grupy.rn

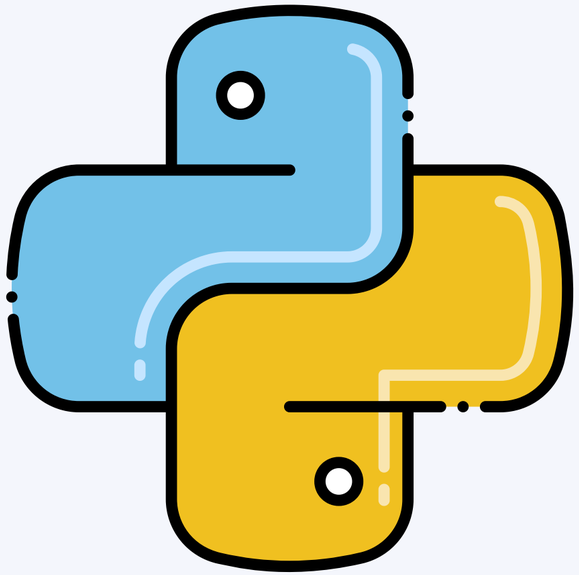
❯ 20º Meetup GruPy RN 2025

Testando múltiplas versões do Python em monorepos

Lições aprendidas com Tox

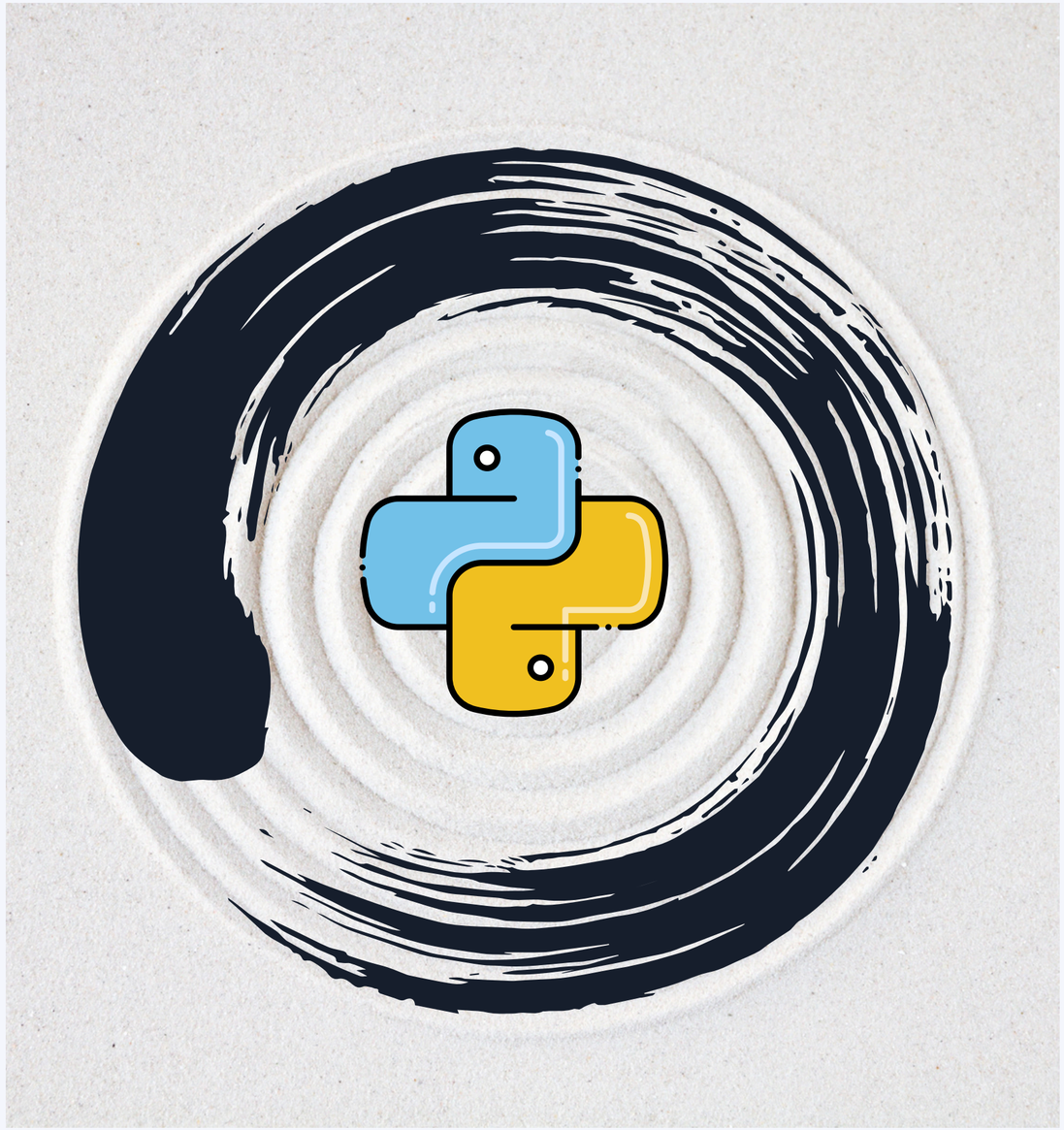
Emídio Neto
PicPay

Agenda



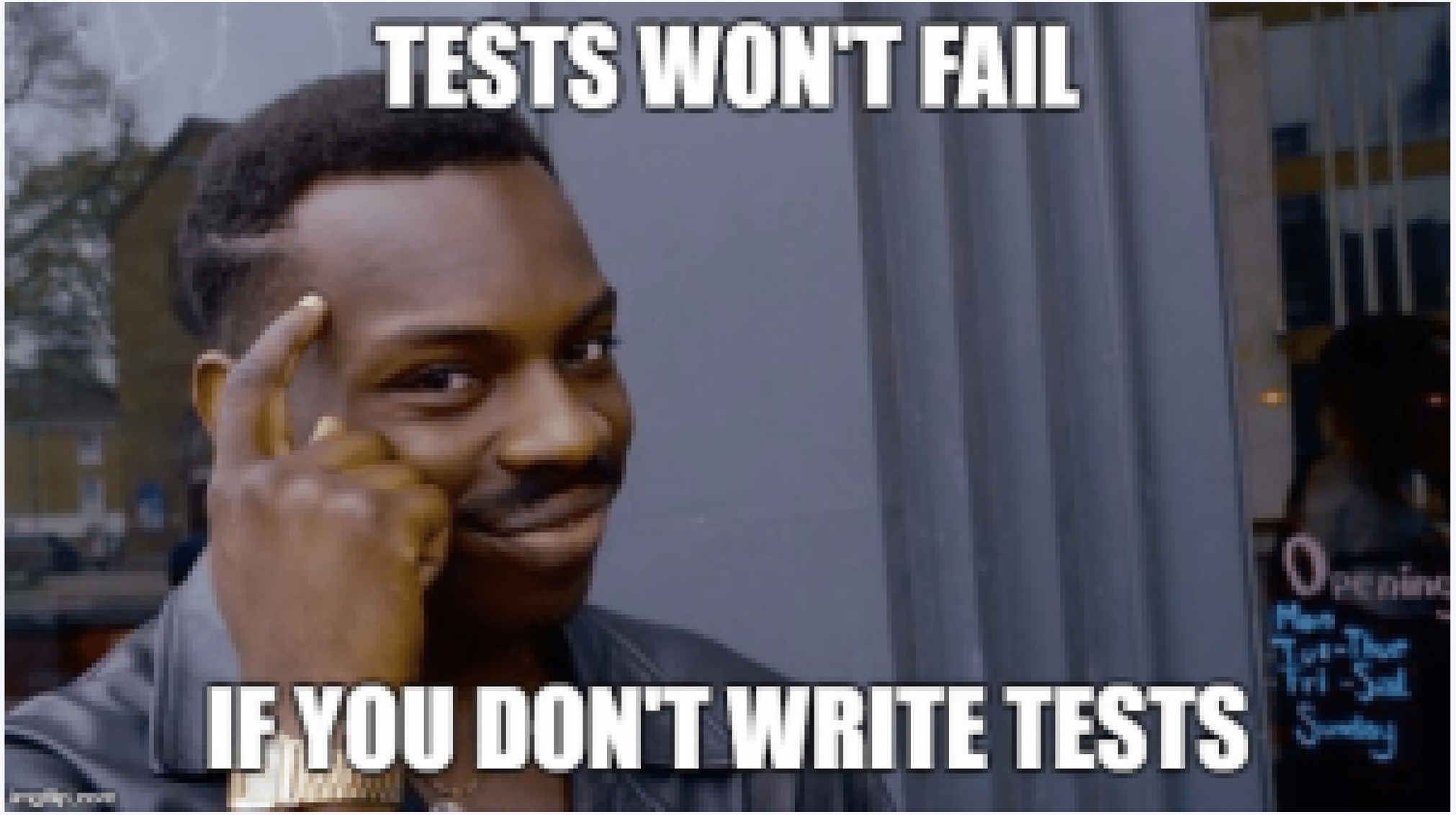
1. Por que testar múltiplas versões ?
2. Como o Tox ajuda (ou não)
3. Desafios e lições aprendidas
4. Q&A

Por que testar múltiplas versões ?



> Errors should never pass silently
>
> Unless explicitly silenced

PEP 20 – The Zen of Python


TESTS WON'T FAIL
IF YOU DON'T WRITE TESTS

Por que testar múltiplas versões ?

1. Garantir compatibilidade
2. Preparação para o futuro e detecção de depreciações
3. Requisitos específicos a depender do ambiente/cliente

“

PEP 585 – Built-in Generic Types (Python 3.9)

A partir do Python 3.9, você pode usar os tipos integrados diretamente (por exemplo, list[int]), o que pode quebrar o código que depende do estilo antigo.

snippet.py

```python
def double_numbers(nums: list[int]) -> list[int]:
    return [n * 2 for n in nums]
```

snippet

```
$ python3.8 snippet.py

Traceback (most recent call last):
  File "snippet.py", line 1, in <module>
    def double_numbers(nums: list[int]) -> list[int]:
TypeError: 'type' object is not subscriptable
```

“

PEP 604 – Allow writing union types as X | Y

No Python 3.9, embora você pudesse usar tipos integrados (como list[int]) graças ao PEP 585, os tipos union ainda precisam da sintaxe “typing.Union” se você quiser compatibilidade com versões anteriores (<3.10).

snippet

```python
def get_items(data: list[int] | None) -> list[str] | None:
    if data is None:
        return None
    return [str(x) for x in data]
```

snippet

```
$ python3.9 snippet.py
Traceback (most recent call last):
  File "snippet.py", line 1, in <module>
    def get_items(data: list[int] | None) -> list[str] | None:
TypeError: unsupported operand type(s) for |: 'types.GenericAlias' and 'NoneType'
```

Breaking changes que afetam a execução em runtime não são legais

Vocês estão testando o código que escrevem???

¯_(ツ)_/¯

Uma boa prática é rodar testes para todas versões Python que sua biblioteca suporta

Matriz de Testes

Static Type Checkers

: mypy

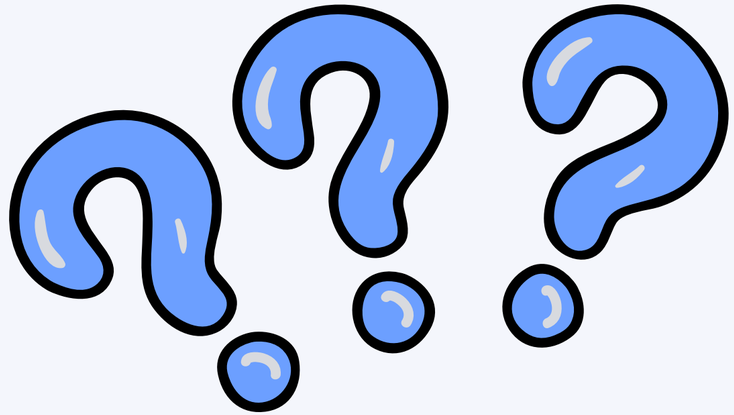
PYRIGHT

Pacotes populares fazem isso

pytest

psycopg2

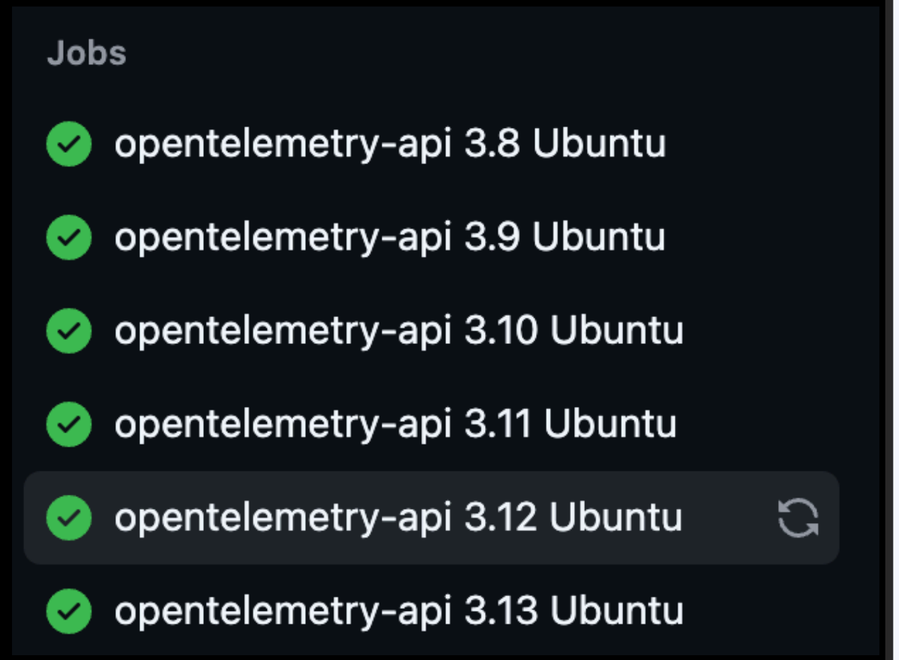
Quem já precisou testar código para múltiplas versões do Python?



Testes

Considere a seguinte matriz de testes:

package: [”opentelemetry-api”]
OS: [”Ubuntu”]
python: [”3.8", “3.9”, “3.10”, “3.11”, “3.12”, “3.13”]



urllib3

Jobs

- package
- macOS 3.9
- Windows 3.9
- Ubuntu 3.9
- macOS 3.10
- Windows 3.10
- Ubuntu 3.10
- macOS 3.11
- Windows 3.11
- Ubuntu 3.11
- macOS 3.12
- Windows 3.12
- Ubuntu 3.12
- macOS 3.13
- Windows 3.13
- Ubuntu 3.13
- macOS 3.14
- Windows 3.14
- Ubuntu 3.14

FastAPI

- test (3.13, pydantic-v1)
- test (3.13, pydantic-v2)
- test (3.12, pydantic-v1)
- test (3.12, pydantic-v2)
- test (3.11, pydantic-v1)
- test (3.11, pydantic-v2)
- test (3.10, pydantic-v1)
- test (3.10, pydantic-v2)
- test (3.9, pydantic-v1)
- test (3.9, pydantic-v2)
- test (3.8, pydantic-v1)
- test (3.8, pydantic-v2)


Testes
46 test:
47 runs-on: ubuntu-latest
48 strategy:
49 matrix:
50 python-version:
51 - "3.13"
52 - "3.12"
53 - "3.11"
54 - "3.10"
55 - "3.9"
56 - "3.8"
57 pydantic-version: ["pydantic-v1", "pydantic-v2"]

Para uma biblioteca parece OK

mas e pra 80?

opentelemetry-python Public

18 Python Packages

+

opentelemetry-python-contrib Public

63 Python Packages

opentelemetry-python Public

~283 Jobs na CI

+

opentelemetry-python-contrib Public

~715 Jobs na CI

All checks have passed

1 skipped, 384 successful checks

Test 0 / opentelemetry-api 3.12 Ubu

Quase 1000 Jobs

712 checks passed

instrumentation-openai-v2-latest 3.11 Ubuntu

instrumentation-urllib3-0 pypy-3.8 Ubuntu

Só criar uma matriz dos 80 pacotes

...

Até funcionou por um tempo, mas...

- **Job matrix** - A job matrix can generate a maximum of 256 jobs per workflow run. This limit applies to both GitHub-hosted and self-hosted runners.

Como o Tox ajuda (ou não)

“

Tox automatiza e padroniza testes em Python.
Como um orquestrador

https://tox.wiki

Verifica se o pacote é buildado e instalado corretamente em diferentes implementações/versões de Python, dependências etc.

Executa os testes em cada ambiente com a ferramenta de teste da sua preferência

Atua como um frontend para CI

Altamente customizável com Plugins
Ex: tox-uv

Tox não substitui ferramentas como Pytest

Você pode usar essas ferramentas em conjunto com Tox para criar ambientes reproduzíveis

```
$ pip install tox
```

tox.ini

```
[tox]
env_list =
    py3{8,9,10,11,12,13}
    type

[testenv]
deps =
    pytest
commands =
    pytest tests

[testenv:type]
deps =
    pyright
commands =
    pyright
```

Definição dos ambientes

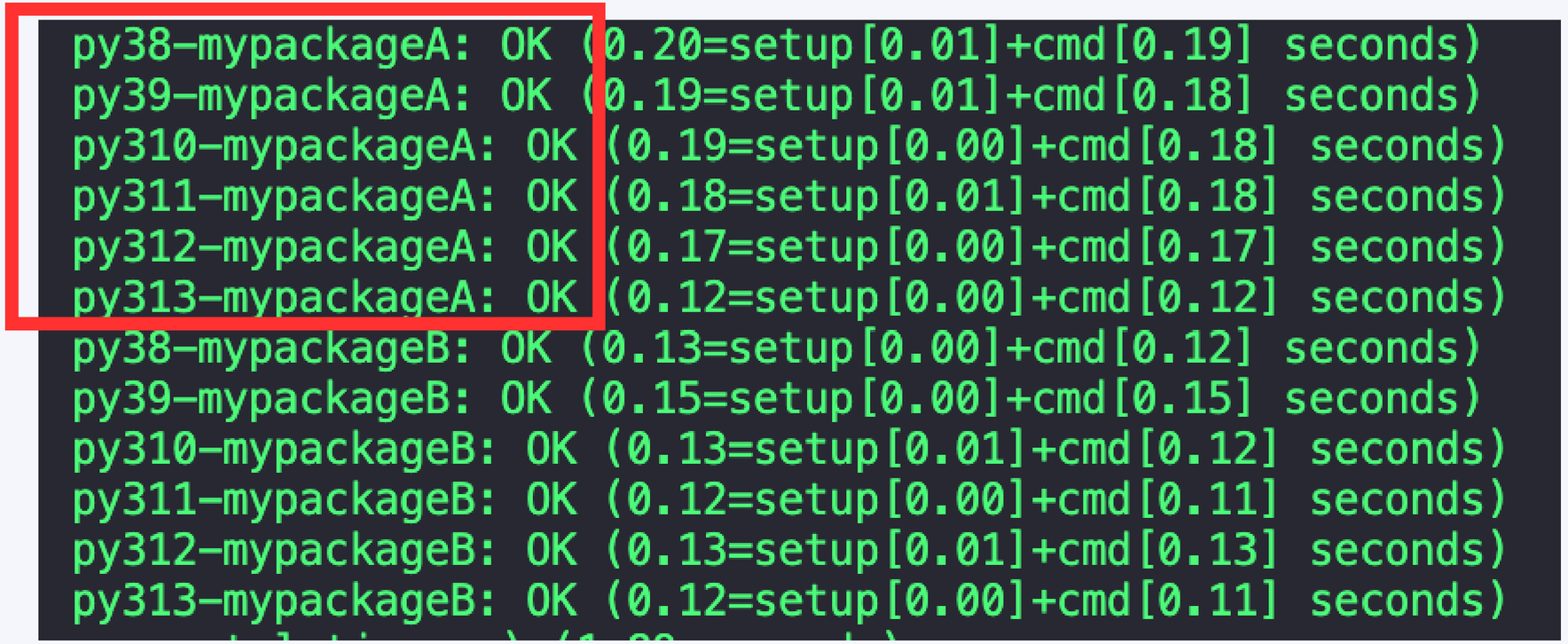
$ tox

```
  py38: OK (0.20=setup[0.00]+cmd[0.19] seconds)
  py39: OK (0.19=setup[0.00]+cmd[0.19] seconds)
  py310: OK (0.19=setup[0.00]+cmd[0.18] seconds)
  py311: OK (0.17=setup[0.00]+cmd[0.17] seconds)
  py312: OK (0.20=setup[0.00]+cmd[0.20] seconds)
  py313: OK (0.12=setup[0.00]+cmd[0.12] seconds)
  type: OK (2.80=setup[1.31]+cmd[1.49] seconds)
  congratulations :) (3.92 seconds)
```

tox.ini

```
[tox]
env_list =
    py3{8,9,10,11,12,13}-mypackageA
    py3{8,9,10,11,12,13}-mypackageB

[testenv]
deps =
    pytest
    mypackageA: -r requirementsA.txt
    mypackageB: -r requirementsB.txt
commands =
    mypackageA: pytest {toxinidir}/mypackageA
    mypackageB: pytest {toxinidir}/mypackageB
```



- 2 Pacotes
- 2 diretórios diferentes de Testes
- Dependências diferentes para cada ambiente

tox.ini

```ini
[tox]
env_list =
    py3{8,9,10,11,12,13}-mypackageA-{lowest,highest}

[testenv]
deps =
    pytest
    mypackageA-lowest: Flask==2.0.0
    mypackageA-highest: Flask==2.3.2
commands_pre =
    uv pip freeze
commands =
    mypackageA: pytest {toxinidir}/mypackageA
```

```
py313-mypackageA-lowest: commands_pre[0]> uv pip freeze
Using Python 3.13.1 environment at: .tox/py313-mypackageA-lowest
click==8.1.8
flask==2.0.0
iniconfig==2.0.0
itsdangerous==2.2.0
jinja2==3.1.6
markupsafe==3.0.2
packaging==24.2
pluggy==1.5.0
pytest==8.3.5
werkzeug==3.1.3
```

```
py313-mypackageA-highest: commands_pre[0]> uv pip freeze
Using Python 3.13.1 environment at: .tox/py313-mypackageA-highest
blinker==1.9.0
click==8.1.8
flask==2.3.2
iniconfig==2.0.0
itsdangerous==2.2.0
jinja2==3.1.6
markupsafe==3.0.2
packaging==24.2
pluggy==1.5.0
pytest==8.3.5
werkzeug==3.1.3
```

```
py38-mypackageA-lowest: OK (0.65=setup[0.01]+cmd[0.02,0.63] seconds)
py38-mypackageA-highest: OK (0.51=setup[0.01]+cmd[0.01,0.49] seconds)
py39-mypackageA-lowest: OK (0.52=setup[0.01]+cmd[0.02,0.49] seconds)
py39-mypackageA-highest: OK (0.48=setup[0.00]+cmd[0.01,0.47] seconds)
py310-mypackageA-lowest: OK (0.51=setup[0.00]+cmd[0.01,0.49] seconds)
py310-mypackageA-highest: OK (0.45=setup[0.01]+cmd[0.02,0.43] seconds)
py311-mypackageA-lowest: OK (0.52=setup[0.01]+cmd[0.01,0.50] seconds)
py311-mypackageA-highest: OK (0.44=setup[0.00]+cmd[0.01,0.42] seconds)
py312-mypackageA-lowest: OK (0.51=setup[0.01]+cmd[0.01,0.49] seconds)
py312-mypackageA-highest: OK (0.46=setup[0.00]+cmd[0.01,0.44] seconds)
py313-mypackageA-lowest: OK (0.47=setup[0.01]+cmd[0.02,0.45] seconds)
py313-mypackageA-highest: OK (0.56=setup[0.01]+cmd[0.01,0.54] seconds)
congratulations :) (6.13 seconds)
```

tox.ini

```
[tox]
env_list =
    py3{8,9,10,11,12,13}-mypackageA
    ruff

[testenv]
deps =
    pytest
    ruff: ruff
commands =
    mypackageA: pytest {toxinidir}/mypackageA
    ruff: ruff check --fix
    ruff: ruff format
```

$ tox -e ruff

```
ruff: commands[0]> ruff check --fix
Found 1 error (1 fixed, 0 remaining).
ruff: commands[1]> ruff format
1 file reformatted, 8 files left unchanged
  ruff: OK (0.11=setup[0.01]+cmd[0.08,0.02] seconds)
  congratulations :) (0.17 seconds)
```

x

6 versões Python
+ pypy3 + lint

```
[tox]
isolated_build = True
skipsdist = True
skip_missing_interpreters = True
envlist =
    ; Environments are organized by individual package, allowing
    ; for specifying supported Python versions per package.

    py3{8,9,10,11,12,13}-test-opentelemetry-api
    pypy3-test-opentelemetry-api
    lint-opentelemetry-api

    py3{8,9,10,11,12,13}-test-opentelemetry-proto-protobuf5
    pypy3-test-opentelemetry-proto-protobuf5
    lint-opentelemetry-proto-protobuf5

    py3{8,9,10,11,12,13}-test-opentelemetry-sdk
    pypy3-test-opentelemetry-sdk
    lint-opentelemetry-sdk
    benchmark-opentelemetry-sdk

    py3{8,9,10,11,12,13}-test-opentelemetry-semantic-conventions
    pypy3-test-opentelemetry-semantic-conventions
    lint-opentelemetry-semantic-conventions
```

```
[testenv]
deps =
  lint: -r dev-requirements.txt
  coverage: pytest
  coverage: pytest-cov

  mypy,mypyinstalled: -r {toxinidir}/mypy-requirements.txt

  api: -r {toxinidir}/opentelemetry-api/test-requirements.txt

  sdk: -r {toxinidir}/opentelemetry-sdk/test-requirements.txt
  benchmark-opentelemetry-sdk: -r {toxinidir}/opentelemetry-sdk/benchmark-requirements.txt

  semantic-conventions: -r {toxinidir}/opentelemetry-semantic-conventions/test-requirements.txt
```

Para cada pacote, instalar as dependências necessárias pro teste

```
commands =
  test-opentelemetry-api: pytest {toxinidir}/opentelemetry-api/tests {posargs}
  lint-opentelemetry-api: pylint {toxinidir}/opentelemetry-api

  test-opentelemetry-sdk: pytest {toxinidir}/opentelemetry-sdk/tests {posargs}
  lint-opentelemetry-sdk: pylint {toxinidir}/opentelemetry-sdk
  benchmark-opentelemetry-sdk: pytest {toxinidir}/opentelemetry-sdk/benchmarks --ben

  test-opentelemetry-proto-protobuf5: pytest {toxinidir}/opentelemetry-proto/tests {
  lint-opentelemetry-proto-protobuf5: pylint {toxinidir}/opentelemetry-proto

  test-opentelemetry-semantic-conventions: pytest {toxinidir}/opentelemetry-semantic-
  lint-opentelemetry-semantic-conventions: pylint {toxinidir}/opentelemetry-semantic-
```

Para cada pacote, rodar os testes, lint ou benchmark

e se eu quiser rodar todos testes somente de um pacote?

```
$ tox -f test- opentelemetry-sdk
```

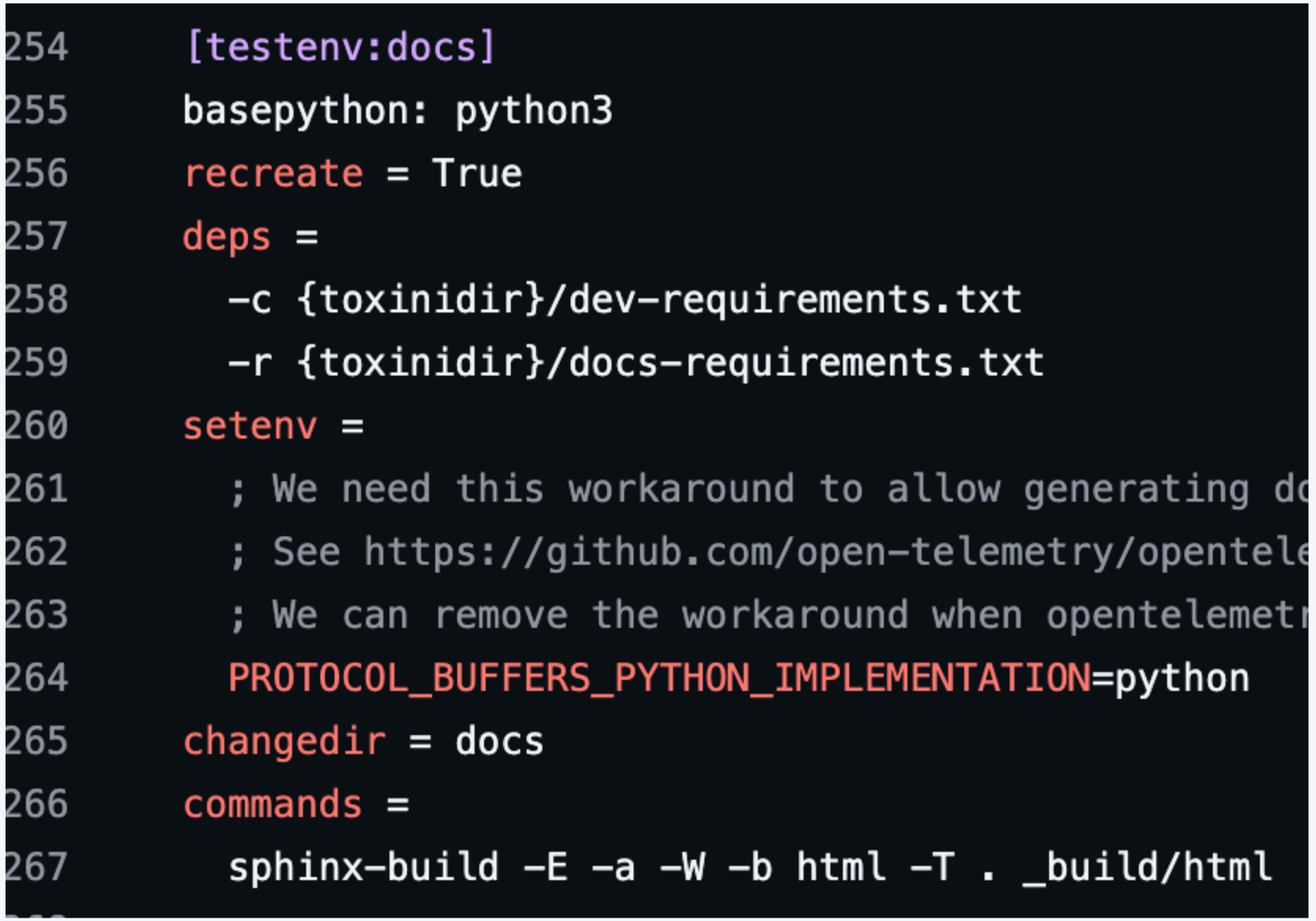
Vai executar a matriz de testes para todas versões definidas no ambiente

```
py38-test-opentelemetry-sdk: OK (19.48=setup[3.69]+cmd[15.80] seconds)
py39-test-opentelemetry-sdk: OK (17.54=setup[1.19]+cmd[16.34] seconds)
py310-test-opentelemetry-sdk: OK (17.13=setup[0.77]+cmd[16.36] seconds)
py311-test-opentelemetry-sdk: OK (16.15=setup[0.68]+cmd[15.48] seconds)
py312-test-opentelemetry-sdk: OK (15.79=setup[0.63]+cmd[15.16] seconds)
py313-test-opentelemetry-sdk: OK (15.38=setup[0.51]+cmd[14.88] seconds)
pypy3-test-opentelemetry-sdk: OK (27.29=setup[6.15]+cmd[21.14] seconds)
```

```
$ tox -e py38-test-opentelemetry-api
```

```
316     changedir =
317       tests/opentelemetry-docker-tests/tests
318
319     commands_pre =
320       pip freeze
321       docker-compose up -d
322     commands =
323       otlpexporter: pytest otlpexporter {posargs}
324       opencensus: pytest opencensus {posargs}
325
326     commands_post =
327       docker-compose down -v
```

Você
pode integrar
com diversas ferramentas
disponíveis no seu sistema operacional



```
254     [testenv:docs]
255     basepython: python3
256     recreate = True
257     deps =
258       -c {toxinidir}/dev-requirements.txt
259       -r {toxinidir}/docs-requirements.txt
260     setenv =
261       ; We need this workaround to allow generating do
262       ; See https://github.com/open-telemetry/opentele
263       ; We can remove the workaround when opentelemetr
264       PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION=python
265     changedir = docs
266     commands =
267       sphinx-build -E -a -W -b html -T . _build/html
```

Incluindo gerar as docs da biblioteca :)

```
344     [testenv:shellcheck]
345     commands_pre =
346       sh -c "sudo apt update -y && sudo apt install --assume-yes shellcheck"
347     commands =
348       sh -c "find {toxinidir} -name \*.sh | xargs shellcheck --severity=warning"
349
```

E até rodar shellscript hehe

Desafios e Lições Aprendidas

1. Arquivos tox.ini imensos -- Chegando a ~1040 linhas
2. Difícil manter sincronizado o que está no tox.ini VS o que roda na CI
3. Instalação de pacotes lenta em alguns testes
4. CI de Lint demorava séculos para rodar
5. Testar um mesmo pacote em múltiplas versões do Python só que também para múltiplas versões de uma dependência específica

1. Matriz no Github Action tem limite de 250 Jobs -- sem chances

2. Testar o “básico” do repo -contrib em todo PR no repositório core

3. Testes no -contrib apontam para branch main no -core

```
[testenv]
test_deps =
  opentelemetry-api@{env:CORE_REPO}\#egg=opentelemetry-api&subdirectory=opentelemetry-api
  opentelemetry-semantic-conventions@{env:CORE_REPO}\#egg=opentelemetry-semantic-convention
  opentelemetry-sdk@{env:CORE_REPO}\#egg=opentelemetry-sdk&subdirectory=opentelemetry-sdk
  opentelemetry-test-utils@{env:CORE_REPO}\#egg=opentelemetry-test-utils&subdirectory=tests
```

Tox tem vários plugins
Um deles é o tox-uv

Ajuda bastante se você já usa **uv** em seu projeto

Instalação de pacotes ultra mega rápido

Você pode integrar tox com outras ferramentas, como docker

Facilita ter testes mais robustos

Comece com poucos ambientes e “divida” conforme necessário

Separar em vários ambientes pode reduzir o tempo de execução dos testes na CI

Para repositórios menores: simplesmente funciona

Você vai automatizar várias tarefas “tediosas” e vai ter a garantia de que o que tá rodando localmente é o que tá rodando na CI

Obrigado!

@emdneto

emidioneto


CLOUD NATIVE
COMMUNITY GROUPS
RN
SAVE THE
DATE
12 de abril
Sebrae RN
Av. Lima e Silva, 76
CLOUD NATIVE
COMPUTING FOUNDATION
SEBRAE