

Caso de Estudio:

Despliegue de sistemas software

Prof. Jose María Álvarez Rodríguez
Febrero 2024



01

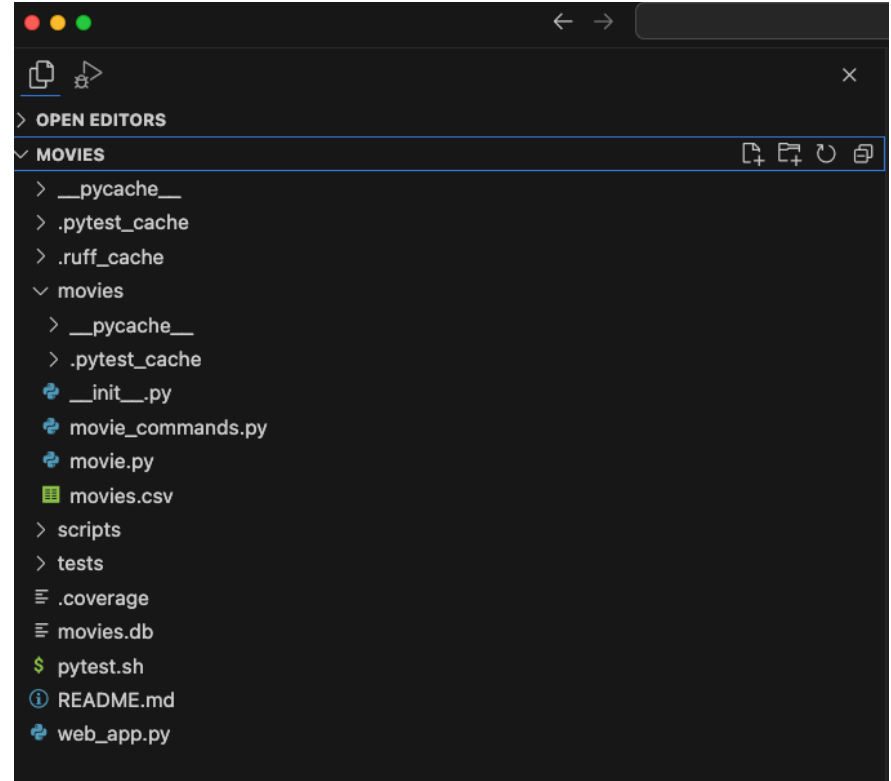
Contexto

Tipo de aplicación

- Se trata de una aplicación con operaciones CRUD (Create, Retrieve, Update y Delete) sobre información de películas de cine (título, duración y categoría).
- Cuenta con una capa de acceso datos, una capa de servicios de negocio y un API de operación de servicios con REST con tres métodos:
 - Crear una nueva película
 - Listar todas las películas
 - Dado un título de una película, devolver toda la información de esta

Diseño actual y distribución del código

- Una clase para representar la información de una película (`movie.py`)
- Una clase para realizar el acceso a datos y la conexión a la fuente de datos (`movie_commands.py`)
- Una clase para implementar la capa de servicios REST (`web_app.py`)
- Una clase para las pruebas unitarias y funcionales del sistema (`tests`)





02

Necesidades

Camino a seguir

- Diseñar e implementar un “pipeline” que nos permita asegurar que:
 - el código está con el formato adecuado
 - el código es correcto de acuerdo con unas guías y estándares
 - el código cumple con una serie de pruebas
 - el código cumple con una serie de métricas
 - →la primera etapa en detalle



Código con formato adecuado

- En Python se cuenta con bibliotecas que se encargan de esta función como:
 - Black: <https://github.com/psf/black>

Código siguiendo estándares

- En Python se cuenta con bibliotecas que se encargan de esta función como:
 - Pylint: <https://pypi.org/project/pylint/>
 - Ruff: <https://github.com/astral-sh/ruff>
 - Flake8: <https://flake8.pycqa.org/en/latest/>

Código cumple con las pruebas

- En Python se cuenta con bibliotecas que se encargan de esta función como:
 - Pytest: <https://docs.pytest.org/en/8.0.x/>
 - El programa provisto ya incluye pruebas con esta biblioteca.

Código cumple con una serie de métricas

- En Python se cuenta con bibliotecas que se encargan de esta función como:
 - Pytest: <https://docs.pytest.org/en/8.0.x/>
 - Utilizando la implementación propuesta se debe incluir una ejecución con el siguiente parámetro:
 - `python -m pytest '$TEST_DIR' --cov=movies`

Código cumple con una serie de métricas

- En Python se cuenta con bibliotecas que se encargan de esta función para el ámbito de seguridad como:
 - Bandit: <https://pypi.org/project/bandit/>
 - Radon: <https://pypi.org/project/radon/>
 - En concreto, para la métrica de “McCabe’s complexity” utilizando el comando siguiente:
 - `radon cc -a "$TARGET_DIR"`



03

Diseño y pasos de implementación

Pipeline objetivo

code-preparation

Formatear código

- black



Chequear estándares

- pylint
- flake8
- ruff



tests

Ejecutar pruebas

- pytest



Ejecutar cobertura de código

- pytest



Ejecutar otras pruebas

- bandit
- radon



deploy

Desplegar servicio (opcional)

- Ej: Heroku



Configure-dependencies



Ejemplo de salida

The screenshot shows a GitLab web interface for a pipeline run. The browser address bar indicates the URL is `gitlab.com/chemaar/devops_uc3m/-/pipelines/1254670456`. The page title is "Update .gitlab-ci.yml file". A green status bar indicates the pipeline is "Passed". The user "JOSE MARIA ALVAREZ RODRIGUEZ" created the pipeline for commit `1bf9cee7` 3 minutes ago. The pipeline is for the `main` branch and consists of 6 jobs. The jobs are grouped into three stages: `code-preparation` (code-formatting, code-quality-checker), `tests` (other-tests, tests-coverage, unit-functional-testing), and `deploy` (deploy-to-heroku). All jobs are marked as successful with green checkmarks. A "Delete" button is visible in the top right corner.

JOSE MARIA ALVAREZ RODRIGUEZ / devops_uc3m / Pipelines / #1254670456

Update .gitlab-ci.yml file

Passed JOSE MARIA ALVAREZ RODRIGUEZ created pipeline for commit `1bf9cee7` 3 minutes ago, finished just now

For `main`

latest 6 jobs 6.16 3 minutes 7 seconds, queued for 0 seconds

Pipeline Needs Jobs 6 Tests 0

code-preparation

- code-formatting
- code-quality-checker

tests

- other-tests
- tests-coverage
- unit-functional-testing

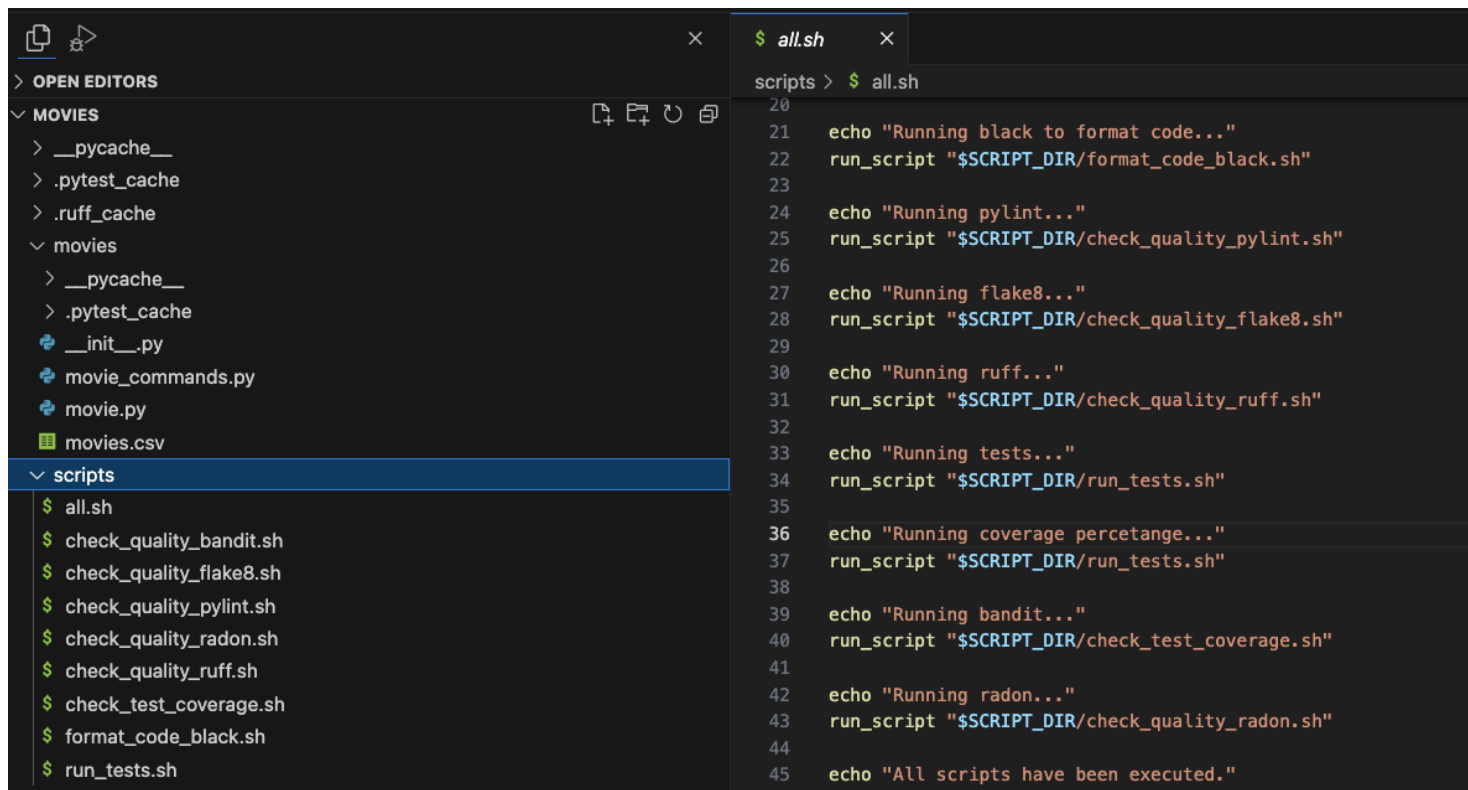
deploy

- deploy-to-heroku

Para cada etapa...

- Instalar y configurar las bibliotecas de Python
- Probar la ejecución en línea de comandos
- Generar un script para ejecutar este comando de tal manera que si falla la ejecución o la condición deseada se devuelva un código de error ($\neq 0$)
 - **Se recomienda bash**
- Ser capaz de ejecutar estos scripts desde el directorio padre de la carpeta “movies”
- Ejecutar todos los scripts
- Implementar el pipeline en Gitlab o Github

Ejemplo de configuración “off-line”



The screenshot shows a code editor with two panels. The left panel displays a file explorer with a project structure. The right panel shows the content of a shell script named `all.sh`.

File Explorer (Left Panel):

- OPEN EDITORS
- MOVIES
 - __pycache__
 - .pytest_cache
 - .ruff_cache
- movies
 - __pycache__
 - .pytest_cache
 - __init__.py
 - movie_commands.py
 - movie.py
 - movies.csv
- scripts
 - `$ all.sh`
 - `$ check_quality_bandit.sh`
 - `$ check_quality_flake8.sh`
 - `$ check_quality_pylint.sh`
 - `$ check_quality_radon.sh`
 - `$ check_quality_ruff.sh`
 - `$ check_test_coverage.sh`
 - `$ format_code_black.sh`
 - `$ run_tests.sh`

Shell Script (Right Panel):

```
$ all.sh
scripts > $ all.sh
20
21 echo "Running black to format code..."
22 run_script "$SCRIPT_DIR/format_code_black.sh"
23
24 echo "Running pylint..."
25 run_script "$SCRIPT_DIR/check_quality_pylint.sh"
26
27 echo "Running flake8..."
28 run_script "$SCRIPT_DIR/check_quality_flake8.sh"
29
30 echo "Running ruff..."
31 run_script "$SCRIPT_DIR/check_quality_ruff.sh"
32
33 echo "Running tests..."
34 run_script "$SCRIPT_DIR/run_tests.sh"
35
36 echo "Running coverage percentage..."
37 run_script "$SCRIPT_DIR/run_tests.sh"
38
39 echo "Running bandit..."
40 run_script "$SCRIPT_DIR/check_test_coverage.sh"
41
42 echo "Running radon..."
43 run_script "$SCRIPT_DIR/check_quality_radon.sh"
44
45 echo "All scripts have been executed."
```




04

Resultados

Salida esperada: black

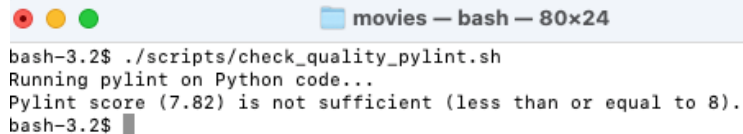


```
movies — bash — 80x24
bash-3.2$ ./scripts/format_code_black.sh
Formatting Python code with black...
All done! ✨🍰✨
3 files left unchanged.
No black issues found in the Python code.
bash-3.2$
```

Salida esperada: pylint

- Se aceptará un valor por encima de 7
- Se dispondrá de un mensaje por pantalla además de la devolución del código apropiado

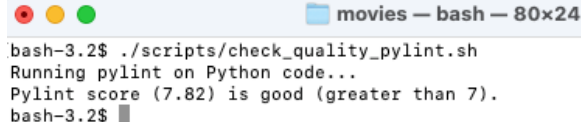
Incorrecto



A terminal window titled 'movies — bash — 80x24' showing the execution of a script. The output indicates that the pylint score is not sufficient.

```
bash-3.2$ ./scripts/check_quality_pylint.sh
Running pylint on Python code...
Pylint score (7.82) is not sufficient (less than or equal to 8).
bash-3.2$
```

Correcto



A terminal window titled 'movies — bash — 80x24' showing the execution of a script. The output indicates that the pylint score is good.

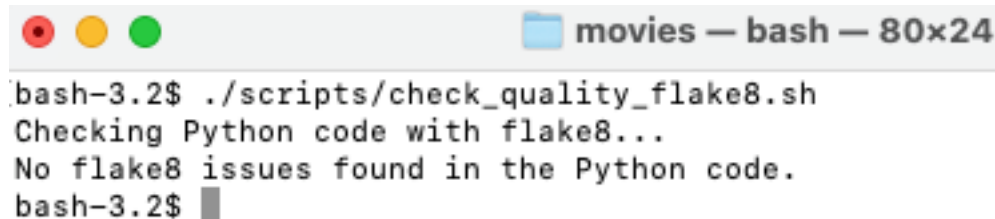
```
bash-3.2$ ./scripts/check_quality_pylint.sh
Running pylint on Python code...
Pylint score (7.82) is good (greater than 7).
bash-3.2$
```

Salida esperada: flake8

- Salida inicial

```
bash-3.2$ ./scripts/check_quality_flake8.sh
Checking Python code with flake8...
...movies/movie.py:67:80: E501 line too long (88 > 79 characters)
...movies/movie.py:79:80: E501 line too long (98 > 79 characters)
...movies/movie_commands.py:15:80: E501 line too long (80 > 79 characters)
Error: flake8 found issues in the Python code.
```

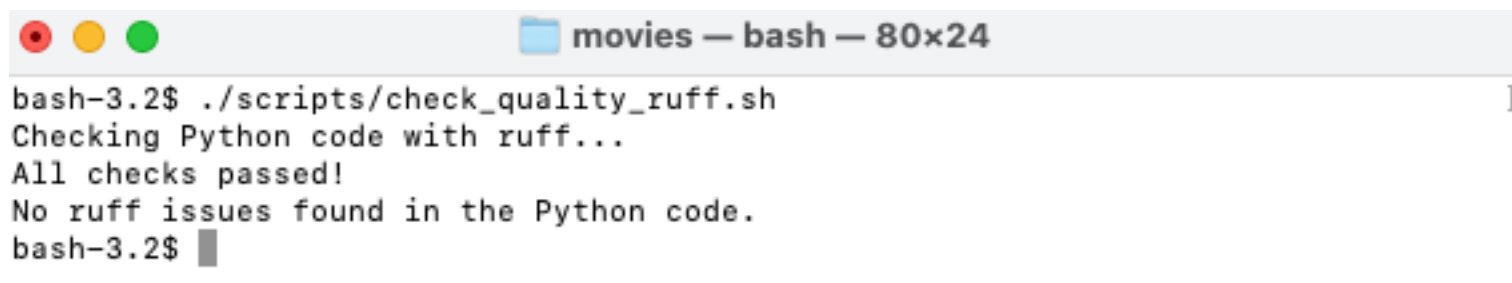
- Salida final: configurar flake8 poniendo un valor de línea máxima de 100



A terminal window titled "movies — bash — 80x24" showing the execution of the script. The output indicates that no flake8 issues were found after configuring the maximum line length to 100.

```
bash-3.2$ ./scripts/check_quality_flake8.sh
Checking Python code with flake8...
No flake8 issues found in the Python code.
bash-3.2$
```

Salida esperada: ruff



```
bash-3.2$ ./scripts/check_quality_ruff.sh
Checking Python code with ruff...
All checks passed!
No ruff issues found in the Python code.
bash-3.2$
```

The image shows a terminal window titled "movies — bash — 80x24". The terminal displays the execution of a script `./scripts/check_quality_ruff.sh`. The output of the script is: "Checking Python code with ruff...", "All checks passed!", and "No ruff issues found in the Python code.". The prompt `bash-3.2$` is shown at the end of the output.

Salida esperada: pytest

```
movies — bash — 80x24
_tdd_sources/movies
plugins: cov-4.1.0
collected 6 items

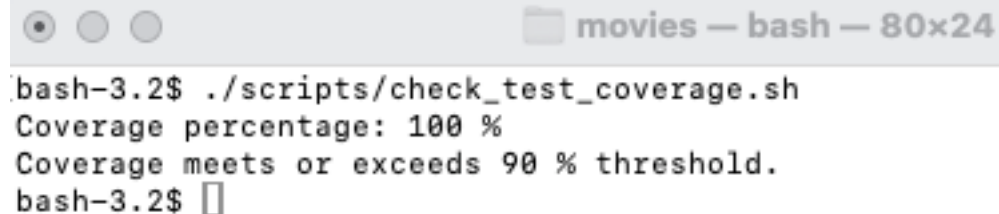
tests/test_app.py ... [ 50%]
tests/test_movies.py ... [100%]

===== warnings summary =====
tests/test_app.py::test_create_movie
tests/test_app.py::test_get_movie
tests/test_app.py::test_list_movies
tests/test_app.py::test_list_movies
tests/test_app.py::test_list_movies
/Users/josemaria.alvarez/anaconda3/envs/LOTAR/lib/python3.11/site-packages/pydantic/main.py:1024: PydanticDeprecatedSince20: The `dict` method is deprecated; use `model_dump` instead. Deprecated in Pydantic V2.0 to be removed in V3.0. See Pydantic V2 Migration Guide at https://errors.pydantic.dev/2.6/migration/
  warnings.warn('The `dict` method is deprecated; use `model_dump` instead.',
category=PydanticDeprecatedSince20)

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 6 passed, 5 warnings in 0.25s =====
All pytest tests have been passed successfully.
bash-3.2$
```

Salida esperada: pytest con cobertura

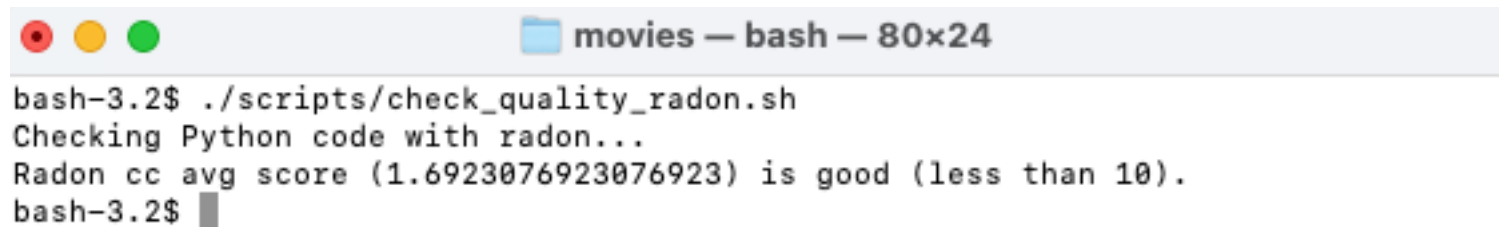
- Se aceptará un valor de cobertura de código superior al 90%

A terminal window titled 'movies — bash — 80x24' showing the execution of a script to check test coverage. The script outputs a coverage percentage of 100% and confirms it meets the 90% threshold.

```
bash-3.2$ ./scripts/check_test_coverage.sh
Coverage percentage: 100 %
Coverage meets or exceeds 90 % threshold.
bash-3.2$
```

Salida esperada: radon

- Se aceptará un valor complejidad de McCabe menor a 10

A screenshot of a terminal window with a light gray title bar. The title bar contains three colored window control buttons (red, yellow, green) on the left and a folder icon followed by the text "movies — bash — 80x24" on the right. The terminal content shows a shell prompt "bash-3.2\$" followed by the command "./scripts/check_quality_radon.sh". The output consists of three lines: "Checking Python code with radon...", "Radon cc avg score (1.6923076923076923) is good (less than 10).", and a final shell prompt "bash-3.2\$" with a dark gray cursor block.

```
bash-3.2$ ./scripts/check_quality_radon.sh
Checking Python code with radon...
Radon cc avg score (1.6923076923076923) is good (less than 10).
bash-3.2$ █
```


Salida esperada: bandit

movies — bash — 80x24

[main] INFO running on Python 3.11.5

Run started:2024-04-15 21:40:00.512136

Test results:

No issues identified.

Code scanned:

Total lines of code: 91

Total lines skipped (#nosec): 0

Run metrics:

Total issues (by severity):

Undefined: 0

Low: 0

Medium: 0

High: 0

Total issues (by confidence):

Undefined: 0

Low: 0

Medium: 0

High: 0

Files skipped (0):

No bandit issues found in the Python code.

bash-3.2\$



05

Tarea y planificación

Recursos

- Código fuente con la implementación de referencia
 - Bibliotecas Python a instalar (simplemente con pip):
 - requirements.txt
 - pip install jsonschema Flask
 - pip install requests
 - (no se incluye por referencia a path local)
- ```
bandit==1.7.8
black==24.4.0
flake8==7.0.0
flask-restplus==0.13.0
isort==5.13.2
pydantic==2.6.4
pydantic_core==2.16.3
pylint==3.1.0
pytest==8.1.1
pytest-cov==4.1.0
radon==6.0.1
ruff==0.3.3
```

# Notas importantes

- Se requiere cierta configuración del pipeline: versión de Python image y permisos en los scripts
- Se puede requerir instalar utilidades en bash
- Se puede requerir cierta configuración en los nombres de los módulos de Python para ejecutar pytest adecuadamente

# Entregables

- Un archivo comprimido conteniendo
  - El conjunto de scripts para cada tarea
  - Un script para lanzarlos todos seguidos o que sea compatible con un provisto (nombrado)
  - Una pipeline en Gitlab o Github que ejecute estos scripts: fichero .gitlab-ci.yml y una captura de pantalla de la ejecución
  - [Opcional] Incluir en la pipeline el despliegue de la aplicación en Heroku u otro proveedor en la nube (especial atención si se utiliza alguno que requieran datos de facturación, asegurar que se apaga la máquina en la nube)
    - [https://docs.gitlab.com/ee/ci/cloud\\_deployment/heroku.html](https://docs.gitlab.com/ee/ci/cloud_deployment/heroku.html)
    - Requiere registro

# Evaluación y plazo

- Tipo: Individual
- Calificación máxima: 10
- Peso: 1 punto
- Entrega en Aulaglobal el día 30 de abril de 2024
- Nota:
  - Si se utiliza una herramienta de IA generativa para configurar los scripts, se debe incluir una tabla (MSExcel o MSWord) con un enlace al prompt utilizado o bien el texto utilizado