BIGDATA TEAM

HW #09: Web Service Logs

1. Описание задания

В данном ДЗ вам нужно настроить логирование и обработку метрик функционирования Web-сервиса. Когда вы пройдете путь от запуска веб-сервиса до анализа логов и построения метрик, то получите полезный опыт осознания и рефакторинга схемы логирования для production сервисов.

Для решения задания будет полезно вспомнить:

- как делать запросы с помощью requests и парсить HTML с помощью bs4, lxml и/или XPath;
- паттерны проектирования (например Decorator и Context Manager);
- как настраивать логирование приложений

Для решения задания пригодятся умения:

- по настройке логирования Flask-приложений;
- по деплою и тестированию web-сервисов;
- по работе с fixed-size базами данных для хранения и агрегации временных рядов;

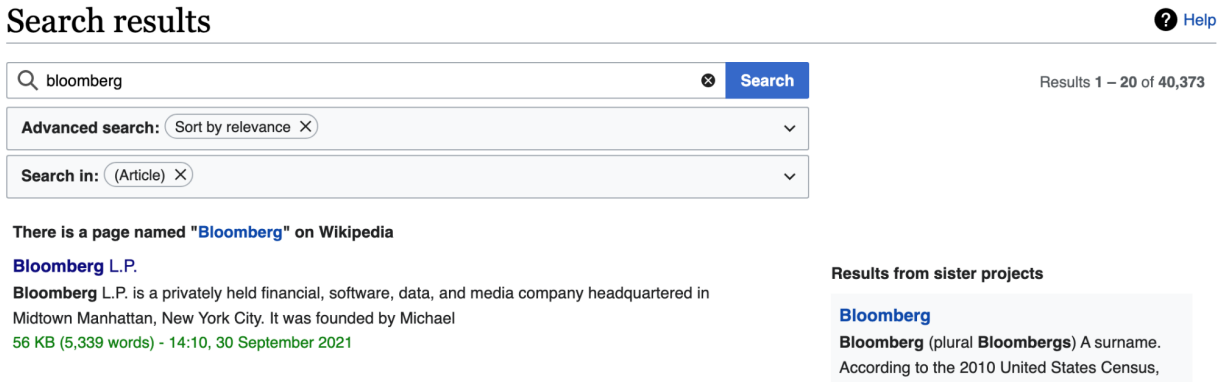
1.1. Поисковый движок на основе Википедии

За основу возьмем прокси-сервис поиска по Википедии. Пользовательские запросы будем проксировать Википедии и выводить результат пользователю. Web-сервис (Flask-приложение) должен предоставлять следующие route'ы:

endpoint: /api/search?query=<multi-word query>
формат ответа: JSON
описание: взять пользовательский запрос (query), проксировать запрос поисковый запрос Википедии по адресу:

https://en.wikipedia.org/w/index.php?search=<query>

В полученной выдаче распарсить информацию о числе найденных документов по запросу пользователя. Пример выдачи:



Найденное число документов вернуть в формате json API:
{"version": 1.0[1], "article_count": 40373}

Если по запросу ничего не найдено, то запрос должен считаться валидным и выдавать:
{"version": 1.0, "article_count": 0}

Примеры запросов указаны в файле:

- github:big-data-team/python-course/../wikipedia_search_queries.txt

Вам нужно прогнать все запросы, указанные в файле, и получить логи сервиса в согласованном формате, чтобы на основе логов можно было удобно посчитать метрики и настроить мониторинг. Интересующие метрики:

1. Среднее число документов (статей Википедии) на один запрос пользователя;
2. Среднее время поиска или среднее время ожидания ответа;
3. (опционально) гистограмма по частоте задания вопросов по часам;

Обработка исключительных ситуаций:

- В случае обращения по несуществующему route должен возвращаться **код 404** с текстом “This route is not found”;
- В случае недоступности wikipedia.org (ожидаемый exception - requests.**exceptions.ConnectionError**), необходимо возвращать ошибку 503. Для этой ошибки должен быть зарегистрирован обработчик, который будет возвращать сообщение “Wikipedia Search Engine is unavailable”. Для тестирования этого поведения необходимо будет запатчить обращения “requests.get”.

[1] Сразу начнем версионировать наш API, как в лучших практиках.

1.2. Требования к реализации

Flask приложение должно называться app и создаваться с помощью конструкции:

```
app = Flask(__name__)
```

Настройте логирование вашего Flask приложения на консоль в следующем формате:

```
format: "%(asctime)s.%(msecs)03d %(name)s %(levelname)s %(message)s"
datefmt: "%Y%m%d_%H%M%S"
```
[2]

Добавьте логирование следующих событий в ваше приложение:

уровень: DEBUG
сообщение: ”start processing query: %s”, аргументы - query

уровень: INFO
сообщение: ”found %s articles for query: %s”, аргументы - article_count, query

уровень: DEBUG
сообщение: ”finish processing query: %s”, аргументы - query

Прогоните предоставленные запросы, указанные в файле:

- github:big-data-team/python-course/../wikipedia_search_queries.txt

Полученный в терминале лог сохраните в файл wiki_search_app.log. Возьмите этот файл за пример для разработки консольного приложения для настройки мониторинга и подсчета метрик.

Консольное приложение должно предоставлять возможность парсить логи и отправлять метрики в Graphite. Поскольку запущенный Graphite в тестовом окружении дополнительное развлечение, то в целях наработки практики обойдемся без запущенного сервиса и подготовим команды, которые можно просто скопировать для загрузки данных в Graphite (по умолчанию host:localhost, port:2003):

```
$ python3 task_*_graphite_cli.py --process wiki_search_app.log --host localhost --port 2003
```

[2] Удобный формат, который легко читать людям и легко парсить машинам

```
...
echo "wiki_search.article_found 40369 1600042876" | nc -N localhost 2003
echo "wiki_search.complexity 3.080[3] 1600042876" | nc -N localhost 2003
...
```

Где на каждый запрос пользователя мы будем получать две метрики: число найденных документов и сложность обработки запроса (время ожидания:

```
finish_time - start_time
```

в секундах). Обе метрики отправляются с временной меткой, соответствующей `finish_time`.

Обратите внимание, что:

- в логах некоторые запросы могут идти вперемешку, см. пример с занятия - обработка двух параллельных запросов;
- некоторые запросы могут повторяться, но гарантируется, что одинаковые запросы не будут обрабатываться параллельно;
- время (complexity) нужно считать с точностью до миллисекунд;
- время начала обработки запроса и его конца может переходить через сутки;
- поскольку при выкатке в dev и production у логгера могут быть разные настройки, то имя логгера приложения может отличаться от вашего. Примеры названия логгеров

  ```
  task_Surname_Name_web_service_log
  task_Anothersurname_Name_web_service_log
  awersome_web_service
  ...
  ```

На все эти граничные случаи нужно подготовить юнит-тесты.

2. Рекомендации

Рекомендации по разработке:

- следите за качеством кода и проверяйте “глупые” ошибки с помощью pylint, следите за поддерживаемостью и читаемостью кода;
- держите уровень покрытия кода тестами на уровне 80+%, следуйте TDD (сначала тесты, потом реализация);
- отделяйте фазу рефакторинга от фазы добавления новой функциональности.
  - фиксируем функциональность, все тесты зеленые;

[3] С точностью до третьего знака после запятой. Решение с “round” работать не будет, поскольку round(0.2300, 3) == 0.23, а не “0.230”.

    - проводим рефакторинг;
    - по окончании фазы рефакторинга снова все тесты зеленые;
- следите за скоростью выполнения unit-test'ов, несколько секунд - это хорошо, в противном случае нужно уменьшать размер тестируемых датасетов или разделять тесты на фазы (см. обсуждение про mark.slow);

2.1. Ретроспектива

По результатам (или даже по ходу) выполнения задания, ответьте на следующие вопросы:

1. Почему текущая модель логирования плохая (неудобная) с учетом того, какие метрики мы хотим считать? Как это можно легко исправить?
2. Почти наверняка вам пришлось использовать strptime для преобразования текстового представления времени в объект типа datetime. Изучите документацию logging.Formatter и скажите каким образом получать объекты типа unix time stamp, которые можно напрямую кастовать к float/int?

3. Критерии оценивания

Балл за задачу складывается из:

- **40%** - правильная реализация API Web-сервиса
- **20%** - качество покрытия юнит-тестами web-сервиса (без доступа к Интернету, все обращения requests.get мокаем)
    - оценка качества проводится автоматически вызовом pytest:
        - `PYTHONPATH=. pytest -v --cov=task_*_web_service_log test_*_web_service_log.py --cov-report term-missing`
        - уровень покрытия тестами должен быть выше 80%
        - проверяем код Python версии 3.7 с помощью `pytest==6.0.1`
        - точная формула: 20% x min([test_coverage / 0.8], 1.0)
- **20%** - правильная реализация парсинга и построения метрик
- **10%** - качество покрытия юнит-тестами web-сервиса
    - оценка качества проводится автоматически вызовом pytest:
        - `PYTHONPATH=. pytest -v --cov=task_*_graphite_cli test_*_graphite_cli.py --cov-report term-missing`
        - уровень покрытия тестами должен быть выше 80%
        - проверяем код Python версии 3.7 с помощью `pytest==6.0.1`
        - точная формула: 20% x min([test_coverage / 0.8], 1.0)
- **10%** - поддерживаемость и читаемость кода
    - в общем случае см. Clean Code и Google Python Style Guide
    - оценка качества будет проводиться автоматическим вызовом pylint:
        - `pylint task_*.py --extension-pkg-whitelist=lxml`[4] `--disable=no-member`[5]
        - качество кода должно оцениваться выше 8.0 / 10.0
        - проверяем код Python версии 3.7 с помощью `pylint==2.5.3`
        - точная формула: 10% x min([lint_quality / 8.0], 1.0)

Discounts (скидки и другие акции):

- **100%** за плагиат в решениях (всем участникам процесса)
- **100%** за посылку решения после hard deadline
- **30%** за посылку решения в после soft deadline и до hard deadline
- **5%** за каждую посылку после 2й посылки в день (каждый день можно делать до 2х посылок без штрафа)

Пример работы системы штрафов:

[4] pylint не умеет делать introspection исходников на C, вызываемых через Factory Method
[5] pylint ругается на использование app.logger

День	Посылка	Штраф
День 1	Посылка 1	Без штрафа
День 1	Посылка 2	Без штрафа
День 1	Посылка 3	-5%
День 2	Посылка 4	Без штрафа
День 2	Посылка 5	Без штрафа
День 3	Посылка 6	Без штрафа
День 3	Посылка 7	Без штрафа
День 3	Посылка 8	-5%
День 3	Посылка 9	-5%
Итоговый штраф: -15%		

Для подсчета финальной оценки **всегда** берется **последняя** оценка из Grader.

4. Инструкция по отправке задания

Оформление задания:

- Код задания (Short name): **HW09:Web Service Logs**
- Выполненное ДЗ запакуйте в архив **PY-MADE-2021-Q4**_<Surname>_<Name>_HW#**.zip**, пример -- **PY-MADE-2021-Q4**_Dral_Alexey_**HW09**.zip. (Проверяйте отсутствие пробелов и невидимых символов после копирования имени отсюда.[6]) Если ваше решение лежит в папке my_solution_folder, то для создания архива hw.zip на Linux и Mac OS выполните команду[7]:
    - `zip -r hw.zip my_solution_folder/*`
- На Windows 7/8/10: необходимо выделить все содержимое директории my_solution_folder/ нажать правую кнопку мыши на одном из выделенных объектов, выбрать в открывшемся меню "Отправить >", затем "Сжатая ZIP-папка". Теперь можно переименовать архив.
- Решение задания должно содержаться в одной папке.

[6] Онлайн инструмент для проверки: https://www.soscisurvey.de/tools/view-chars.php
[7] Флаг -r значит, что будет совершен рекурсивный обход по структуре директории

- Перед проверкой убедитесь, что дерево вашего архива выглядит так:
  - `| PY-MADE-2021-Q4_<Surname>_<Name>_HW09.zip`
  - `| ---- task_<Surname>_<Name>_web_service_log.py`
  - `| ---- test_<Surname>_<Name>_web_service_log.py`
  - `| ---- task_<Surname>_<Name>_graphite_cli.py`
  - `| ---- test_<Surname>_<Name>_graphite_cli.py`
  - `| ---- *.{html,txt,out,log.example}`[8]
  - При несовпадении дерева вашего архива с представленным деревом, ваше решение не будет возможным автоматически проверить, а значит, и оценить его.
- Для того, чтобы сдать задание, необходимо:
  - Зарегистрироваться и залогиниться в сервисе Everest
  - Перейти на страницу приложения: MADE Python Grader
  - Выбрать вкладку Submit Job (если отображается иная).
  - Выбрать в качестве “Task” значение: **HW09:Web Service Logs**[9]
  - Загрузить в качестве “Task solution” файл с решением
  - В качестве Access Token указать тот, который был выслан по почте
- **Перед отправкой задания**, оставьте, пожалуйста, отзыв о домашнем задании по ссылке: https://rebrand.ly/pymade2021q4_feedback_hw. Это позволит нам скорректировать учебную нагрузку по следующим заданиям (в зависимости от того, сколько часов уходит на решение ДЗ), а также ответить на интересующие вопросы.

Внимание: если до дедлайна остается меньше суток, и вы знаете (сами проверили или коллеги сообщили), что сдача решений сломана, обязательно сдайте свое решение, прислав нам ссылку на выполненное задание (Job) на почту с темой письма “Short name. ФИО.”. Например: “**HW09:Web Service Logs**. Иванов Иван Иванович.” Таким образом, мы сможем увидеть какое решение у вас было до дедлайна и сможем его оценить. Пример ссылки:

- https://everest.distcomp.org/jobs/67893456230000abc0123def

Любые вопросы / комментарии / предложения пишите согласно предложениям на портале.

Всем удачи!

[8] Архив с тестовыми данными должен занимать **менее 200 КБ** пространства на жестком диске
[9] Сервисный ID: python.web_service_log

5. FAQ (часто задаваемые вопросы)

"You are not allowed to run this application", что делать?

Если Вы видите надпись "You are not allowed to run this application" во вкладке Submit Job в Everest, то на данный момент сдача закрыта (нет доступных для сдачи домашних заданий, по техническим причинам или другое). Попробуйте, пожалуйста, еще раз через некоторое время. Если Вы еще ни разу не сдавали, у коллег сдача работает, но Вы видите такое сообщение, сообщите нам об этом.

Grader показывает 0 или < 0, а отчет (Grading report) не помогает решить проблему

Ситуации:

- система оценивания показывает оценку (Grade) < 0, а отчет (Grading report) не помогает решить проблему. Пример: в случае неправильно указанного access token система вернет -401 и информацию о том, что его нужно поправить;
- система показывает 0 и в отчете (Grading report) не указано, какие тесты не пройдены. Пример: вы отправили невалидный архив (rar вместо zip), не приложили нужные файлы (или наоборот приложили лишние - временные файлы от Mac OS и т.п.), рекомендуется проверить содержимое архива в консоли:

  ```
  unzip -l your_solution.zip
  ```

Если Вы столкнулись с какой-то из них присылайте ссылку на выполненное задание (Job) в чат курса. Пример ссылки:

https://everest.distcomp.org/jobs/67893456230000abc0123def

Как правильно настроить окружение, чтобы оно совпадало с тестовым окружением?

1. Если еще не установлено, то установите conda https://docs.conda.io/projects/conda/en/latest/user-guide/install/
2. Настройте окружение для разработки на основе README.md курса https://github.com/big-data-team/python-course
3. Скачайте необходимые датасеты для выполнения задания https://github.com/big-data-team/python-course#study-datasets

6. Дополнительные задания (не на оценку)

PEP 3333 -- Python Web Server Gateway Interface v1.0.1

Изучите стандарт для расширения кругозора:

- PEP 3333 -- Python Web Server Gateway Interface v1.0.1

Исправляем логирование

В рамках раздела 2.1 вы должны были ответить на ряд вопросов. Пользуясь ответом на указанные вопросы:

1. Исправьте формат логирования;
2. Добавьте удобное событие, которое упрощает построение метрик с помощью декоратора или менеджера контекста (для подсчета времени).

Насколько упростился код, вывод и кодовая база всего проекта в совокупности (включая инструмент для парсинга логов)?

Какие еще предложения по упрощению парсинга вы можете предложить? Рекомендуется использоваться хештег #hw10 для удобства обсуждения и поиска релевантных сообщений.

Wikipedia Search Extended API

Flask может иметь особенности обработки нескольких ключей в URL. Для тренировки этих навыков, рекомендуется расширить API для работы с запросами в следующем формате:

endpoint: /api/extended_search?word=python&word=network&...
формат ответа: JSON

Heroku service

Gunicorn не дружит с Windows[10], но если хочется поэкспериментировать, то рекомендуется посмотреть в сторону PaaS платформ типа Heroku. Разместите приложение через gunicorn на PaaS платформе Heroku (бесплатных dino-hours должно хватить с головой, чтобы разместить один экспериментальный Web-сервис).

[10] В качестве альтернативы для Window можно посмотреть на waitress

Можете хвастаться друзьям и коллегам запущенным Web-сервисом, но просьба не выкладывать исходный код в открытый доступ, чтобы другие слушатели самостоятельно решали задачу. Инструкция по умолчанию использует Django, но с переводом команд Flask не должно возникнуть проблем:

https://devcenter.heroku.com/articles/getting-started-with-python

Graphite Scheme Design

Graphite можно запустить в Docker'е (см. инструкции по установке). Загрузите полученные метрики Web-сервиса Wikipedia Search Engine в Graphite и изучите содержимое whisper-файла с помощью whisper-info.py и whisper-fetch.py (см. примеры команд с занятия).

Продумайте архитектуру хранения метрик (релевантные вопросы):

1. Частота производства данных;
2. Минимально-достаточная точность данных;
3. Максимальный период времени для оперативных данных;
4. Наименьшая точность, которая имеет смысл;
5. Как далеко смотреть в прошлое.

Создайте нужную схему данных и проверьте ее поведение при загрузке метрик.