ФОРМА ЗАДАНИЯ НА ВЫПОЛНЕНИЕ
КУРСОВОГО ПРОЕКТА (КУРСОВОЙ РАБОТЫ)

Санкт-Петербургский государственный политехнический университет

ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА
(КУРСОВОЙ РАБОТЫ)

студенту группы 3540904/90201 Дейлид Ивану Анатольевичу
(номер группы) *(фамилия, имя, отчество)*

1. Тема проекта (работы): *CI для open source пакетов Python*

2. Срок сдачи студентом законченного проекта (работы) *23 декабря 2019*

3. Исходные данные к проекту (работе): *Материалы лекций, официальная документация по Python, докумментация сервиса Travis CI.*

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов): введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения.

Примерный объём пояснительной записки *одна страница* машинописного текста.

5. Перечень графического материала (с указанием обязательных чертежей и плакатов): *отсутствует*

6. Консультанты _____

_____

7. Дата получения задания: «_5_».__октября_ 2013г.

Руководитель _____ *Баранов А.В.*

(подпись) *(инициалы, фамилия)*

Задание принял к исполнению _____ *Дейлид И. А.*
(подпись студента) *(инициалы, фамилия)*

05.10.19 *(дата)*

Санкт-Петербургский государственный политехнический университет
Институт информационных технологий и управления
Кафедра «Информационные и управляющие системы»

КУРСОВАЯ РАБОТА

CI для open source пакетов Python

Выполнил
студент гр. 3540904/90201 И. А. Дейлид

Руководитель
ст. преподаватель А.В.Баранов

«22» декабря 2019 г.

Санкт-Петербург
2019

Оглавление

Введение

Успех языка Python во многом связан с его простотой и большим сообществом разработчиков. В интернете можно найти огромное количество кода, пригодного для повторного использования. Энтузиасты в свое время смогли в том числе и решить множество фундаментальных недостатков языка, таких как низкая производительность при вычислениях, для чего были созданы пакеты с ПО для обработки матричных вычислений с помощью библиотек, скомпилированных на устройстве с помощью языка C++, пакеты, взаимодействующие с экосистемой операционной системы. Все это позволило получить на практике удобный и приемлемо быстрый язык программирования.

Однако, несмотря на огромное количество созданных пакетов, большинство распространяемого в интернете кода, в том числе кода, расположенного не на форумах, а в открытых репозиториях в виде проекта, не оформляются как пакет для языка Python, как правило разработчики просто делятся своим локальным проектам, не заботясь о том, имеются ли у пользователей необходимые зависимости, есть ли ограничения по версии языка, аппаратной платформе и т. п.

Если речь заходит о переиспользовании подобного открытого проекта в более крупном, необходимо встроить его в пайплайн разработки. В случае CI, делать это без оформления кода в виде полноценного Python проекта с возможность установки, описанием зависимости, структурированности и инструкций по сборке крайне не рекомендуется.

О том, как это можно сделать для проектов на языке Python и пойдет речь.

Структура проекта на языке Python

Хотя язык Python содержит, к примеру, официально принятый стандарт по стилю кодирования, единого стандарта для структурирования проекта нет. Тем не менее, существуют практики, частично перекочевавшие из других языков, которые рекомендуется применять для проектов на данном языке.

Одна из наиболее популярных книг, предлагает следующий вариант структуры проекта “Sample”:

README.rst	Файл с кратким описанием проекта
LICENSE	Файл с лицензией проекта
setup.py	Файл для установки проекта средствами Python
requirements.txt	Файл с зависимостями проекта
sample/__init__.py	Файл с инициализацией пакета, который выполняет все необходимые предварительные действия и оперирует другим программным кодом пакета
sample/core.py	Файл с ключевой функциональностью пакета
sample/helpers.py	Структурированная функциональность проекта
docs/conf.py	Файл с документацией о конфигурации проекта в виде Python скрипта
docs/index.rst	Файл с документацией проекта в формате языка разметки

tests/test_basic.py	Базовый тест проекта
tests/test_advanced.py	Продвинутый тест проекта
Makefile	Опционально, Makefile для сборки проекта с помощью утилиты make

Далее мы подробнее остановимся на каждом пункте

Readme.rst

Файл README является устоявшимся стандартом для любого репозитория с открытым исходным кодом. По его наличию или отсутствию можно судить о том, насколько репозиторий в принципе пригоден для повторного использования сторонним разработчиком.

Большинство хостингов проектов поддерживают автоматическое чтение README файла и преобразования языка разметки в HTML формат для отображения на странице в браузере сразу после структуры проекта.

Отдельно хочется выделить формат файла. Наиболее популярный язык разметки сейчас - Markdown с расширением, как правило, .md. Он рекомендуется к использованию, например, сервисом github, где для него приводят отдельную страницу документации. Однако для Python проектов предпочтительнее является формат reStructedText. С исторической точки зрения это обусловлено тем, что система автоматической генерации документации Sphinx, изначально созданной для языка Python, использовала данный формат в качестве формата для файлов с разметкой. Практическое же объяснение заключается в том, что хостинг Python проектов pypi.org, который теперь является официальным, использует в качестве превью страницы каждого пакета файл README.rst, в случае

использования Markdown разметки, она не будет сконвертирована в HTML.

LICENSE

Для каждого более менее крупного проекта обязательным является файл с описанием лицензии. В случае, если вы используете open-source проект в своем промышленном проекте, важно обращать внимание на тип лицензирования, чтобы не столкнуться с юридическими ограничениями в дальнейшем. Наиболее популярная лицензия открытых проектов Python на текущий момент лицензия MIT, позволяющая использовать программный код как в открытом ПО, так и в коммерческом проприетарном.

setup.py

Данный файл является ключевым для установки проекта. Файл содержит вызов функции setup() из стандартного пакета setuptools. Аргументы данной функции содержат краткое описание проекта, документацию, зависимости от версий сторонних проектов, файлы для тестирования, требуемую версионность языка и аппаратную платформу, лицензию, ключевые слова, версию пакета, тип пакета (библиотека, утилиты) и многое другое.

Пример файла для установки проекта можно увидеть ниже

```
from setuptools import setup, find_packages
setup(
    name="HelloWorld",
    version="0.1",
    packages=find_packages(),
    scripts=["say_hello.py"],

    # Project uses reStructuredText, so ensure that the
docutils get
    # installed or upgraded on the target machine
    install_requires=["docutils>=0.3"],

    package_data={
        # If any package contains *.txt or *.rst files,
include them:
        "": ["*.txt", "*.rst"],
        # And include any *.msg files found in the
"hello" package, too:
        "hello": ["*.msg"],
    },

    # metadata to display on PyPI
    author="Me",
    author_email="me@example.com",
    description="This is an Example Package",
    keywords="hello world example examples",
    url="http://example.com/HelloWorld/",   # project
home page, if any
    project_urls={
        "Bug Tracker":
"https://bugs.example.com/HelloWorld/",
        "Documentation":
"https://docs.example.com/HelloWorld/",
```

```
        "Source Code":
"https://code.example.com/HelloWorld/",
    },
    classifiers=[
        "License :: OSI Approved :: Python Software
Foundation License"
    ]

    # could also include long_description, download_url,
etc.
)
```

Для того, чтобы протестировать проект, следует создать чистое виртуальное окружение под используемую вами версию языка, после чего выполнить команду `pip install path/to/package`

requirments.txt

Данный файл хранит в себе все версии установленных в окружении библиотек. Получить его можно с помощью команды `pip3 freeze`

Пример генерируемого файла:

```
aiohttp==3.6.1
async-timeout==3.0.10
Cython==0.29.13
dataclasses==0.6
decorator==4.4.0
defusedxml==0.6.0
Django==2.2.5
...
yarl==1.3.0
zero-cm==1.0.0
```

Минусы данного подходы заключаются в том, что версия указывается однозначно, хотя проект может использовать, к примеру, версию не ниже версии 1.0.0, записать в файл версию 1.1.0, и сторонний проект, который конфликтует с версией 1.1.0 поломает себе зависимости, хотя этого можно было избежать.

В случае создания данного файла в ручном режиме можно указать соотношения версий (не ниже, не выше, ниже или равно).

Установить библиотеки необходимых версий можно с помощью команды `pip3 install -r requirements.txt`

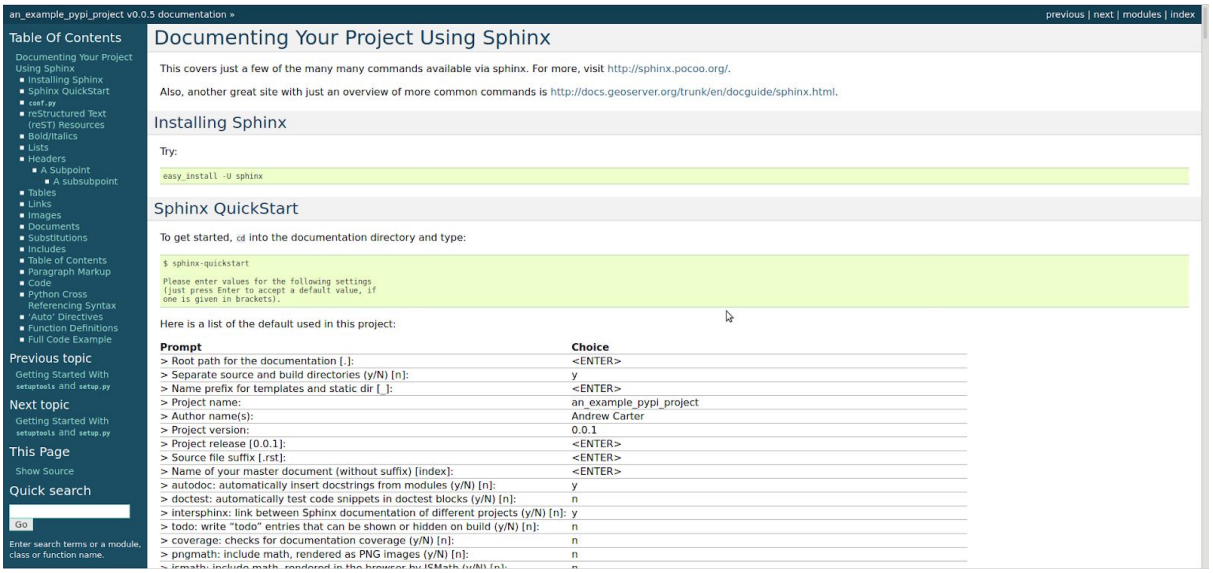
sample

Размещаемый код должен удовлетворять требованиям стандартов языка Python, так его синтаксис должен соответствовать принятому стандарту PEP8, который можно проверить с помощью множества свободных утилит, а также крайне рекомендуется использовать стандарт PEP484 с аннотацией типов. Это позволит провести тестирование на ошибку в типизации данных, что в значительной степени поднимет качество тестирования.

docs

Документация является обязательной частью любого проекта, крупного или небольшого. Исторически средством автоматической генерации документации является Sphinx, однако сейчас это позволяет делать в том числе и Doxygen.

Пример страницы сгенерированной в Sphinx



В случае использования Sphinx атрибуты для документации аккуратно встраиваются в стандартные средства документирования кода в Python

Пример кода для Sphinx

```
def my_fn(foo, bar=True):
    """A really useful function.

    Returns None
    """
```

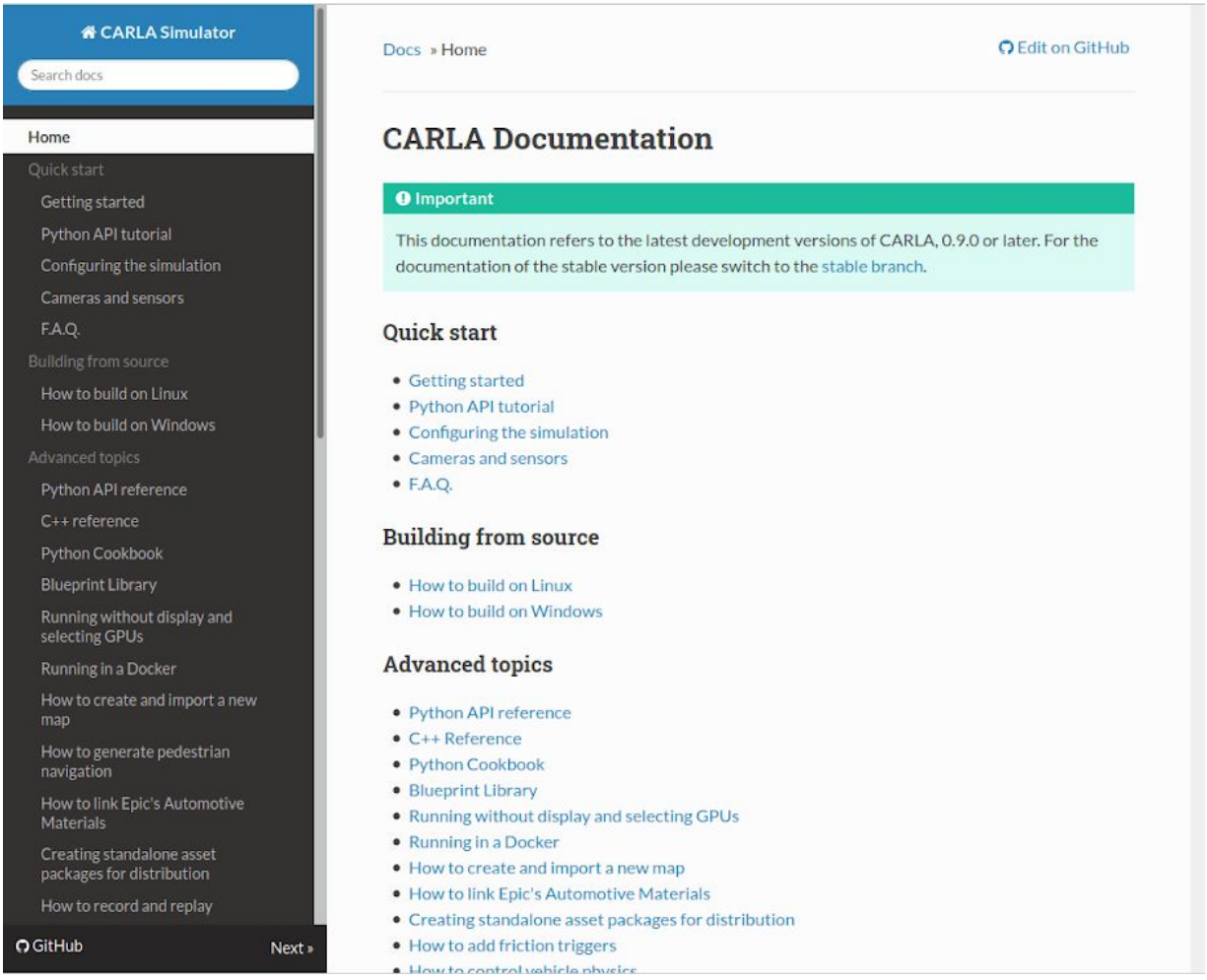
Пример кода для Doxygen

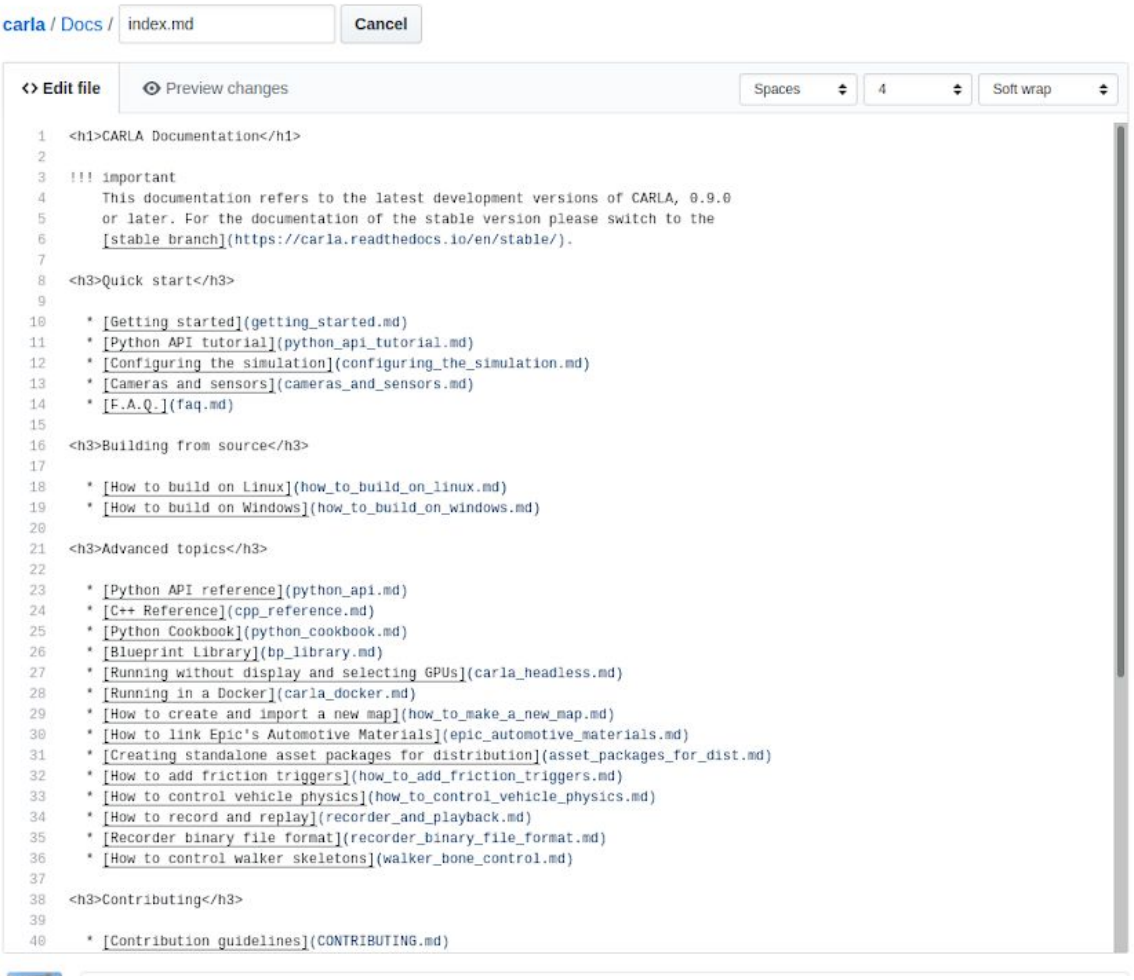
```
## @package pyexample
#  Documentation for this module.
#
#  More details.

## Documentation for a function.
#
#  More details.
def func():
```

```
    pass
```

Если же столь большой документации не требуется - можно ограничится одним или несколькими файлами в формате reStructedText

Онлайн документацию принято визуализировать с помощью специальных сервисов, поддерживающих автоматическое чтение директории /docs проекта и отображение в виде html страниц





tests

Для автоматизации тестов, предлагается использовать модуль unittest стандартной библиотеки Python. Основными элементами данного модуля являются **test case** - минимальный блок тестирования. Он проверяет ответы для разных наборов данных, **test suite** - он используется для объединения тестов, которые должны быть выполнены вместе, **test runner** - компонент, который управляет выполнением тестов и предоставляет пользователю результат.

Пример файла test.py

```
import unittest

class TestStringMethods(unittest.TestCase):

  def test_upper(self):
      self.assertEqual('foo'.upper(), 'FOO')

  def test_isupper(self):
      self.assertTrue('FOO'.isupper())
      self.assertFalse('Foo'.isupper())

  def test_split(self):
      s = 'hello world'
      self.assertEqual(s.split(), ['hello', 'world'])
      # Проверим, что s.split не
работает, если разделитель - не
строка
      with self.assertRaises(TypeError):
          s.split(2)

if __name__ == '__main__':
    unittest.main()
```

Существует также и другие варианты юнит-тестирования, однако к данной теме это не относится.

Makefile

Иногда код проекта может содержать файл Makefile. В нем может производиться установка компонентов из репозиториев пакетного менеджера среды, объявление переменных среды, сборка объектных

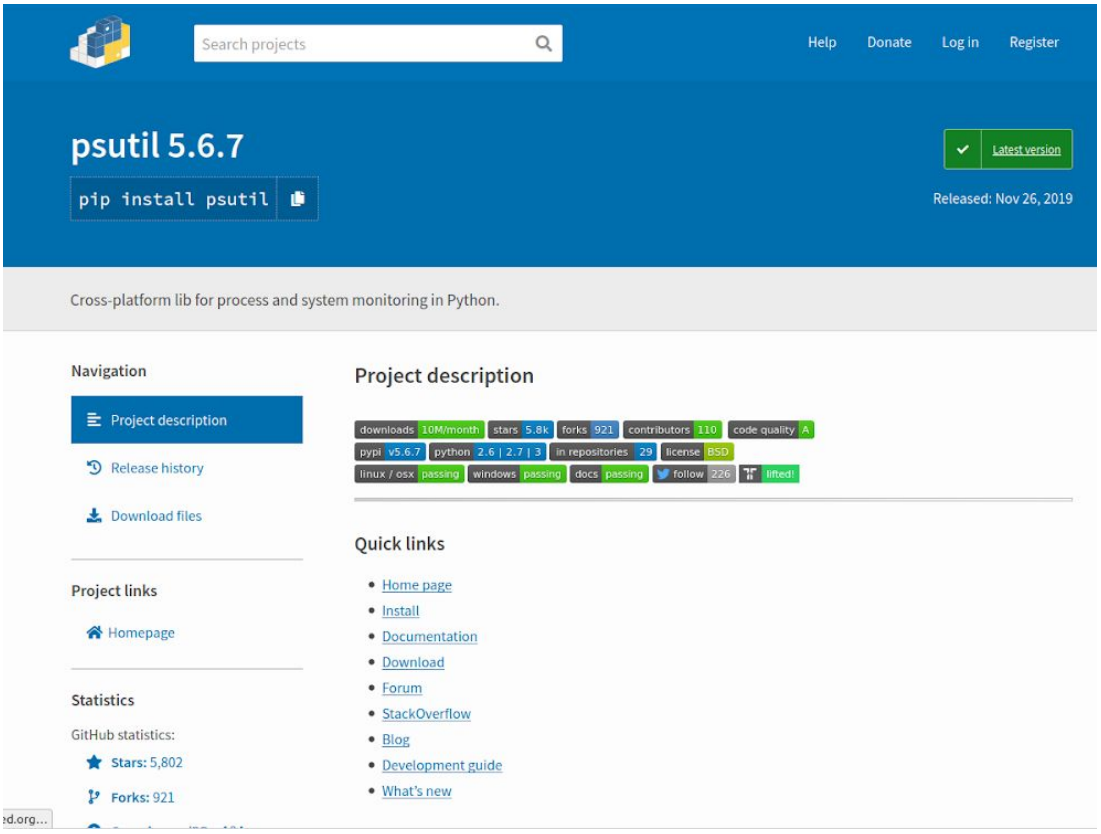
модулей, запуск тестирования или генерации автоматической документации кода.

Виды вариантов установки проекта

Pip

Начиная с версии языка 3.5 PIP является утилитой, поставляемой вместе с интерпретатором и официально поддерживаемой комитетом стандартизации. PIP позволяет читать файлы зависимостей и setup.py файлы, выполняя установку пакетов.

По умолчанию pip имеет несколько репозиториев, откуда он скачивает пакеты и зависимости. Располагаются они на pypi.org, любой желающий может выложить свой проект туда.



Кроме того можно указать отдельный репозиторий, к примеру, созданный внутри компании. А также при желании можно устанавливать пакеты непосредственно с git репозитория, как локального, так и удаленного.

```
pip install git+git_repository_url@branch_name
```

easy install

Ранее предоставлялась возможность установки проекта с помощью специальной утилиты, сейчас данная функциональность является устаревшей.

CI для Open Source проекта на языке Python

Travis CI

Зачастую для python проектов используется Travis CI - CI-сервис, используемый для создания и тестирования проектов. Он автоматически обнаруживает новые коммиты, сделанные и отправленные в репозиторий GitHub. И после каждой новой фиксации кода Travis CI будет строить проект и запускать тесты соответственно. Travis CI интегрируется как с github, так и pypi.org, что делает его очень удобным для тестирования и распространения python проектов с открытым исходным кодом.

Из ключевых особенностей можно выделить следующие:

- Быстрая установка
- Live build views для мониторинга проектов GitHub
- Подтянуть запрос поддержки

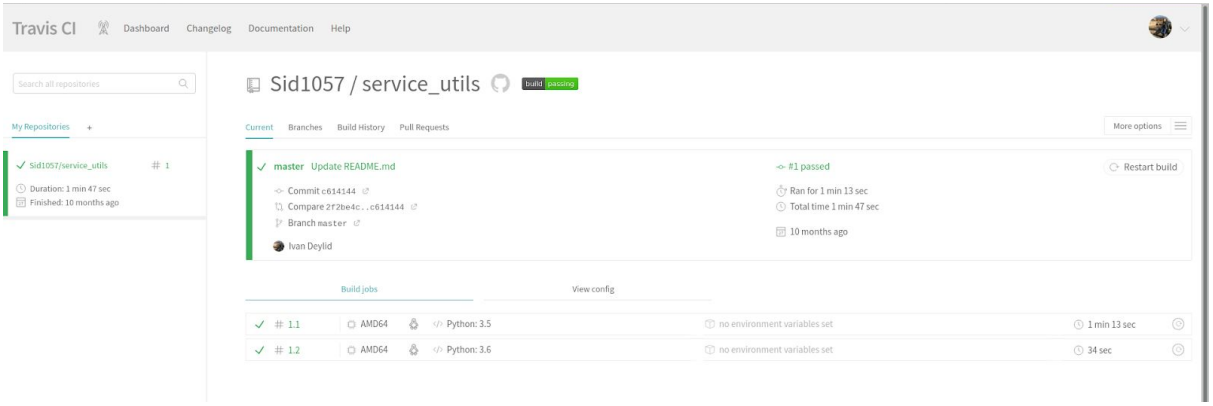
- Развертывание в нескольких облачных сервисах
- Предустановленные службы базы данных
- Автоматическое развертывание при прохождении сборки
- Чистые виртуальные машины для каждой сборки
- Поддерживает macOS, Linux и iOS

Для подключения системы к репозиторию необходимо произвести его регистрацию и добавить yaml конфиг, который будет указывать на параметры тестирования:

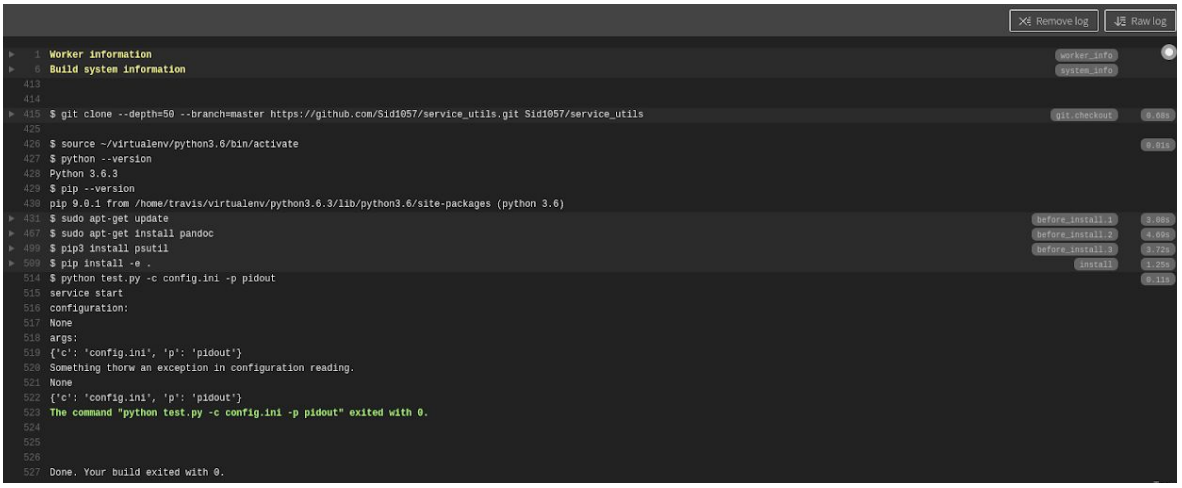
.travis.yaml

```
language: python

python:
    - "3.5"
    - "3.6"

before_install:
    - sudo apt-get update
    - sudo apt-get install pandoc
    - pip3 install psutil

install:
    - pip install -e .

script: python test.py -c config.ini -p pidout

after_failure: echo "Test failed"
```

Пример информационной страницы о сборке:



Пример лога сборки проекта после очередного коммита



Существуют множество других сред для CI, большинство из них предоставляются бесплатно для open source проектов, вот некоторые из них, которые поддерживают язык Python и бесплатны для применения в OpenSource проектах:

Jenkins

Jenkins - это сервер автоматизации с открытым исходным кодом, в котором происходит центральная сборка и непрерывная интеграция. Это автономная Java-программа с пакетами для Windows, macOS и других Unix-подобных операционных систем. Имея сотни доступных плагинов,

Jenkins поддерживает создание, развертывание и автоматизацию проектов разработки программного обеспечения.

Ключевые особенности Jenkins:

- Простая установка и обновление на различных ОС
- Простой и удобный интерфейс
- Расширяемый с огромным ресурсом плагина сообщества
- Простая настройка среды в пользовательском интерфейсе
- Поддержка распределенных сборок с архитектурой ведущий-ведомый
- Построить графики на основе выражений
- Поддержка оболочек и выполнения команд Windows на этапах предварительной сборки
- Поддерживает уведомление о статусе сборки

GitLab

GitLab - это набор инструментов для управления различными аспектами жизненного цикла разработки программного обеспечения. Основным продуктом является веб-менеджер Git-репозитория с такими функциями, как отслеживание проблем, аналитика и Wiki.

GitLab позволяет запускать сборки, запускать тесты и развертывать код при каждом коммите или нажатии. Вы можете создавать задания на виртуальной машине, в контейнере Docker или на другом сервере.

Ключевые особенности GitLab:

- Просмотр, создание и управление кодами и данными проекта с помощью инструментов ветвления
- Проектируйте, разрабатывайте и управляйте кодами и проектными данными из единой распределенной системы контроля версий, обеспечивая быструю итерацию и доставку деловых ценностей
- Обеспечивает единый источник правдивости и масштабируемости для совместной работы над проектами и кодом
- Помогает командам доставки полностью использовать CI, автоматизируя сборки, интеграцию и проверку исходных кодов.
- Обеспечивает сканирование контейнеров, статическое тестирование безопасности приложений (SAST), динамическое тестирование безопасности приложений (DAST) и сканирование зависимостей для обеспечения безопасности приложений наряду с соблюдением лицензий.
- Помогает автоматизировать и сократить выпуск и доставку приложений

Buddy

Buddy - это программное обеспечение CI / CD, которое создает, тестирует, развертывает веб-сайты и приложения с кодом из GitHub, Bitbucket и GitLab. В нем используются контейнеры Docker с предустановленными языками и средами для разработки, а также DevOps для мониторинга и уведомления о действиях.

Основные характеристики Buddy:

- Простая настройка изображений на основе Docker в качестве тестовой среды
- Интеллектуальное обнаружение изменений, современное кэширование, параллелизм и всесторонняя оптимизация
- Создавать, настраивать и повторно использовать сборки и тестовые среды
- Простые и зашифрованные, фиксированные и настраиваемые области: рабочее пространство, проект, конвейер, действия
- Подключаемые сервисы с Elastic, MariaDB, Memcached, Mongo, PostgreSQL, RabbitMQ, Redis, Selenium Chrome и Firefox
- Монитор с прогрессом в режиме реального времени и журналами, неограниченной историей
- Управление рабочими процессами с помощью шаблонов для клонирования, экспорта и импорта конвейеров
- Первоклассная поддержка Git и интеграции

Spinnaker

Spinnaker - это многооблачная платформа непрерывной доставки, которая поддерживает выпуск и развертывание изменений программного обеспечения в разных облачных провайдерах, включая AWS EC2, Kubernetes, Google Compute Engine, Google Kubernetes Engine, Google App Engine и т. Д.

Ключевые особенности Spinnaker:

- Создает конвейеры развертывания, которые запускают интеграционные и системные тесты, увеличивают и уменьшают

группы серверов и отслеживают развертывание. Запуск конвейеров через события Git, Jenkins, Travis CI, Docker, cron или другие конвейеры Spinnaker.

- Создание и развертывание неизменяемых образов для ускорения развертывания, упрощения отката и устранения сложных для устранения проблем смещения конфигурации
- Свяжите свои выпуски с такими сервисами мониторинга, как Datadog, Prometheus, Stackdriver или SignalFx, используя их метрики для анализа канареек
- Установите, настройте и обновите ваши экземпляры Spinnaker с помощью Halyard - инструмента администрирования Spinnaker CLI
- Настройка уведомлений о событиях для электронной почты, Slack, HipChat или SMS (через Twilio)

Buildbot

Buildbot - это «основанная на Python инфраструктура CI», которая автоматизирует циклы компиляции и тестирования для проверки изменений кода, а затем автоматически перестраивает и тестирует дерево после каждого изменения. Поэтому проблемы со сборкой быстро выявляются.

Ключевые особенности Buildbot:

- Автоматизируйте системы сборки, развертывания приложений и управления сложными процессами выпуска программного обеспечения

- Поддержка распределенного параллельного выполнения на нескольких платформах, гибкая интеграция с системами контроля версий, расширенные отчеты о состоянии
- Работает на разных подчиненных платформах.
- Произвольный процесс сборки и обработки проектов с использованием C и Python
- Минимальные требования к хосту: Python и Twisted
- Примечание: Buildbot перестанет поддерживать Python 2.7 и требует перехода на Python 3.

Заключение

Существует множество инструментов CI на выбор разработчика. Однако важным шагом на пути к построению интеграционного тестирования является правильное использование языка, его средств и общепринятых сообществом правил. В случае нарушения целостности проекта - легко сделать свой репозиторий непригодным для последующего использования, а в случае коммерческой разработки плодить неподдерживаемый legacy код. Если же грамотно использовать все встроенные средства языка для развертывания приложения, поддержка интеграционного тестирования будет происходить как явление само собой разумеющиеся с минимальным количеством препятствий на пути.