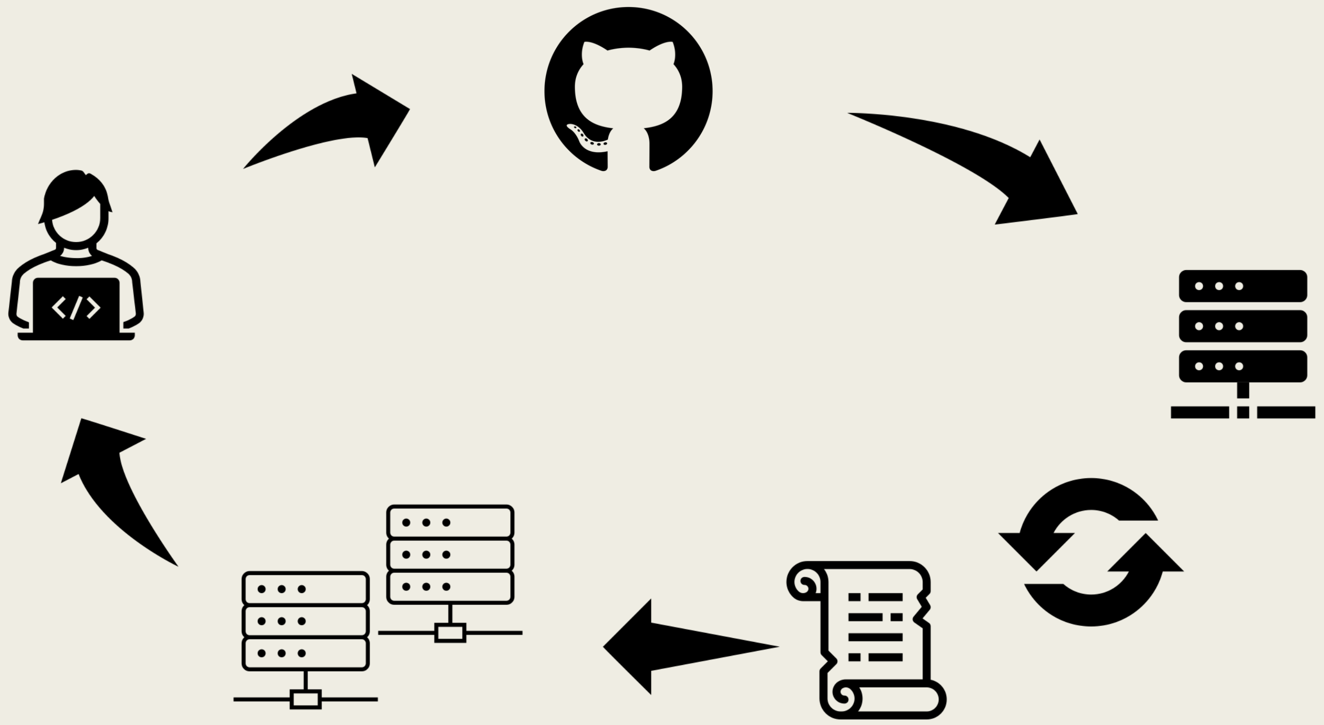
SOLUZIONE #1

1 **Web scraping** della pagina web

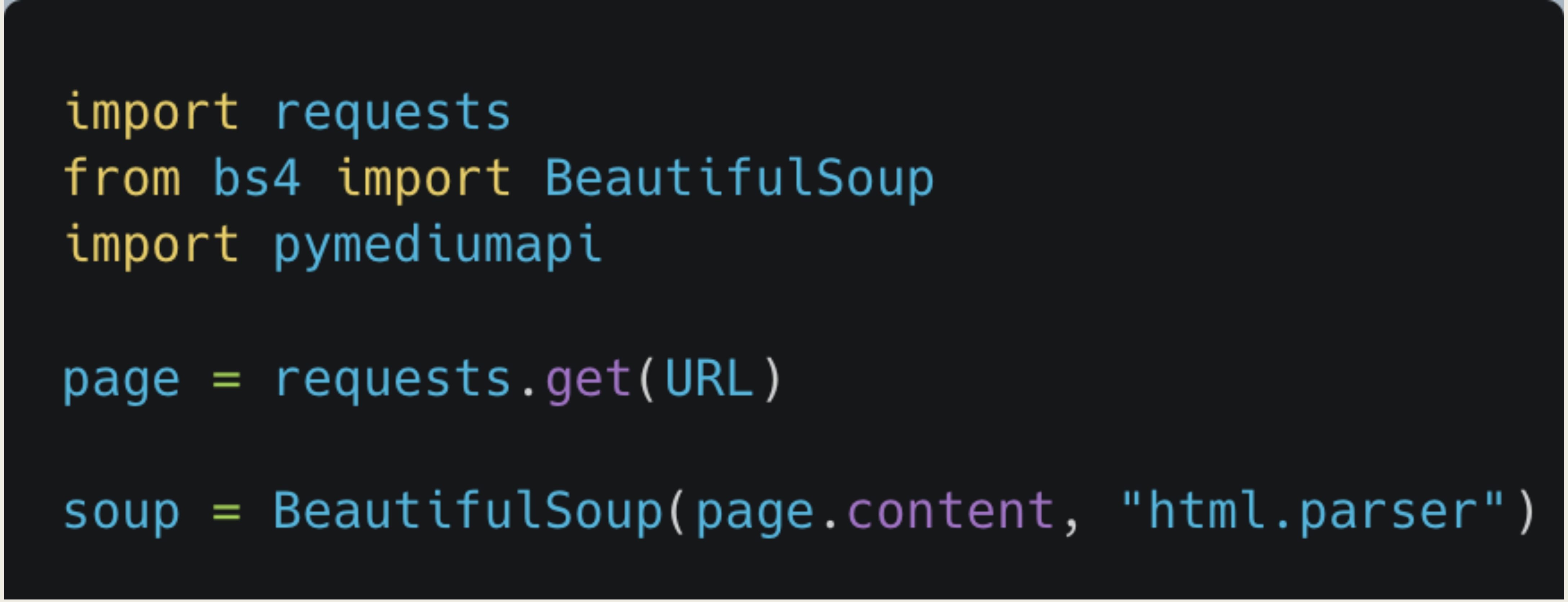


SOLUZIONE #1

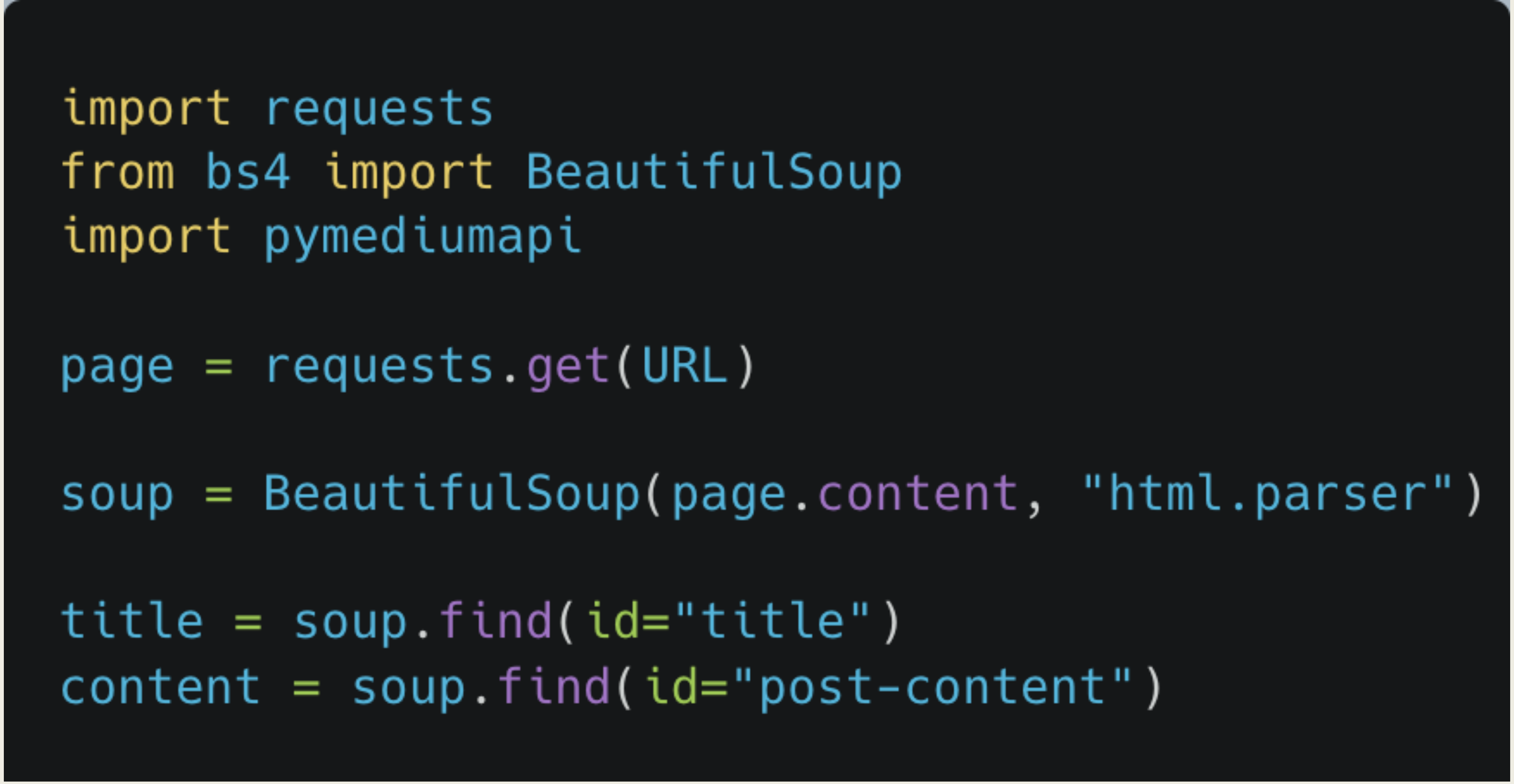
1 **Web scraping** della pagina web



```python
import requests
from bs4 import BeautifulSoup
import pymediumapi
import requests
from bs4 import BeautifulSoup
import pymediumapi

page = requests.get(URL)
```



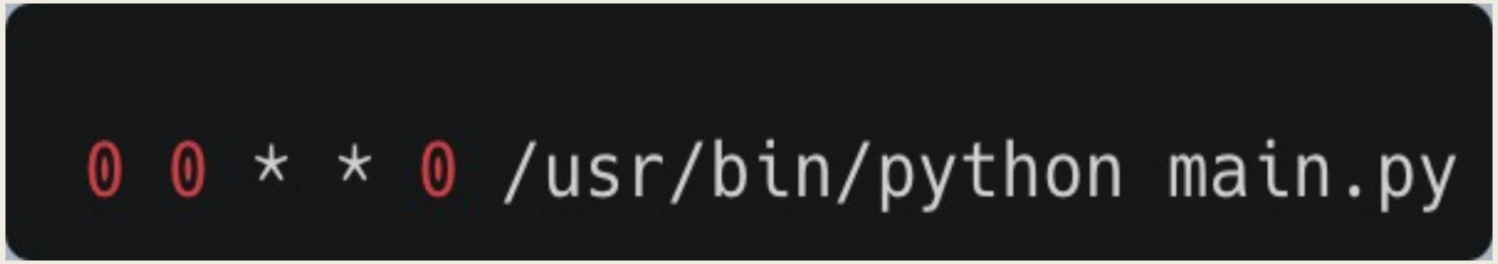
```python
import requests
from bs4 import BeautifulSoup
import pymediumapi

page = requests.get(URL)

soup = BeautifulSoup(page.content, "html.parser")
```



```python
import requests
from bs4 import BeautifulSoup
import pymediumapi

page = requests.get(URL)

soup = BeautifulSoup(page.content, "html.parser")

title = soup.find(id="title")
content = soup.find(id="post-content")
```

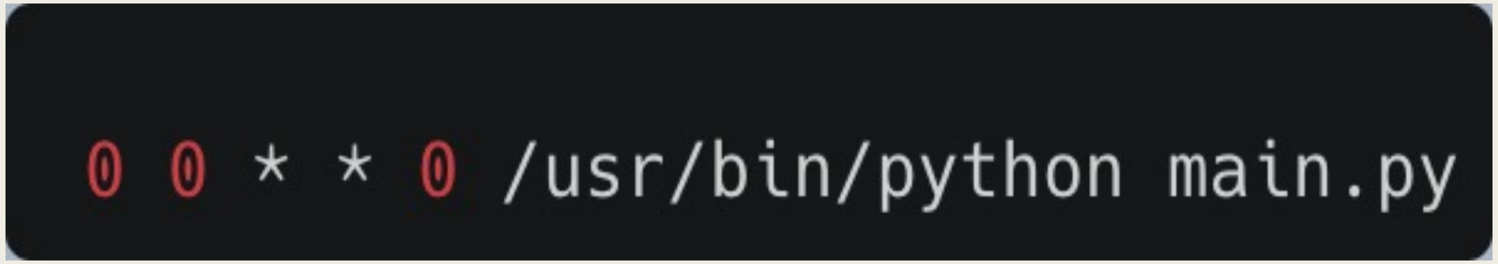
```python
import requests
from bs4 import BeautifulSoup
import pymediumapi

page = requests.get(URL)

soup = BeautifulSoup(page.content, "html.parser")

title = soup.find(id="title")
content = soup.find(id="post-content")

client = pymediumapi.Client(token)
client.authenticate()
client.create_post(title=title, content=content)
```

Crontab *“At 00:00 on Sunday”*



```
0 0 * * 0 /usr/bin/python main.py
```

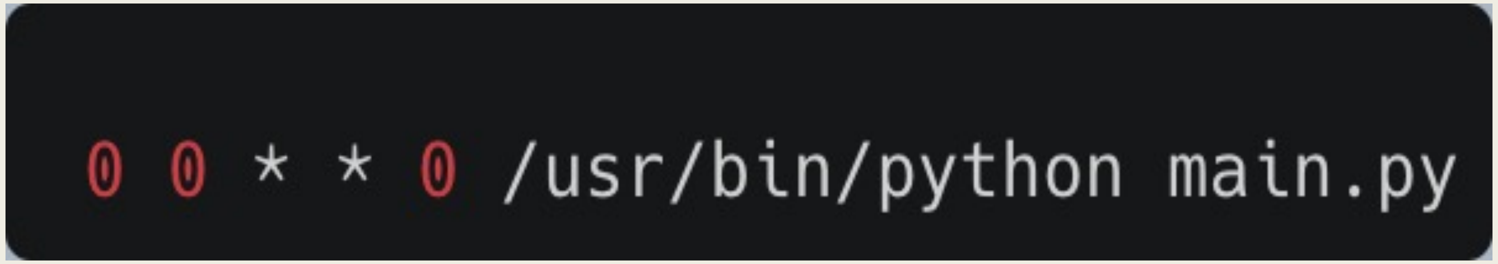
***Crontab** "At 00:00 on Sunday"*



```
0 0 * * 0 /usr/bin/python main.py
```

Contenuti STATICI, complesso per contenuti DINAMICI

Crontab *"At 00:00 on Sunday"*

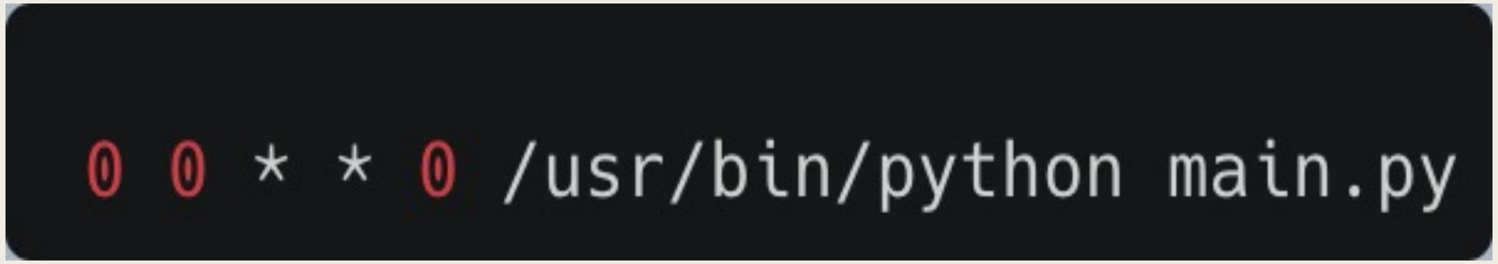


```
0 0 * * 0 /usr/bin/python main.py
```

Contenuti STATICI, complesso per contenuti DINAMICI

TARGET: Github Pages

Crontab “At 00:00 on Sunday”



```
0 0 * * 0 /usr/bin/python main.py
```

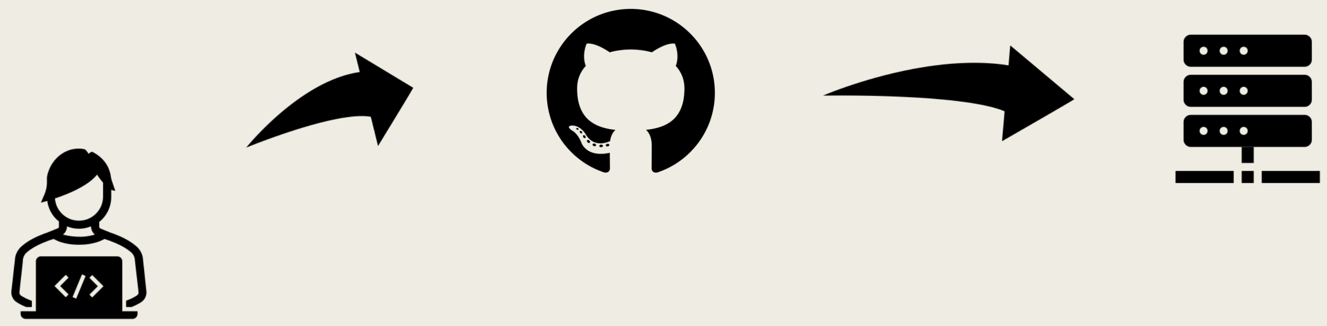
Contenuti STATICI, complesso per contenuti DINAMICI

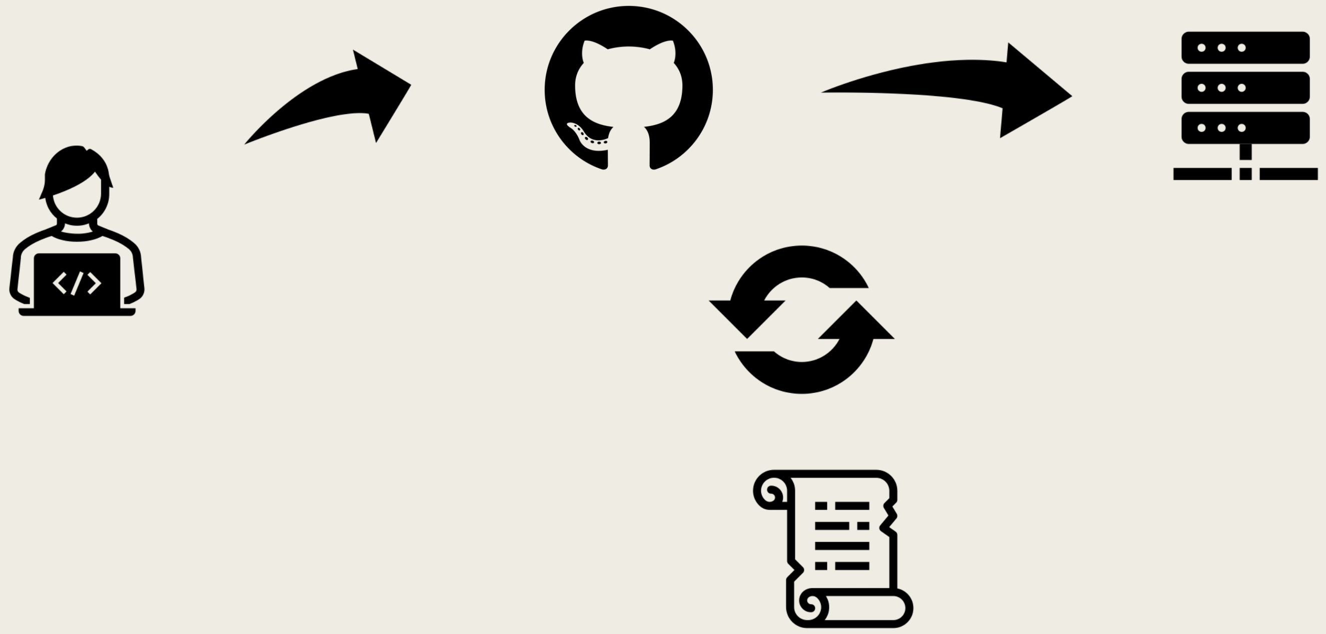
TARGET: Github Pages

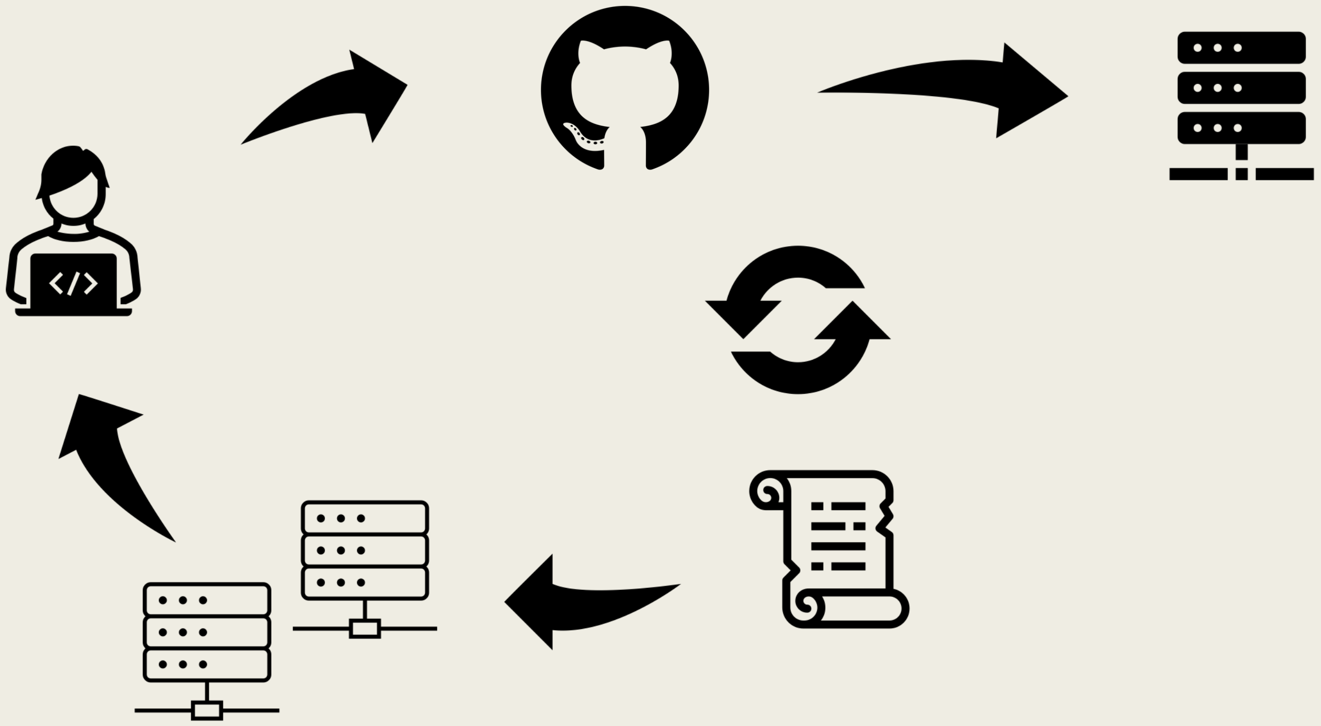
Non riutilizzabile

SOLUZIONE #2

1. **Web scraping** della pagina web
2. Scraping della repository tramite API di GitHub

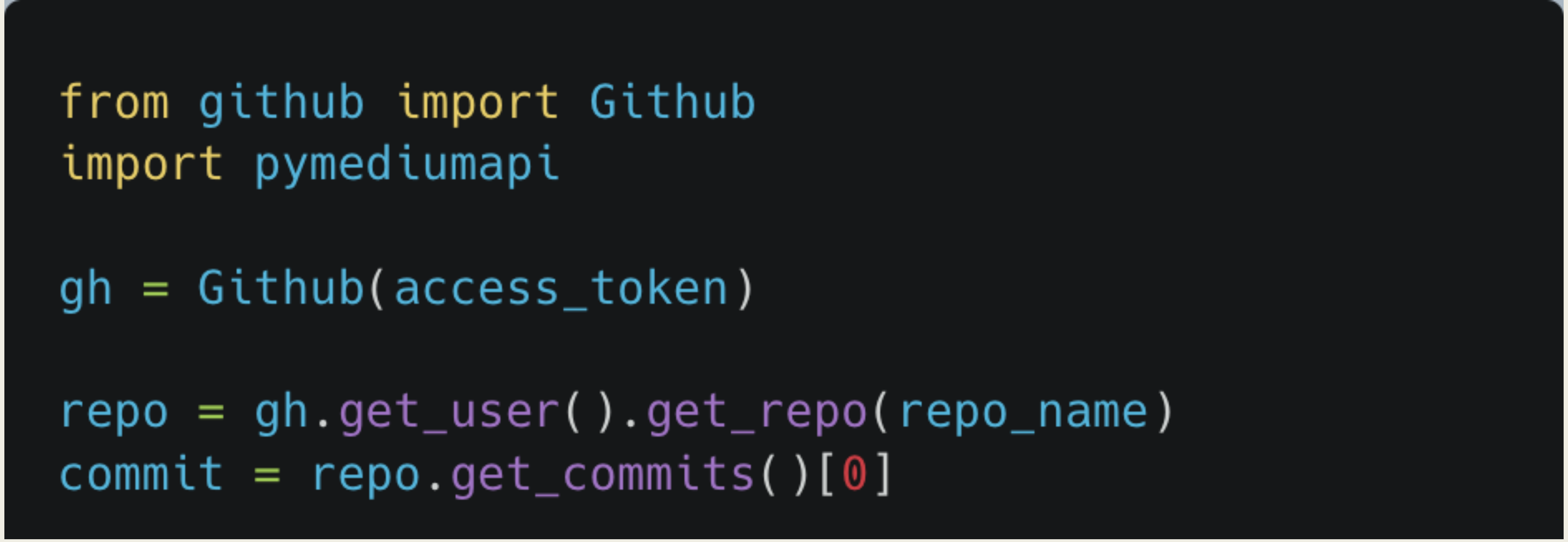




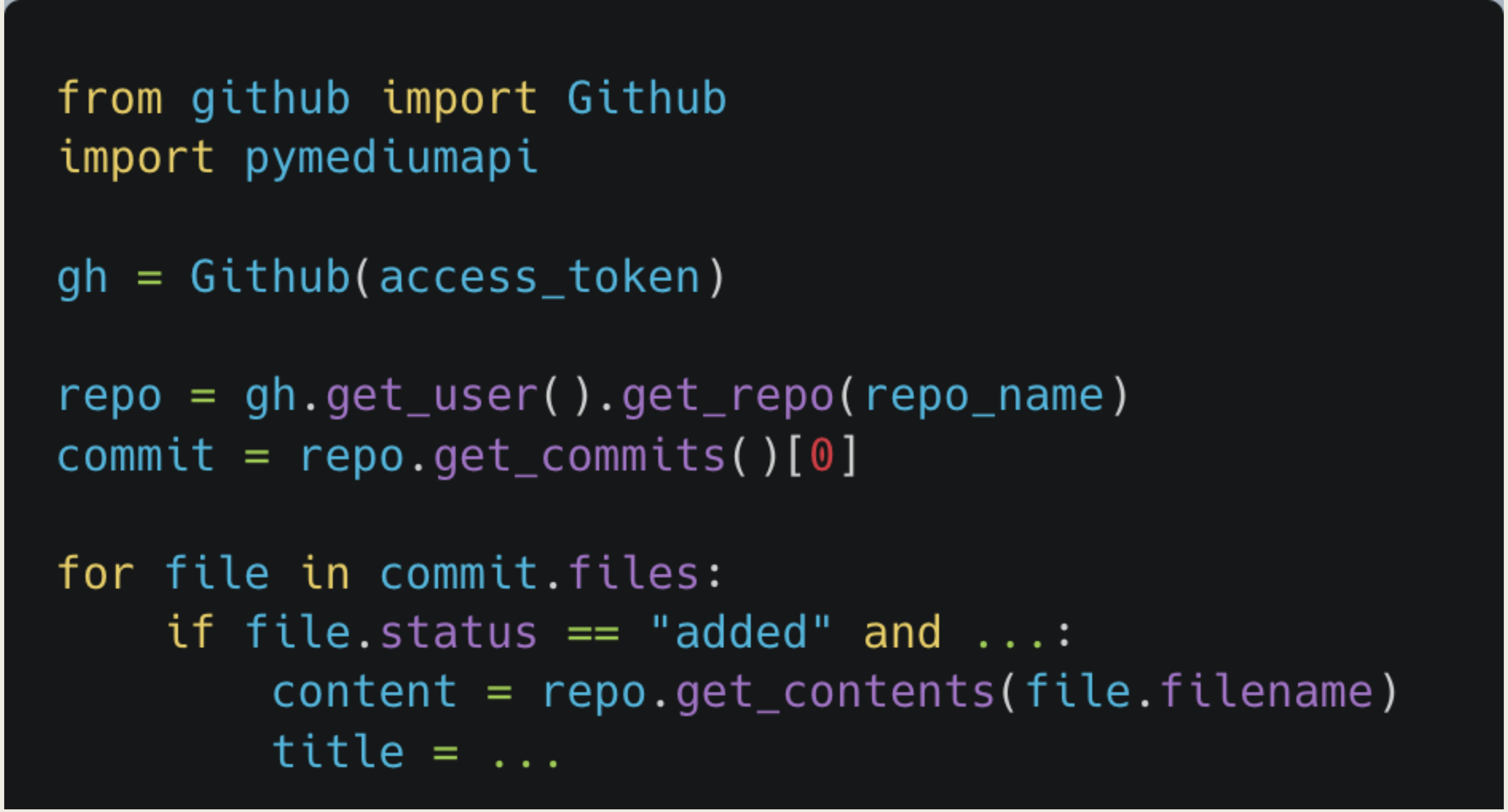


```
from github import Github
import pymediumapi
```

```python
from github import Github
import pymediumapi

gh = Github(access_token)
```



```python
from github import Github
import pymediumapi

gh = Github(access_token)

repo = gh.get_user().get_repo(repo_name)
commit = repo.get_commits()[0]
```



```python
from github import Github
import pymediumapi

gh = Github(access_token)

repo = gh.get_user().get_repo(repo_name)
commit = repo.get_commits()[0]

for file in commit.files:
    if file.status == "added" and ...:
        content = repo.get_contents(file.filename)
        title = ...
```

```python
from github import Github
import pymediumapi

gh = Github(access_token)

repo = gh.get_user().get_repo(repo_name)
commit = repo.get_commits()[0]

for file in commit.files:
    if file.status == "added" and ...:
        content = repo.get_contents(file.filename)
        title = ...

        client = pymediumapi.Client(token)
        client.authenticate()
        client.create_post(title=title, content=body)
```

Riutizzabile

Riutizzabile

TARGET: Repository

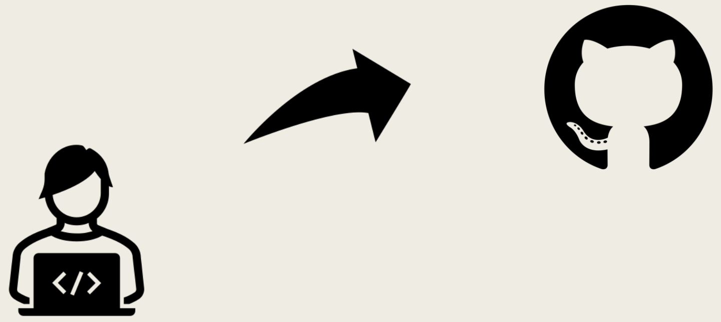
Riutizzabile

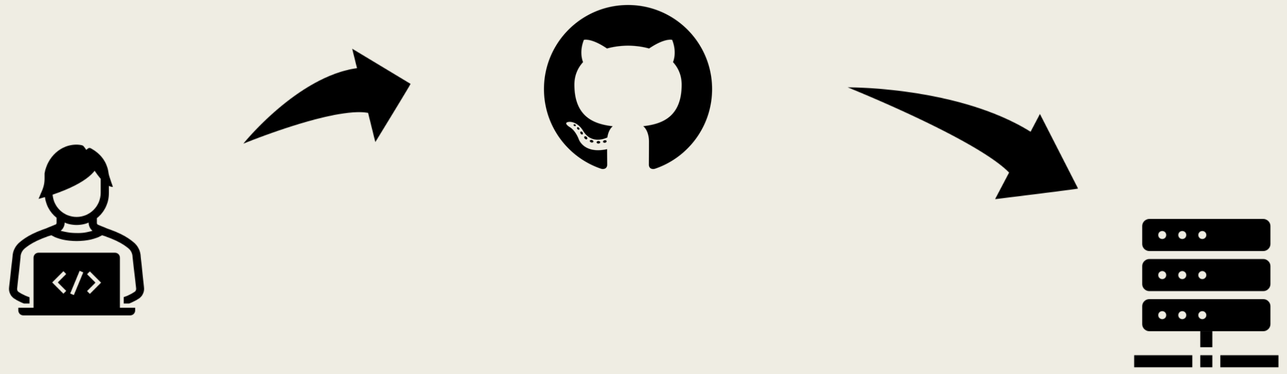
TARGET: Repository

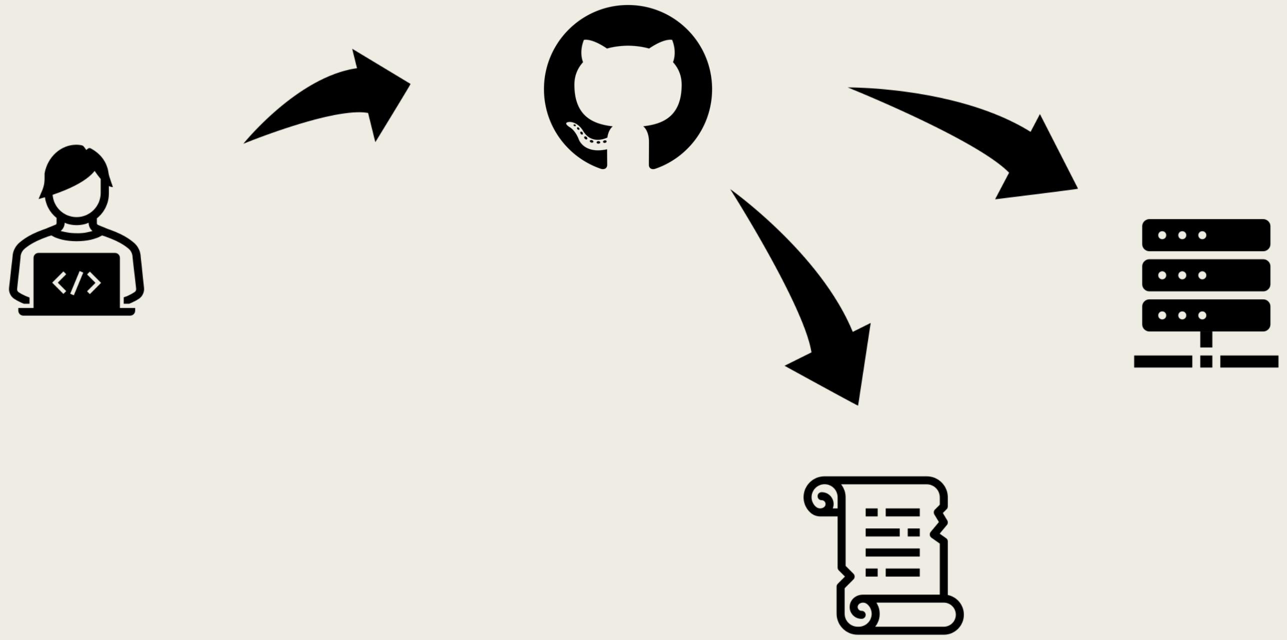
Crontab

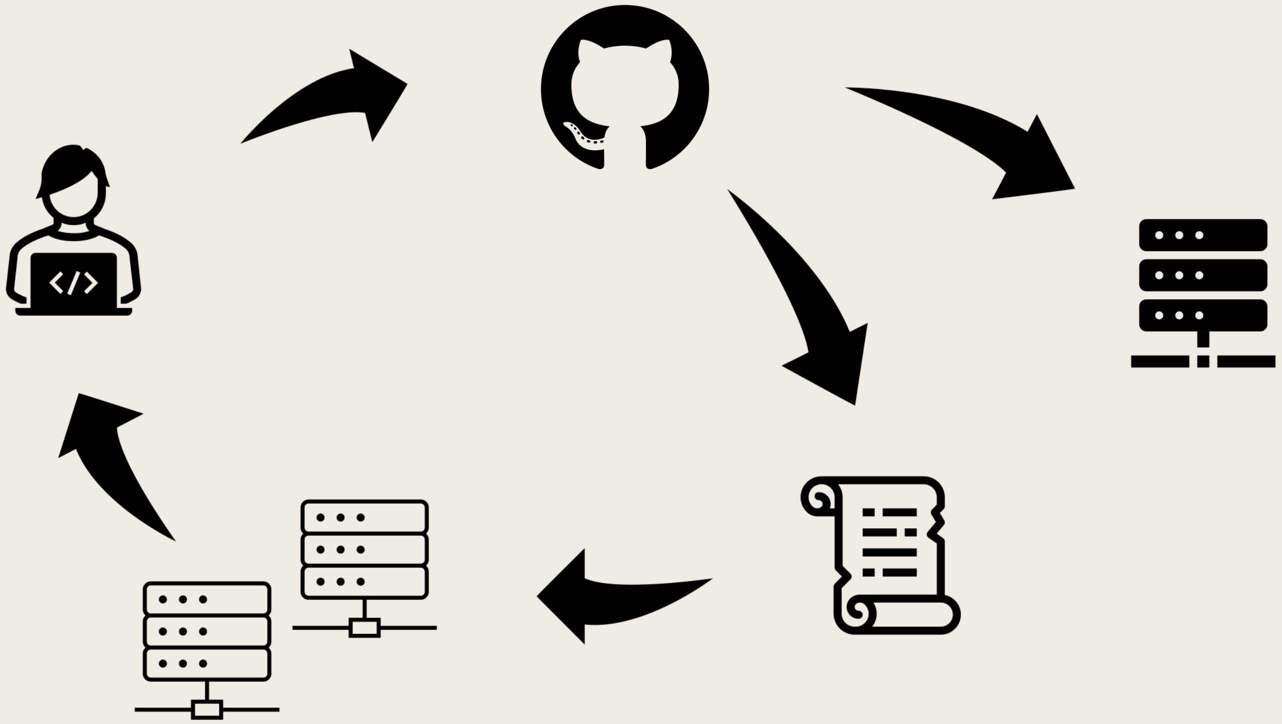
SOLUZIONE #3

1. **Web scraping** della pagina web
2. Scraping della repository tramite API di GitHub
3. Utilizzare lo script python in una **Github Action**



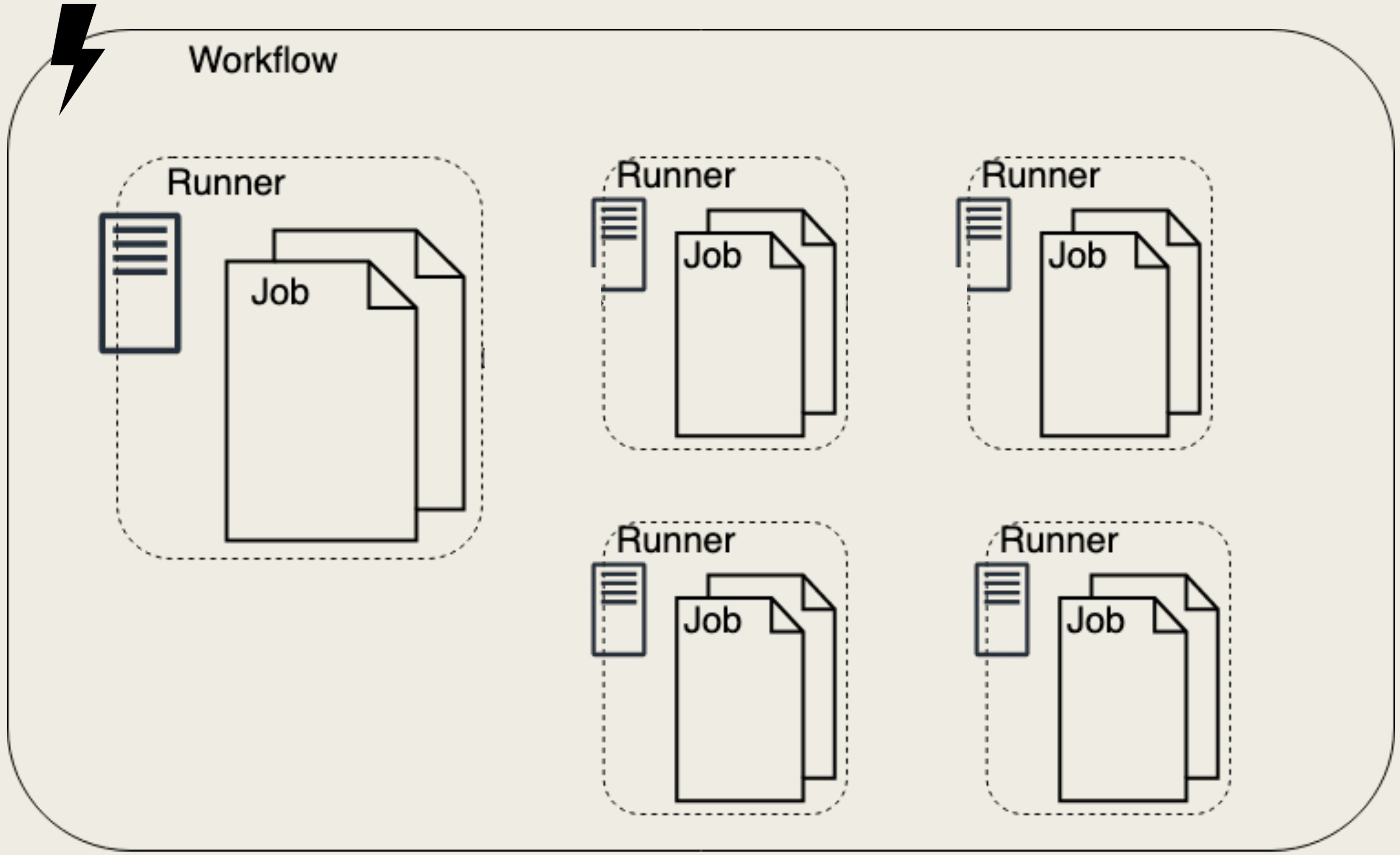


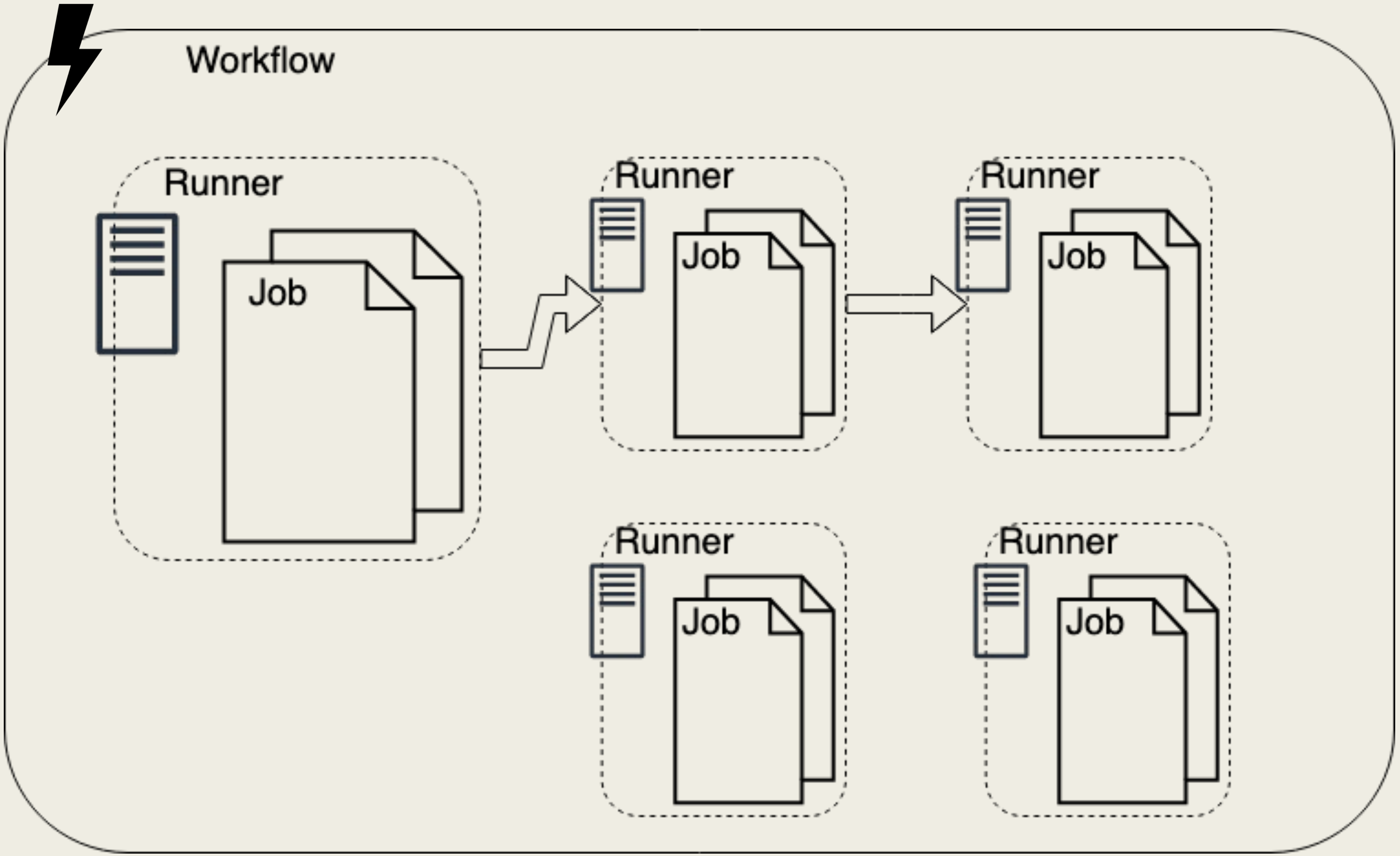


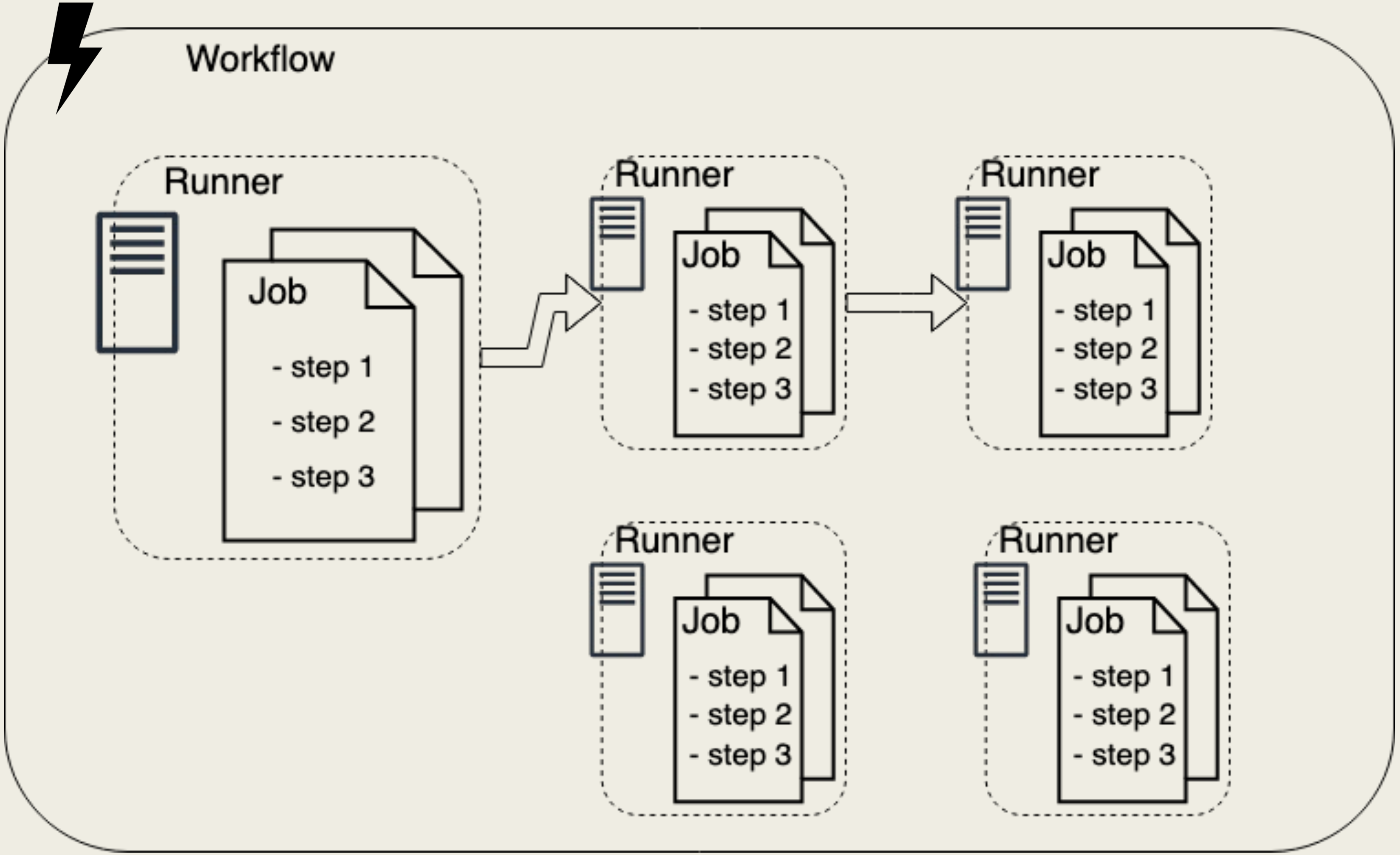


COS’È GITHUB ACTIONS?

Workflow


Workflow
Runner
Job
Runner
Job
Runner
Job
Runner
Job
Runner
Job


Workflow
Runner
Job
Runner
Job
Runner
Job
Runner
Job
Runner
Job


Workflow
Runner
Job
- step 1
- step 2
- step 3
Runner
Job
- step 1
- step 2
- step 3
Runner
Job
- step 1
- step 2
- step 3
Runner
Job
- step 1
- step 2
- step 3
Runner
Job
- step 1
- step 2
- step 3

Step vs Action

- STEP
  - Comando shell
  - `pip install –r requirements.txt`

Step vs Action

- STEP
  - Comando shell
  - `pip install –r requirements.txt`

- ACTION
  - ~ *Funzione*

Step vs Action

- **STEP**
  - Comando shell
  - `pip install –r requirements.txt`

- **ACTION**
  - ~ *Funzione*
  - Input

Step vs Action

- STEP
  - Comando shell
  - `pip install –r requirements.txt`

- ACTION
  - ~ *Funzione*
  - Input
  - Body: steps

Step vs Action

- STEP
  - Comando shell
  - `pip install –r requirements.txt`

- ACTION
  - ~ *Funzione*
  - Input
  - Body: steps
  - Output

COME USARE PYTHON IN GITHUB ACTIONS?

1. Direttamente nel *workflow*
2. ***Composite** Action*
3. ***Container** Action*

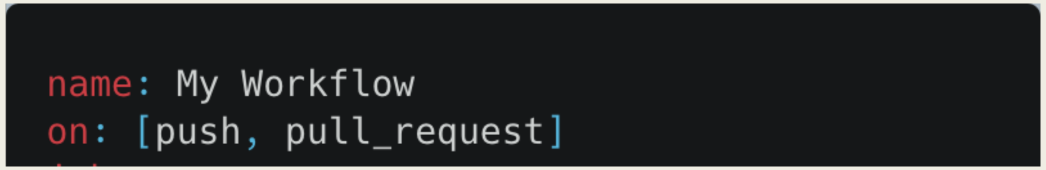
OPZIONE #1
Direttamente in un *workflow*

- `.github/workflows/`

OPZIONE #1

Direttamente in un *workflow*

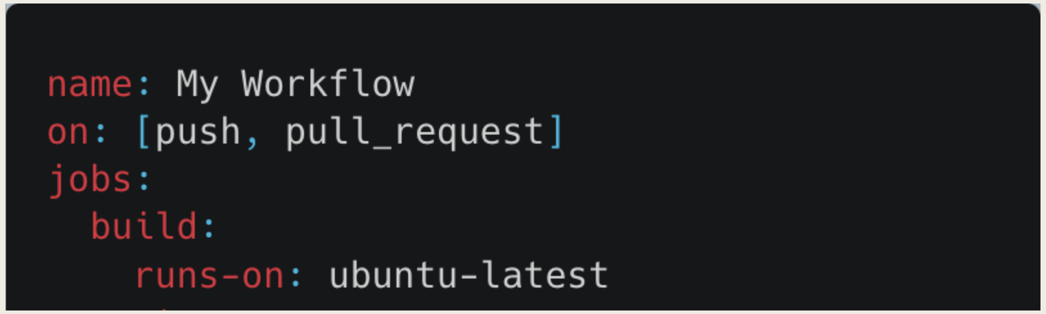
- `.github/workflows/`
- Eventi



```
name: My Workflow
on: [push, pull_request]
```

OPZIONE #1
Direttamente in un *workflow*

- `.github/workflows/`
- Eventi
- OS del *runner*

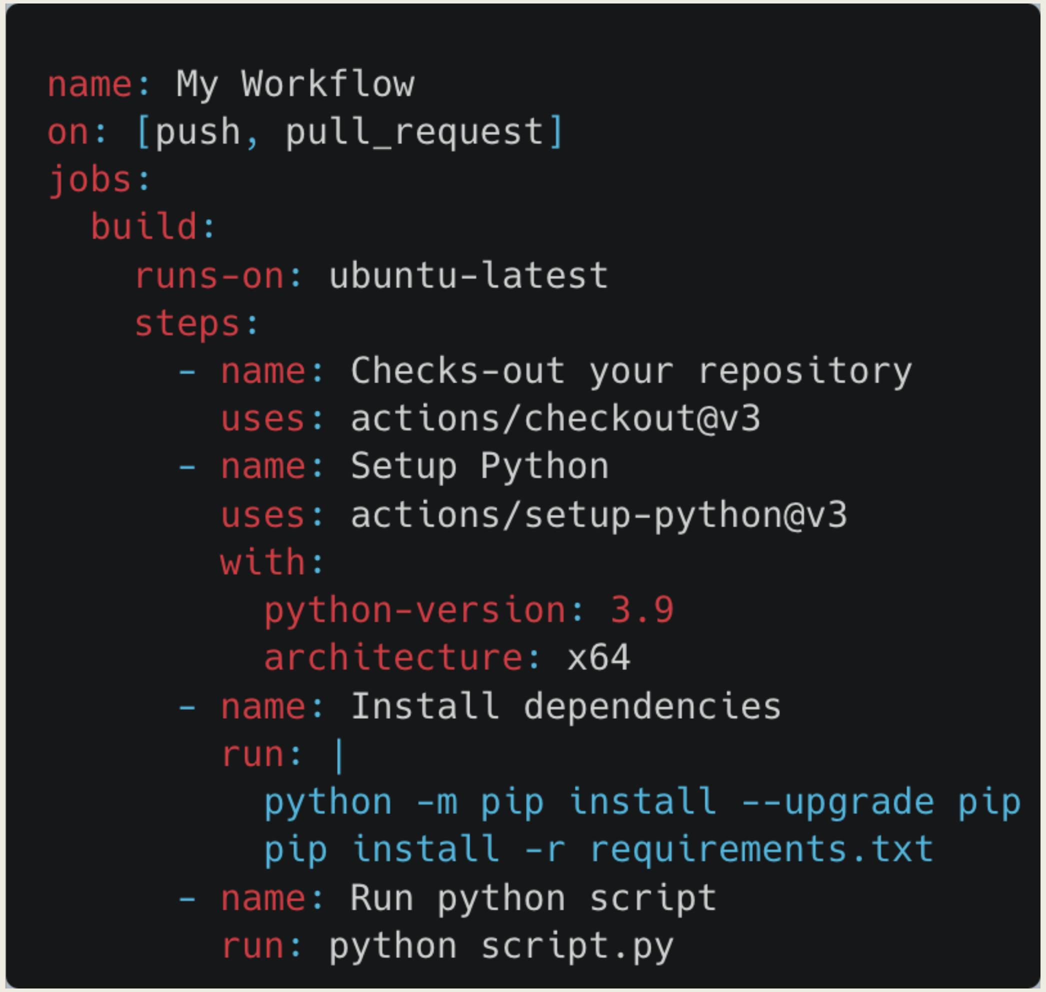


```yaml
name: My Workflow
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
```

OPZIONE #1

Direttamente in un *workflow*

- `.github/workflows/`
- Eventi
- OS del *runner*
- Lista di *steps*



```yaml
name: My Workflow
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checks-out your repository
        uses: actions/checkout@v3
      - name: Setup Python
        uses: actions/setup-python@v3
        with:
          python-version: 3.9
          architecture: x64
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
      - name: Run python script
        run: python script.py
```

Semplice

- Semplice
- Ogni *step* può contenere molte istruzioni da eseguire

Semplice

Ogni *step* può contenere molte istruzioni da eseguire

Non riutilizzabile in altri workflow

OPZIONE #2
Creare un *Composite Action*

- Dentro la **repository**:
  - ***`action.yml`***
  - *`main.py`*
  - *`requirements.txt`*

```yaml
name: 'BlogOps'
description: 'Automate blogging'
author: 'Andrea Grillo'
```

OPZIONE #2

Creare un *Composite Action*

- Dentro la **repository:**
  - ***action.yml***
  - *main.py*
  - *requirements.txt*

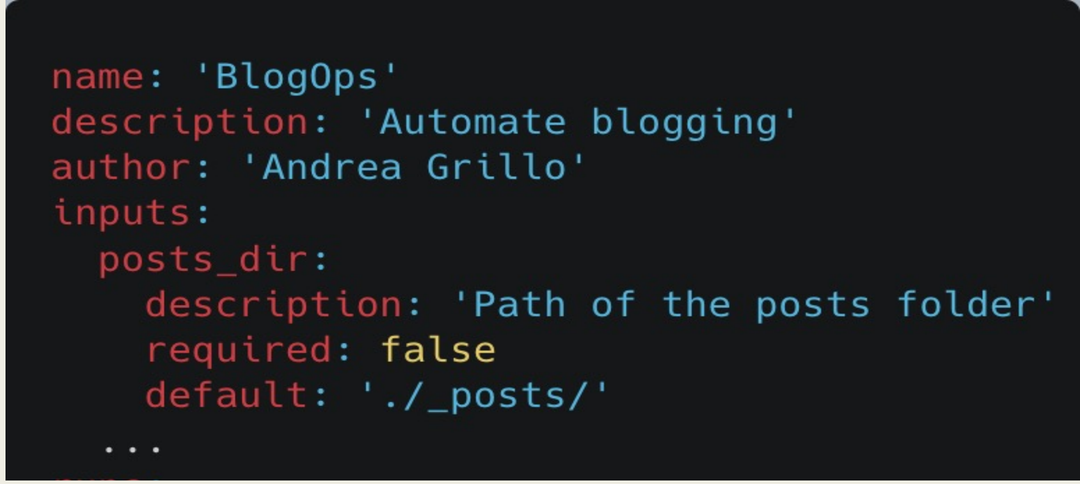
- Action:
  - *Metadati*

OPZIONE #2

Creare un *Composite Action*

- Dentro la **repository:**
  - ***action.yml***
  - *main.py*
  - *requirements.txt*

- Action:
  - *Metadati*
  - *Inputs*



```yaml
name: 'BlogOps'
description: 'Automate blogging'
author: 'Andrea Grillo'
inputs:
  posts_dir:
    description: 'Path of the posts folder'
    required: false
    default: './_posts/'
  ...
```

OPZIONE #2

Creare un *Composite Action*

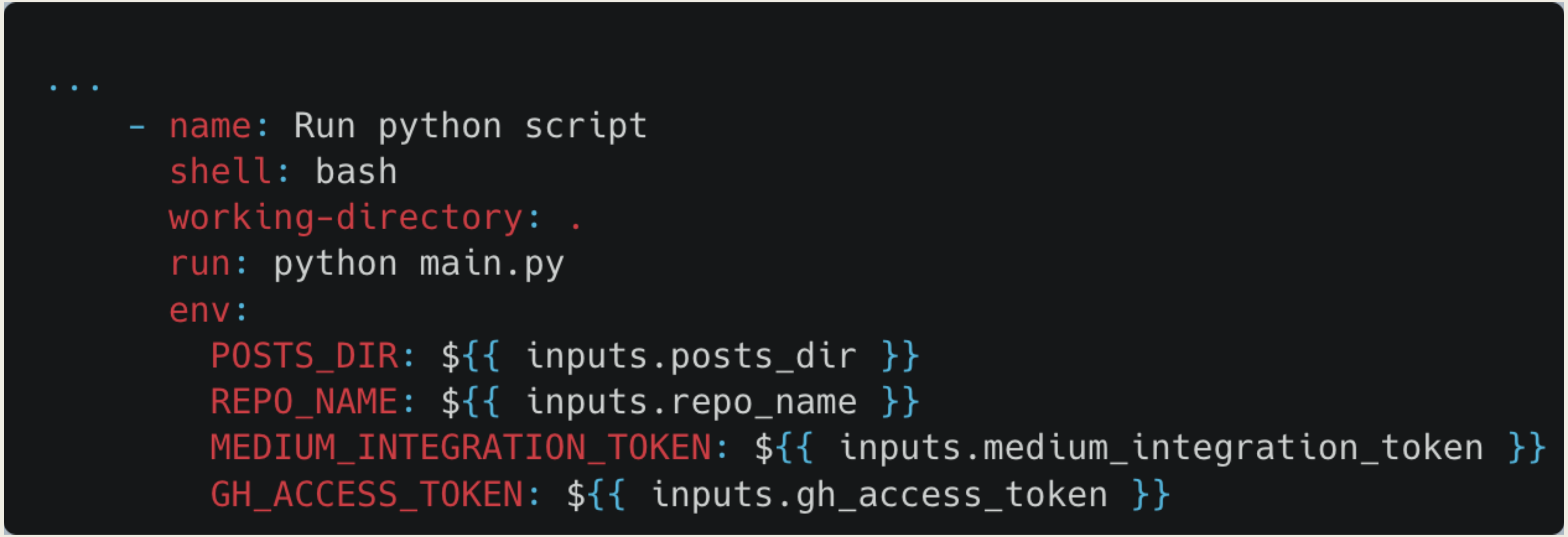
- Dentro la **repository:**
  - ***action.yml***
  - *main.py*
  - *requirements.txt*

- Action:
  - *Metadati*
  - *Inputs*
  - *steps*

```yaml
name: 'BlogOps'
description: 'Automate blogging'
author: 'Andrea Grillo'
inputs:
  posts_dir:
    description: 'Path of the posts folder'
    required: false
    default: './_posts/'
  ...
runs:
  using: 'composite'
  steps:
    - name: Setup Python
      uses: actions/setup-python@v3
      with:
        python-version: 3.9
        architecture: x64
    - name: Install dependencies
      shell: bash
      working-directory: .
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
    - name: Run python script
      shell: bash
      working-directory: .
      run: python main.py
      env:
        POSTS_DIR: ${{ inputs.posts_dir }}
        ...
```

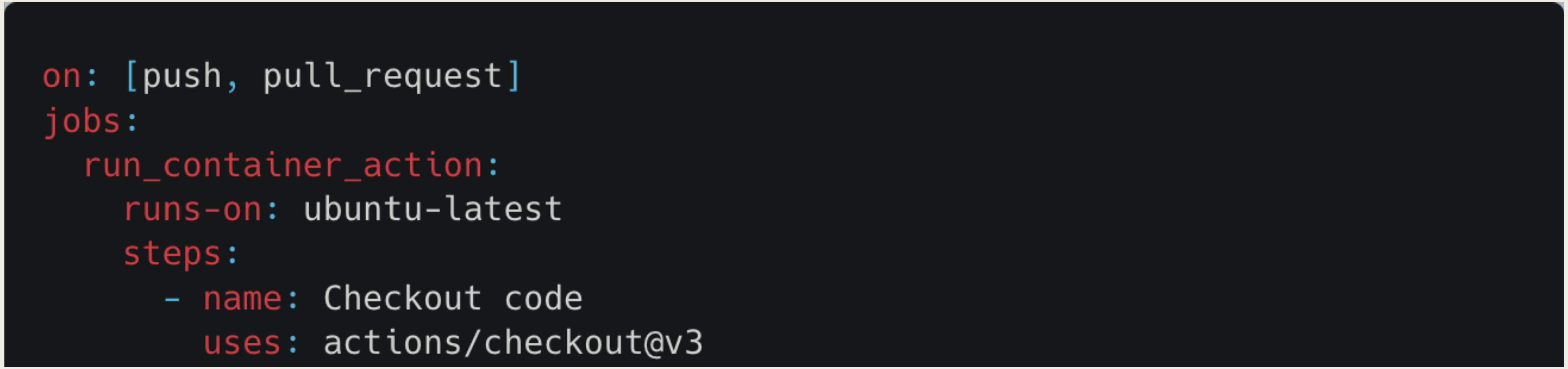
INPUT in una *Composite Action*

Step della composite *Action* che esegue lo script Python:



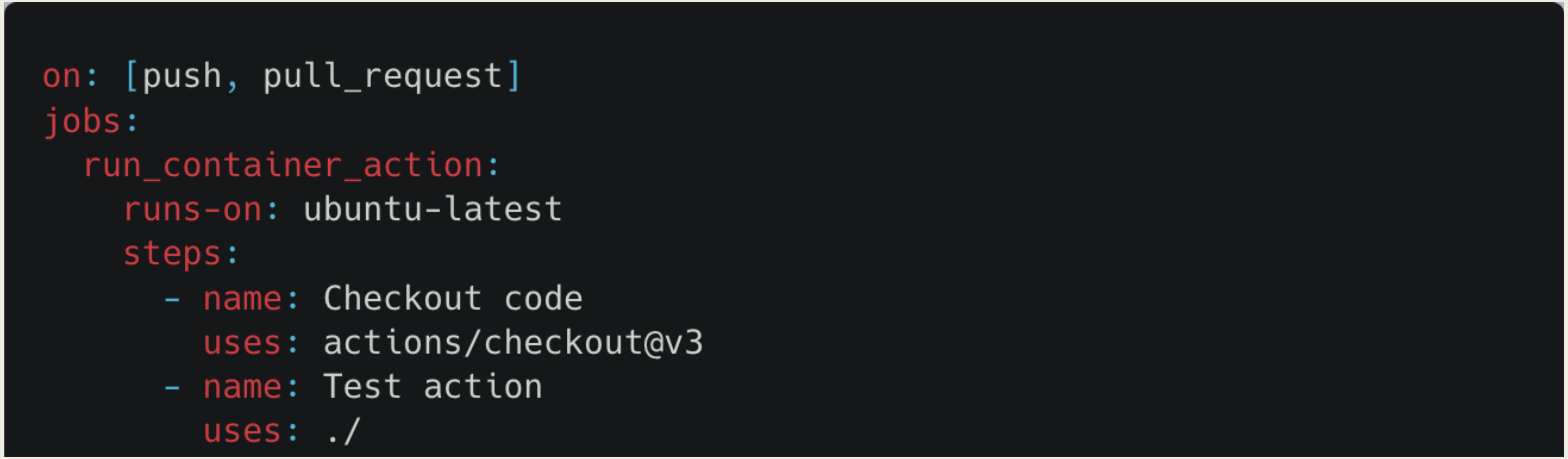
```yaml
...
    - name: Run python script
      shell: bash
      working-directory: .
      run: python main.py
      env:
        POSTS_DIR: ${{ inputs.posts_dir }}
        REPO_NAME: ${{ inputs.repo_name }}
        MEDIUM_INTEGRATION_TOKEN: ${{ inputs.medium_integration_token }}
        GH_ACCESS_TOKEN: ${{ inputs.gh_access_token }}
```

Usare l’action in un WORKFLOW



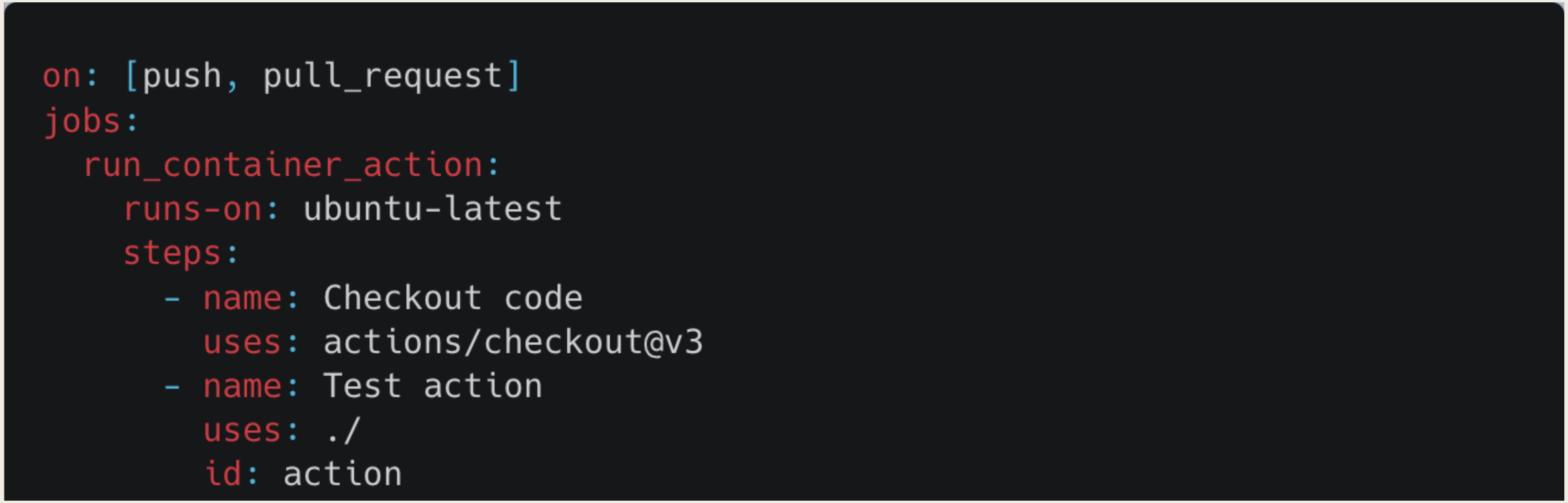
```yaml
on: [push, pull_request]
jobs:
  run_container_action:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
```

Usare l’action in un WORKFLOW



```yaml
on: [push, pull_request]
jobs:
  run_container_action:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      - name: Test action
        uses: ./
```

Usare l’action in un WORKFLOW



```yaml
on: [push, pull_request]
jobs:
  run_container_action:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      - name: Test action
        uses: ./
        id: action
```

Usare l’action in un WORKFLOW

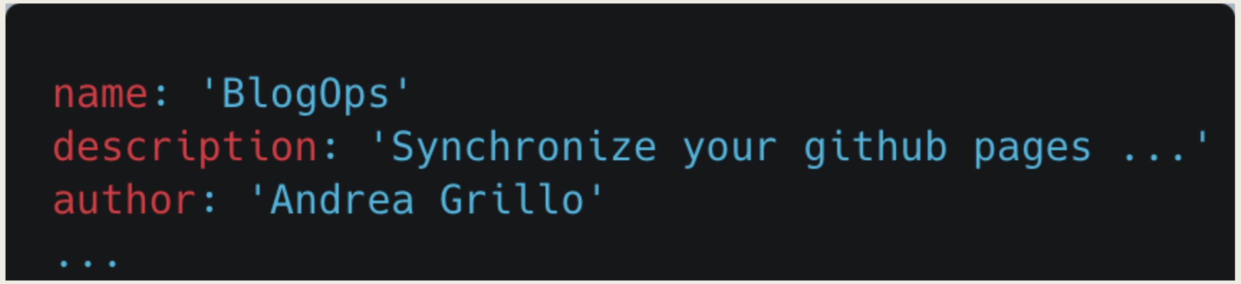
```yaml
on: [push, pull_request]
jobs:
  run_container_action:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      - name: Test action
        uses: ./
        id: action
        with:
          posts_dir: './tests/posts/'
          repo_name: "BlogOps"
          gh_access_token: ${{ secrets.GH_ACCESS_TOKEN }}
          medium_integration_token: ${{ secrets.MEDIUM_INTEGRATION_TOKEN }}
```

OPZIONE #3

Creare un *Container Action*

Dentro la repository:

- ***action.yml***
- *Files python*
- *Requirements*
- *Dockerfile*



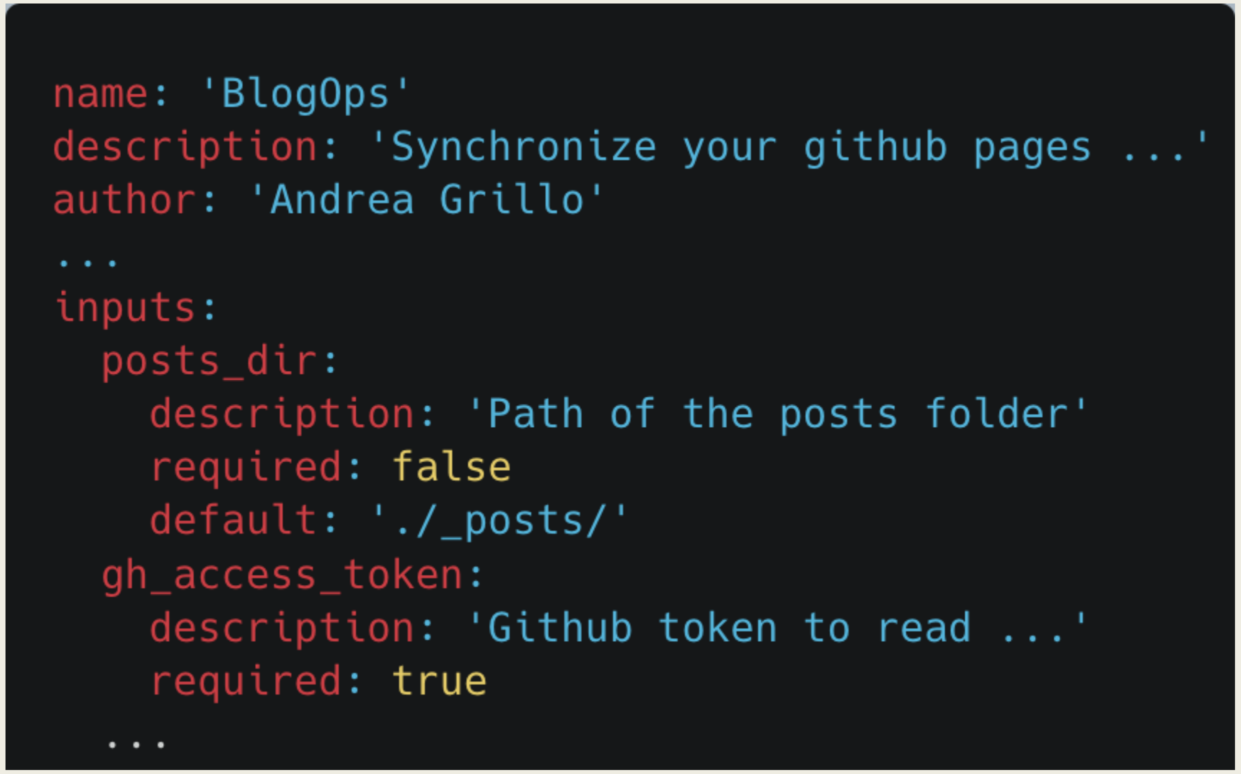
```yaml
name: 'BlogOps'
description: 'Synchronize your github pages ...'
author: 'Andrea Grillo'
...
```

OPZIONE #3

Creare un *Container Action*

Dentro la repository:

- ***action.yml***
- *Files python*
- *Requirements*
- *Dockerfile*



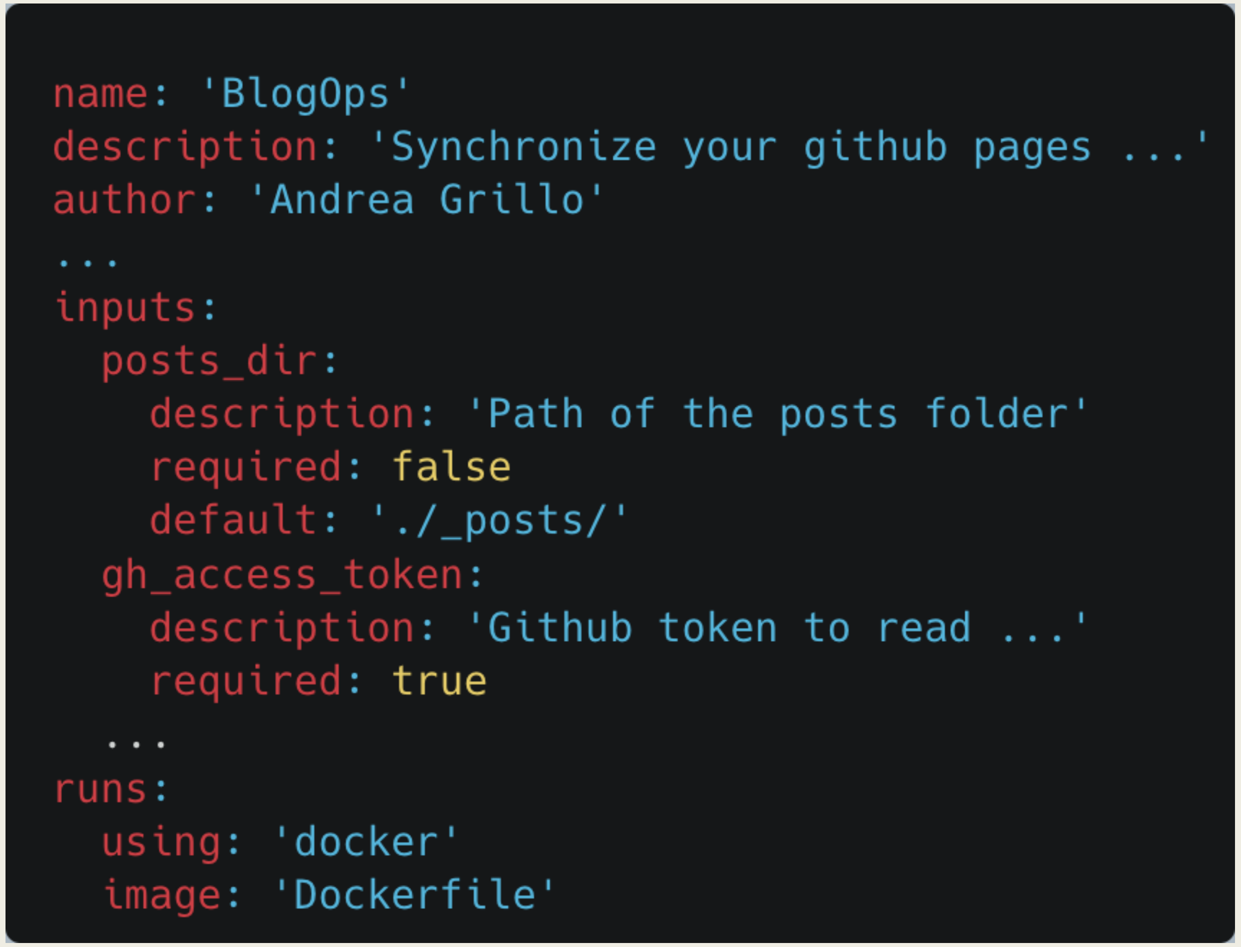
```yaml
name: 'BlogOps'
description: 'Synchronize your github pages ...'
author: 'Andrea Grillo'
...
inputs:
  posts_dir:
    description: 'Path of the posts folder'
    required: false
    default: './_posts/'
  gh_access_token:
    description: 'Github token to read ...'
    required: true
  ...
```

OPZIONE #3

Creare un *Container Action*

Dentro la repository:

- ***action.yml***
- *Files python*
- *Requirements*
- *Dockerfile*



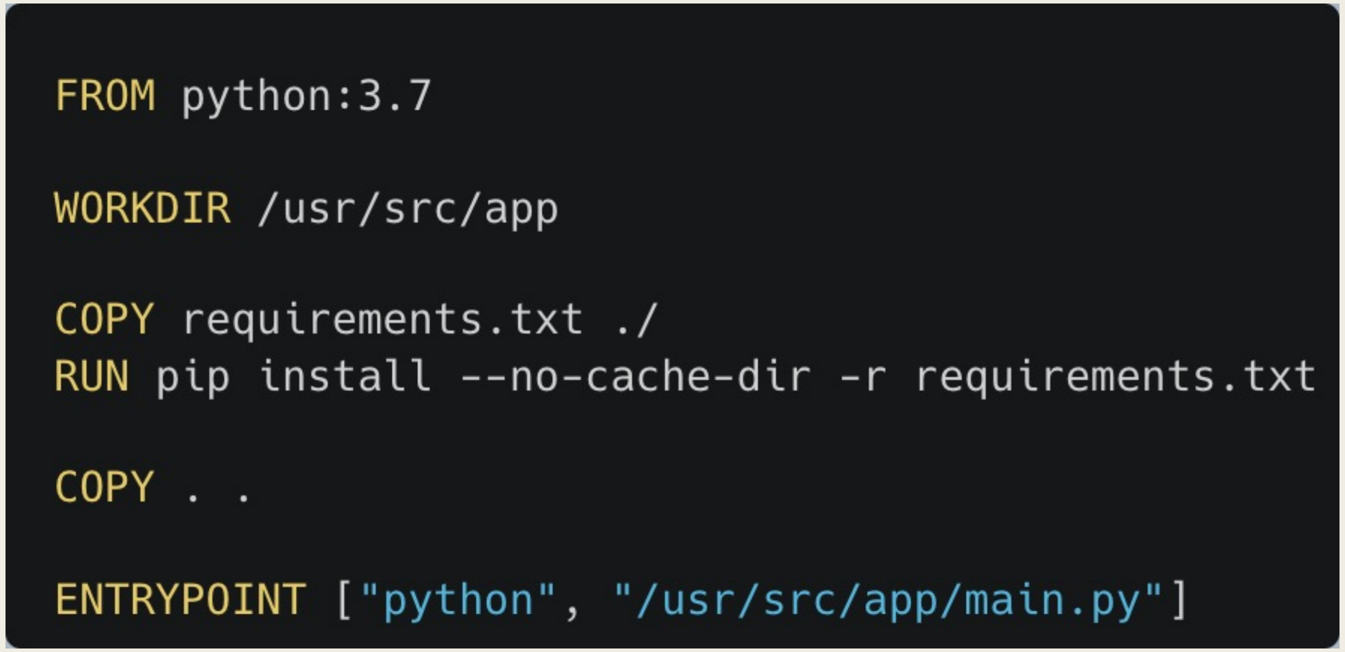
```yaml
name: 'BlogOps'
description: 'Synchronize your github pages ...'
author: 'Andrea Grillo'
...
inputs:
  posts_dir:
    description: 'Path of the posts folder'
    required: false
    default: './_posts/'
  gh_access_token:
    description: 'Github token to read ...'
    required: true
  ...
runs:
  using: 'docker'
  image: 'Dockerfile'
```

OPZIONE #3

Creare un *Container Action*

Dentro la repository:

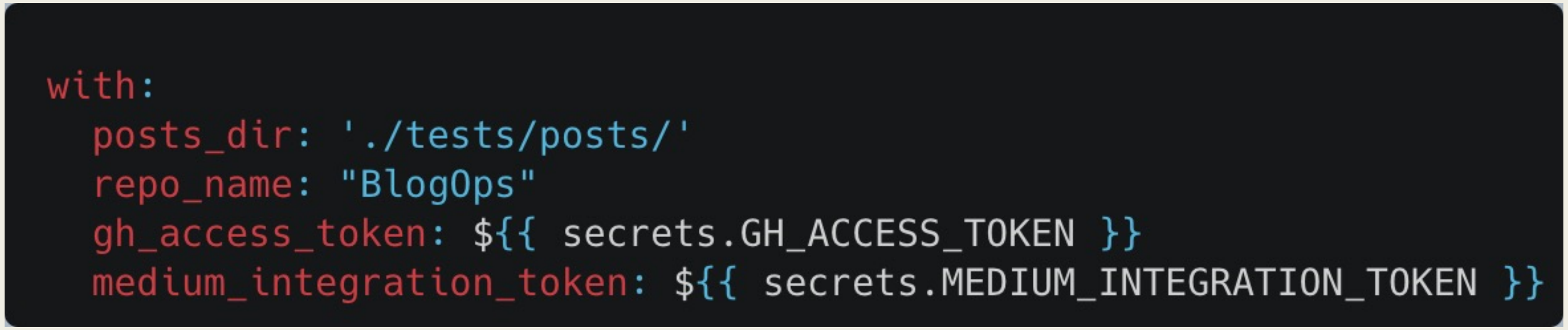
- *`action.yml`*
- *`Files python`*
- *`Requirements`*
- ***`Dockerfile`***



```dockerfile
FROM python:3.7

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

ENTRYPOINT ["python", "/usr/src/app/main.py"]
```

Ricevere gli INPUT dell’Action

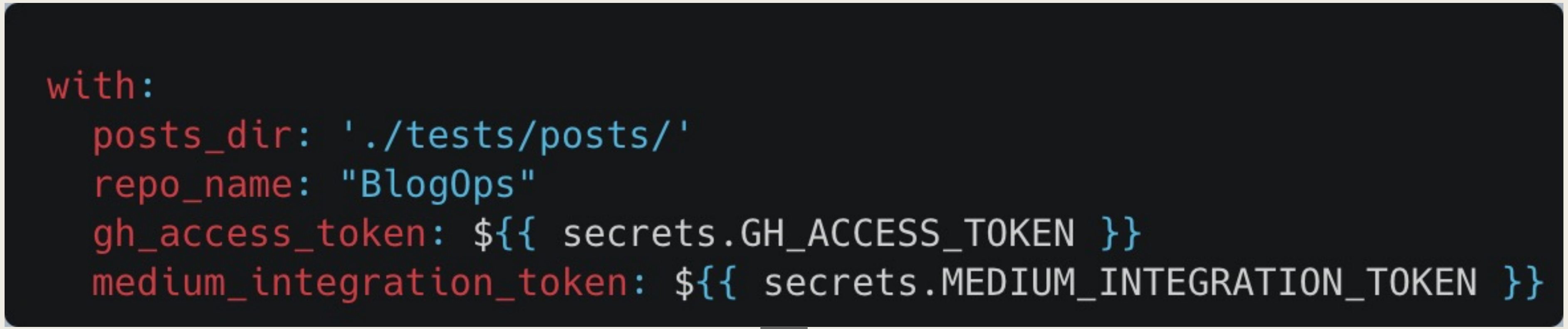
From the workflow…



```yaml
with:
  posts_dir: './tests/posts/'
  repo_name: "BlogOps"
  gh_access_token: ${{ secrets.GH_ACCESS_TOKEN }}
  medium_integration_token: ${{ secrets.MEDIUM_INTEGRATION_TOKEN }}
```

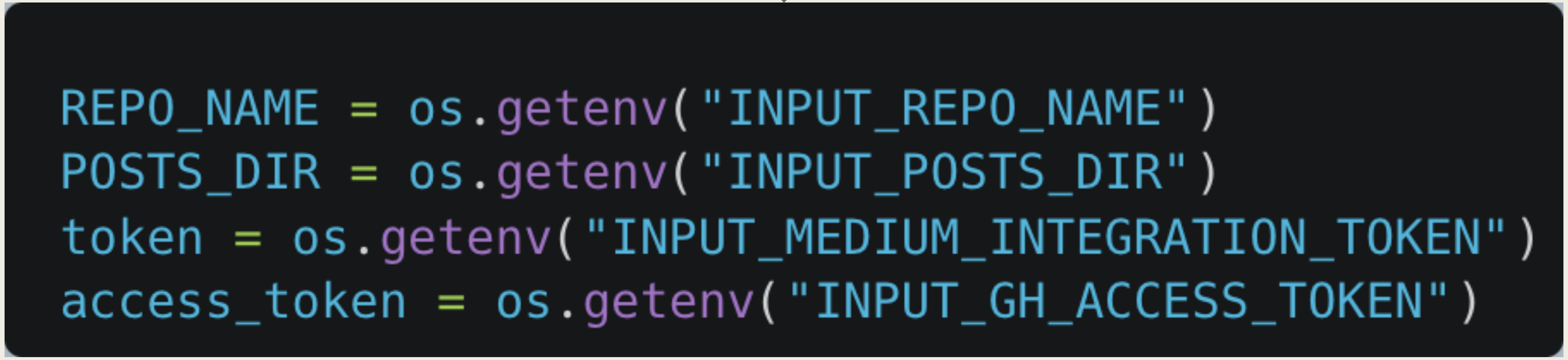
Ricevere gli INPUT dell’Action

From the workflow…



```yaml
with:
  posts_dir: './tests/posts/'
  repo_name: "BlogOps"
  gh_access_token: ${{ secrets.GH_ACCESS_TOKEN }}
  medium_integration_token: ${{ secrets.MEDIUM_INTEGRATION_TOKEN }}
```

…to python `INPUT_[VARIABLE NAME]`



```python
REPO_NAME = os.getenv("INPUT_REPO_NAME")
POSTS_DIR = os.getenv("INPUT_POSTS_DIR")
token = os.getenv("INPUT_MEDIUM_INTEGRATION_TOKEN")
access_token = os.getenv("INPUT_GH_ACCESS_TOKEN")
```

Vantaggi e Svataggi di un Action

Meno **duplicazione** di codice

Vantaggi e Svataggi di un Action

Meno **duplicazione** di codice

Il *workflow* diventa più **leggibile**

Vantaggi e Svataggi di un Action

Meno **duplicazione** di codice

Il *workflow* diventa più **leggibile**

Non si possono utilizzare i **segreti** della repository

TEST E RILASCIO

Test e Rilascio

TEST

- nektos/act
- Docker
- *steps* eseguiti nel container
- `act -j my_job --secret-file .secret`

Test e Rilascio

TEST

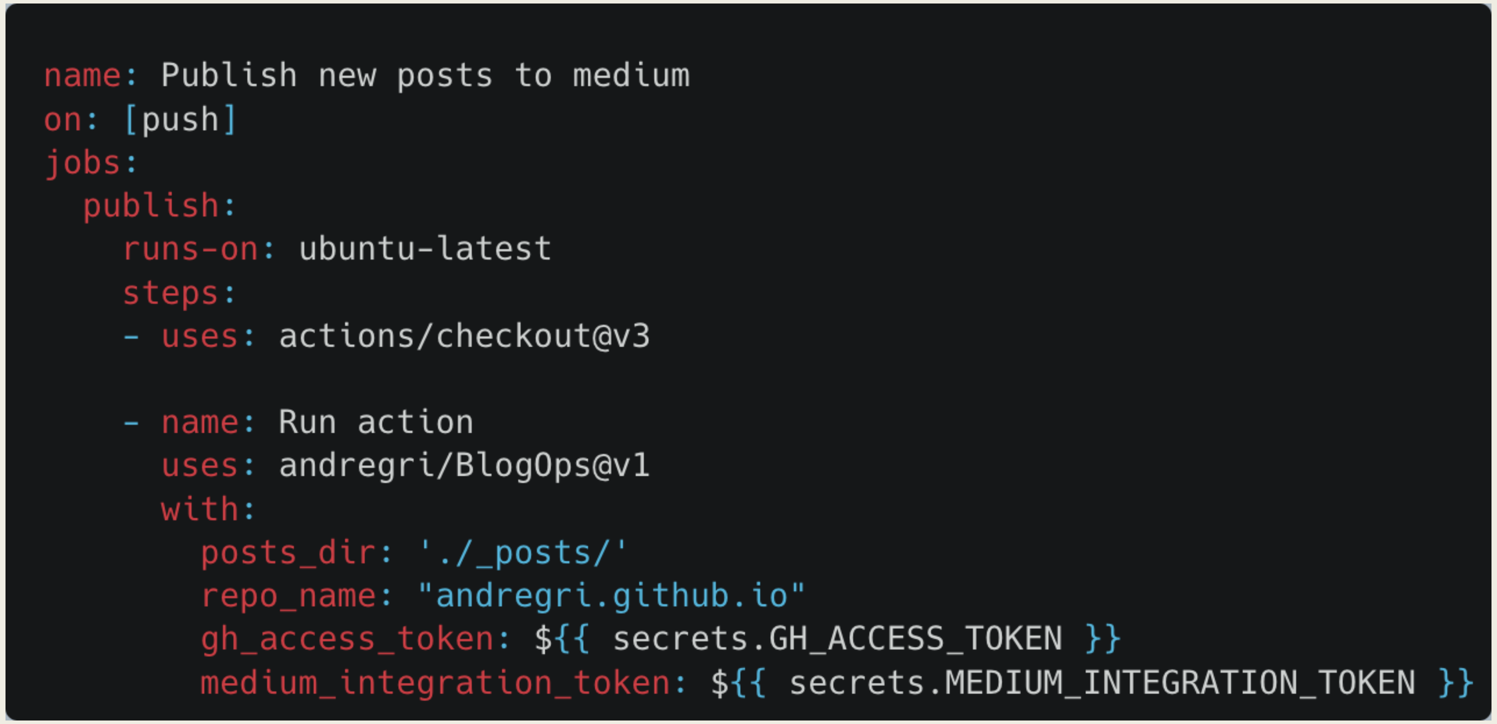
- nektos/act
- Docker
- *steps* eseguiti nel container
- `act -j my_job --secret-file .secret`

RILASCIO

- Marketplace
- autore/repository@id
- commit, branch, tag

Usare un action del *Marketplace*

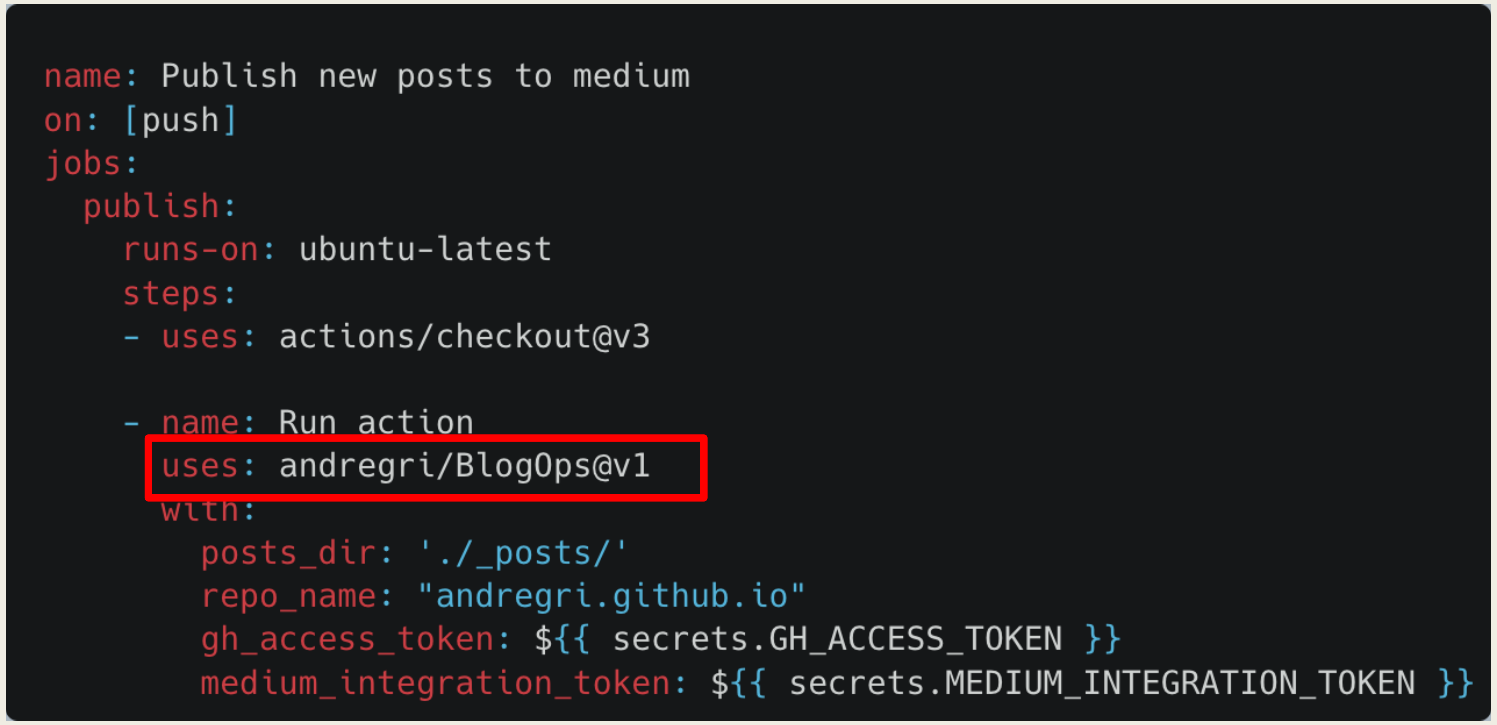
`workflow.yaml`



```yaml
name: Publish new posts to medium
on: [push]
jobs:
  publish:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Run action
      uses: andregri/BlogOps@v1
      with:
        posts_dir: './_posts/'
        repo_name: "andregri.github.io"
        gh_access_token: ${{ secrets.GH_ACCESS_TOKEN }}
        medium_integration_token: ${{ secrets.MEDIUM_INTEGRATION_TOKEN }}
```

Usare un action del *Marketplace*

workflow.yaml



```yaml
name: Publish new posts to medium
on: [push]
jobs:
  publish:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Run action
      uses: andregri/BlogOps@v1
      with:
        posts_dir: './_posts/'
        repo_name: "andregri.github.io"
        gh_access_token: ${{ secrets.GH_ACCESS_TOKEN }}
        medium_integration_token: ${{ secrets.MEDIUM_INTEGRATION_TOKEN }}
```

CONCLUSIONI

- Cos’è GitHub Actions
- Sviluppare un action con Python
- Test in locale con act
- Rilascio sul Marketplace
- andrea.grillo96@gmail.com