Can We Deploy Yet?

by Anastasiia Tymoshchuk

anastasiatymo

Few words about myself

- Lead Dev at Scoutbee in Berlin
- PyBerlin organiser
- 10 years in software development
- 7 years in Python
- Happy Pythonista 🐍😊

anastasiatymo

What "Production" means to you?

Is my code ready for Production?

"You've worked hard on your project. It looks like all the features are actually complete, and most even have tests.

You can breathe a sigh of relief. You're done.

Or are you?"

from "Release It! Design and Deploy Production-ready Software" book by Michael T. Nygard

anastasiatymo

There are couple of checks for you before going to production

Ready for Production?

check your code now 🐍 😎

- Exception handling
- How to become a detective or meaningful logging
- From code review to production or effective CI/CD
- Docker? No problem!
- More hints and ideas

Exceptions

We don't want to return to the user "500 Internal Server Error"

```
try:
    ...  # some code here might raise an exception
except Exception as e:  # catching exceptions
    print("Exception occurred:", repr(e))
```

https://docs.python.org/3.8/library/exceptions.html

Catching Exception and BaseException

```
try:
    raise Exception("My custom exception")
except Exception as e:  # catching Exception is not the best idea
    print("Exception occurred:", repr(e))
except BaseException as e:  # catching BaseException is even worse
    print("BaseException occurred:", repr(e))
```

Guess which one will be printed

```
try:
    input()
except Exception as e:
    print("Exception occurred:", repr(e))
except BaseException as e:
    print("BaseException occurred:", repr(e))
```

```
> BaseException occurred: KeyboardInterrupt()
```

anastasiatymo

Hierarchy of exceptions in Python

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      (...)
```

Handling exceptions

```
try:
    raise Exception("Something custom happened!!!")
except Exception as e:
    print("Printing exception", repr(e))
    raise Exception("I want my custom message!!!")
```

Handling exceptions

```
Traceback (most recent call last):
  File "my_awesome_code.py", line 2, in <module>
    raise Exception("Something custom happened!!!")
Exception: Something custom happened!!!

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "my_awesome_code.py", line 5, in <module>
    raise Exception("I want my custom message!!!")
Exception: I want my custom message!!!
Printing exception Exception('Something custom happened!!!')
```

anastasiatymo

Handling exceptions - raising from

```
try:
    raise Exception("Something custom happened!!!")
except Exception as e:
    print("Printing exception", repr(e))
    raise Exception("I want my custom message!!!") from e
```

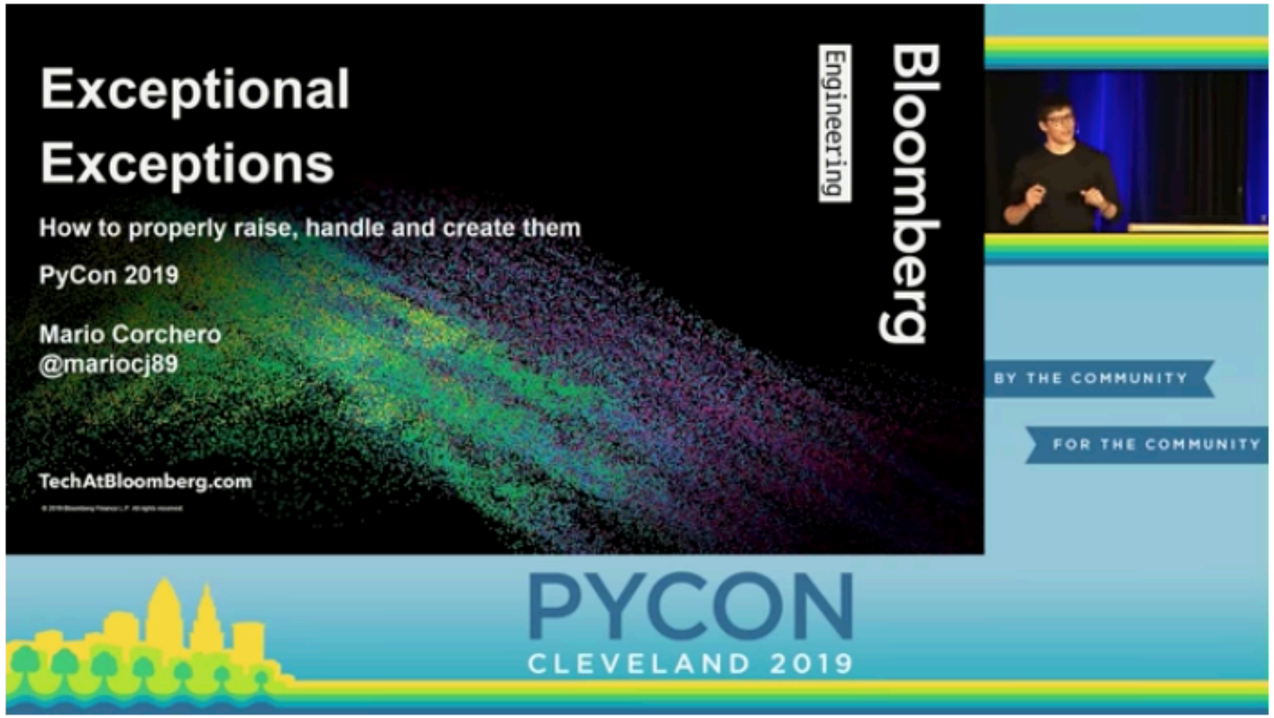
Handling exceptions - raising from

```
Traceback (most recent call last):
  File "my_awesome_code.py", line 2, in <module>
    raise Exception("Something custom happened!!!")
Exception: Something custom happened!!!

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "my_awesome_code.py", line 5, in <module>
    raise Exception("I want my custom message!!!") from e
Exception: I want my custom message!!!
Printing exception Exception('Something custom happened!!!')
```

Custom exceptions

```
class MyCustomException(Exception):
    pass


try:
    raise MyCustomException("Something custom happened!!!")
except MyCustomException as e:
    print("We are handling this exception here!", repr(e))
```

More to learn



PyCon 2019 | Exceptional Exceptions – How To Properly Raise, Handle And Create Them

Speaker: Mario Corchero

anastasiatymo

Logging

“Treat logs as event streams

Logs provide visibility into the behavior of a running app. (...)

Logs are the stream of aggregated, time-ordered events collected from the output streams of all running processes and backing services. Logs in their raw form are typically a text format with one event per line (though backtraces from exceptions may span multiple lines). Logs have no fixed beginning or end, but flow continuously as long as the app is operating.”

from The Twelve-Factor App

Main logging attributes

- when
- where
- what
- who
- outcome

How do we usually log something?

```
import logging

my_logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)

my_logger.info(
    "Hello Pythonista! Conference name %s, talk name %s, key_id = %s"
    % ("EuroPython", "Can we deploy yet?", "1234")
)
```

```
INFO:__main__:Hello Pythonista! Conference name EuroPython, talk name
Can we deploy yet?, key_id = 1234
```

anastasiatymo

How can we improve?

Maybe Structlog?

Logging with Structlog

```
import structlog

logger_structlog = structlog.get_logger()

logger_structlog.info(
    "Hello Pythonista!",
    key_id="1234",
    conference_name="EuroPython",
    talk_name="Can we deploy yet?",
)
```

```
2020-07-19 21:38.48 Hello Pythonista!          conference_name=EuroPython
                key_id=1234 talk_name=Can we deploy yet?
```

How can we improve?

Definitely Structlog!

Let's take a closer look

```python
import structlog

logger_structlog = structlog.get_logger(__name__)

logger_structlog = logger_structlog.bind(
    key_id="1234", conference_name="EuroPython", talk_name="Can we deploy yet?"
)
try:
    raise Exception("Oh, something went wrong...")
except Exception:
    logger_structlog.exception("logging exception")
```

```
2020-07-19 21:43.48 logging exception
conference_name=EuroPython key_id=1234 talk_name=Can we deploy yet?
Traceback (most recent call last):
  File "my_awesome_code.py", line 9, in <module>
    raise Exception("Oh, something went wrong...")
Exception: Oh, something went wrong...
```

anastasiatymo

Also in json format!

```
{
  "event": "Hello Pythonista!",
  "level": "europython",
  "logger": "test",
  "timestamp": "2020-07-19T19:47:03.514339Z"
}
```

Demo time!👩‍💻

More to learn about Structlog

Logging Rethought 2: The Actions of Frank Taylor Jr.

Translations: en en
Sat 20 April 2019
By Markus Holtermann

YouTube

anastasiatymo

Effective CI/CD

Continuous Integration

- test coverage
- reliability
- fault isolation
- transparency
- code quality
- faster development
- code review improvements

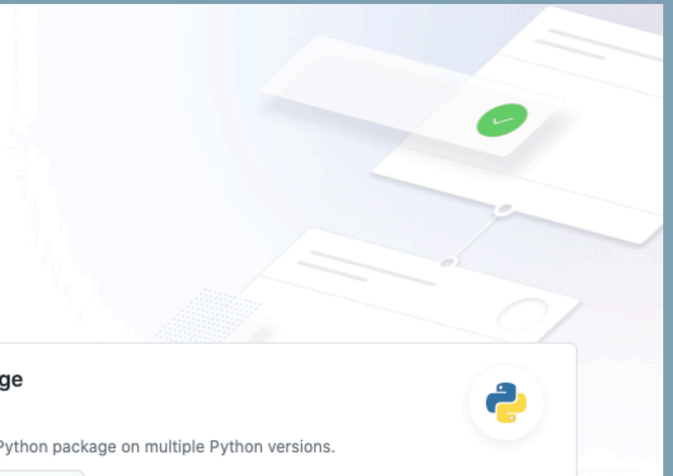
Continuous Integration

- self-hosted solution
- paid solution
- free for open-source

Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow template to get started.

Skip this and set up a workflow yourself →



Workflows made for your Python repository Suggested

Publish Python Package

By GitHub Actions

Publish a Python Package to PyPI on release.

Set up this workflow

```
python -m pip install --upgrade pip
pip install setuptools wheel twine
python setup.py sdist bdist_wheel
```

actions/starter-workflows Python

Python package

By GitHub Actions

Create and test a Python package on multiple Python versions.

Set up this workflow

```
python -m pip install --upgrade pip
pip install flake8 pytest
if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
```

actions/starter-workflows Python

Python application

By GitHub Actions

Create and test a Python application.

Set up this workflow

```
python -m pip install --upgrade pip
pip install flake8 pytest
if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
```

actions/starter-workflows Python

Django

By GitHub Actions

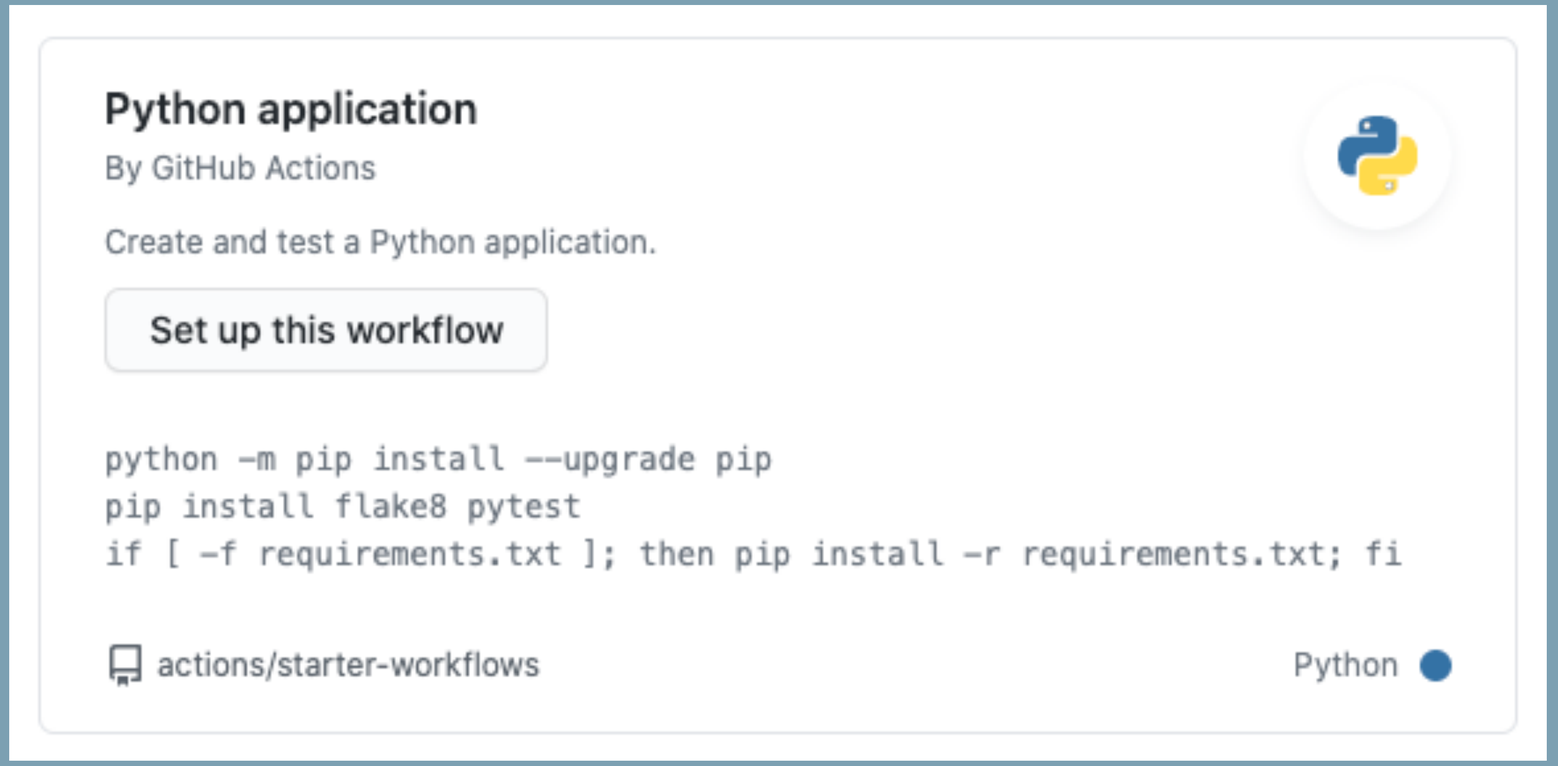
Build and Test a Django Project

Set up this workflow

```
python -m pip install --upgrade pip
pip install -r requirements.txt
python manage.py test
```

actions/starter-workflows Python

anastasiatymo



anastasiatymo

Demo time!👩‍💻

https://github.com/atymoshchuk/can_we_deploy_yet/blob/master/.github/workflows/python-app.yml

Docker? No Problem!

anastasiatymo

Secure your Docker images

- do not use root user
- use trusted and well-known images
- use COPY instead of ADD
- lint your Dockerfile
- save images in your docker registry and maintain them
- check authenticity of the docker image

Do not use root user

```
FROM ubuntu
ENV USER python # Default value
ENV GROUP python # Default value
RUN mkdir /app

# Create group, create user, add user to group
RUN groupadd -r "$GROUP" && useradd -r -g "$GROUP" -s /bin/bash "$USER"

WORKDIR /app
COPY . /app

# Change owner of directories and files to USER and GROUP
RUN chown -R "$USER":"$GROUP" /app

USER "$USER"  # Change user
```

Demo time!👩‍💻

More hints

Do you document your code?

"There is a secret that needs to be understood in order to write good software documentation: there isn't one thing called documentation, there are four."

from https://documentation.divio.com/

- tutorials,
- how-to guides,
- technical reference
- explanation

How to start?

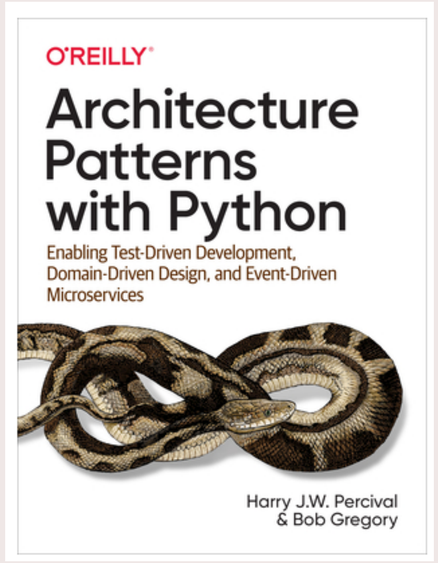
- start as simple as possible
- go to version controlled docs

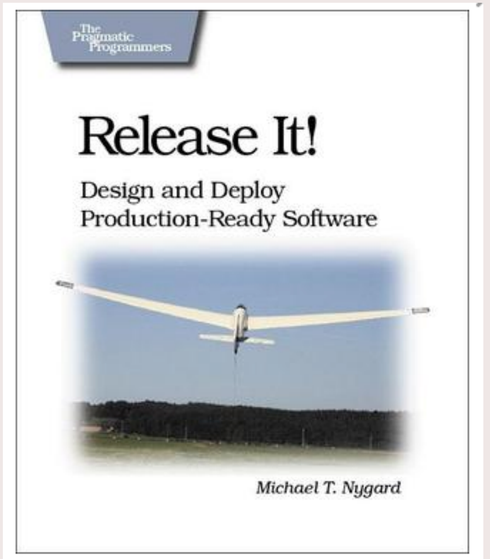
How to start?

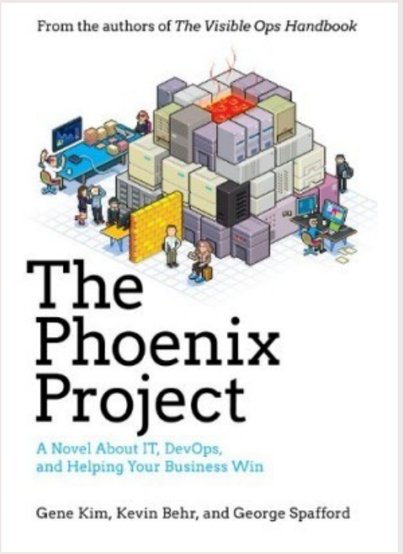
- start with Sphinx
- try Read The Docs

Demo time!👩‍💻

Books for further reading







... and many more referenced on https://atymo.me/

anastasiatymo

Thank you!

Questions?

anastasiatymo