

Задание

Анализатор логов

Жил-был чудный веб-интерфейса и все у него было хорошо: в него ходили пользователи, что-то там кликали, переходили по ссылкам, получали результат. Но со временем некоторые его странички стали тупить и долго грузиться. Менеджеры часто жалуются, мол “вот тут список долго грузился” или “интерфейсик тупит, поиск не работает”. Но так трудно отделить те случаи, где проблемы на их стороне, а где действительно виноват веб-сервис. В логи reverse proxy перед интерфейсом (`nginx`) добавили время запроса (`$request_time` в `nginx http://nginx.org/en/docs/http/ngx_http_log_module.html#log_format`). Теперь можно распарсить логи и провести первичный анализ, выявив подозрительные URL’ы.

Про логи:

- сэмпл лога: `nginx-access-ui.log-20170630.gz`
- шаблон названия логов `nginx-access-ui.log-YYYYMMDD.gz`
- так вышло, что логи могут быть и plain (т.е. “сырые”, без сжатия) и gzip
- лог ротируется раз в день
- опять же, так вышло, что логи интерфейса лежат в папке с логами других сервисов

Про отчет:

- `count` - сколько раз встречается URL, абсолютное значение
- `count_perc` - сколько раз встречается URL, в процентах относительно общего числа запросов
- `time_sum` - суммарный `$request_time` для данного URL’а, абсолютное значение
- `time_perc` - суммарный `$request_time` для данного URL’а, в процентах относительно общего `$request_time` всех запросов
- `time_avg` - средний `$request_time` для данного URL’а
- `time_max` - максимальный `$request_time` для данного URL’а
- `time_med` - медиана `$request_time` для данного URL’а

Задание: реализовать анализатор логов `log_analyzer.py`.

Основная функциональность:

1. Скрипт обрабатывает при запуске последний (со самой свежей датой в имени, не по `mtime` файла!) лог в `LOG_DIR`, в результате работы должен получится

отчет как в report-2017.06.30.html (для корректной работы нужно будет найти и принести себе на диск jquery.tablesorter.min.js). То есть скрипт читает лог, парсит нужные поля, считает необходимую статистику по url'ам и рендерит шаблон report.html (в шаблоне нужно только подставить \$table_json) в report-YYYY.MM.DD.html, где дата в названии соответствует дате обработанного файла логов . Ситуация, что логов на обработку нет возможна, это не должно являться ошибкой.

2. Готовые отчеты лежат в REPORT_DIR. В отчет попадает REPORT_SIZE URL'ов с наибольшим суммарным временем обработки (time_sum).
3. Скрипту должно быть возможно указать считать конфиг из другого файла, передав его путь через --config. У пути конфига должно быть дефолтное значение. Если файл не существует или не парсится, нужно выходить с ошибкой.
4. В переменной config находится конфиг по умолчанию. В конфиге, считанном из файла, могут быть переопределены переменные дефолтного конфига (некоторые, все или никакие, т.е. файл может быть пустой) и они имеют более высокий приоритет по сравнению с дефолтным конфигом. Таким образом, результирующий конфиг получается слиянием конфига из файла и дефолтного, с приоритетом конфига из файла. Ситуацию, когда конфига на диске не оказалось, нужно исключить.
5. Использовать конфиг как глобальную переменную запрещено, т.е. обращаться в своем функционале к нему так, как будто он глобальный - нельзя. Нужно передавать как аргумент.
6. Использовать сторонние библиотеки для реализации основного функционала запрещено, за исключение structlog
7. (задание со “звездочкой”) Если скрипт удачно обработал, то работу не переделывает при повторном запуске.

Мониторинг :

1. скрипт должен писать структурированные логи в JSON через structlog (плюс, рекомендуется познакомиться с утилитой jq). Допускается только использование уровней debug, info, error. Путь до логфайла указывается в конфиге, если не указан, лог должен писаться в stdout.
2. все возможные “неожиданные” ошибки должны попадать в лог вместе с трейсбеком. Имеются в виду ошибки непредусмотренные логикой работы, приводящие к остановке обработки и выходу: баги, нажатие ctrl+C,

кончилось место на диске и т.п.

3. (задание со “звездочкой”) должно быть предусмотрено оповещение о том, что большую часть анализируемого лога не удалось распарсить (например, потому что сменился формат логирования). Для этого нужно задаться относительным (волях/процентах) порогом ошибок парсинга и при его превышении писать в лог, затем выходит.

Тестирование:

на скрипт должны быть написаны тесты с использованием библиотеки `pytest`. Тестируемые кейсы и структура тестов определяется самостоятельно (без фанатизма, достаточно каких-нибудь разумных тестов, занятия про тестирование ждут нас впереди).

Оформление:

1. задать проекту типовую структуру (можно ориентироваться <https://realpython.com/python-application-layouts/> или свои любимые крупные проекты на `github`)
2. настроить `pre-commit` хуки, как минимум линтер, `black`, `isort`, `тому` и `poetry`
3. настроить CI со стандартными проверками: линтер, `black`, `isort`, `тому`, - и запуском тестов.
4. написать `README.md` с его кратким описанием и примерами запуска
5. настройку и управление зависимостями провести через `poetry` и `pyproject.toml` (ну и возможно `setup.cfg`)
6. подготовить `Makefile` для удобного запуска линтовки, тестов (и тестового покрытия) и самого приложения
7. (задание со “звездочкой”) подготовить `Dockerfile`, собранный образ должен уметь по подключенным через `volume` входным данным (директория с логами, конфиг) выдавать в этот же или другой `volume` отчет

Цель задания: получить (прокачать) навык создания полноценного проекта. То есть проекта, наполненного адекватным кодом, который удобно расширять и поддерживать, протестированного, пригодного для мониторинга, имеющего стандартную структуру, минимальное описание и настроенный CI. Совпадение всех чисел с приведенным примером отчета целью не является (лишь бы похожи были =)

Критерии успеха: задание **обязательно**, кроме частей со “звездочкой”, которые являются расширением основного функционала и выполняются по возможности. Критерием

успеха является работающий и оформленный согласно заданию код. Далее успешность определяется code review.

Распространенные проблемы:

- не стоит делать свои кастомные классы ошибок, это иногда (!) имеет смысл для библиотек, но не для задач подобного рода.
- ограничьтесь уровнями логирования DEBUG, INFO и ERROR: <https://dave.cheney.net/2015/11/05/lets-talk-about-logging>
- не выходите через `sys.exit` не из `main`. Это затрудняет тестирование и переиспользование кода.
- чтобы отрендерить шаблон не надо итерироваться по всем его строкам и искать место замены, можно воспользоваться, например, <https://docs.python.org/3/library/string.html#str.format>
- функцию, которая будет парсить лог желательно сделать генератором и передать в качестве аргумента функции создателю отчета.
- не забывайте про кодировки, когда читаете лог и пишите отчет.
- из функции, которая будет искать последний лог удобно возвращать `namedtuple/dataclass` с указанием пути до него, распаршенной через `datetime` даты из имени файла и расширением, например.
- распарщенная дата из имени логфайла пригодится, чтобы составить путь до отчета, это можно сделать “за один присест”, не нужно проходить по всем файлам и что-то искать.
- протестируйте функцию поиска лога, она не должна возвращать `.bz2` файлы и т.п. Этого можно добиться правильной регуляркой.
- найти самый свежий лог можно за один проход по файлам, без использования `glob`, сортировки и т.п.
- нужный открыватель лога (`open/gzip.open`) перед парсингом можно выбрать через тернарный оператор.
- проверка на превышение процента ошибок при парсинге выполняется один раз, в конце чтения файла, а не на каждую строчку/ошибку.
- коммитить толстые бинари (например `nginx-access-ui.log-20170630.gz`) в `git` не стоит, он для исходного кода, в основном

Полезные ссылки:

- <https://docs.python.org/3/library/statistics.html>
- <https://jesseduffield.com/Beginners-Guide-To-Abstraction/>

- <https://melevir.medium.com/короче-говоря-принцип-единой-ответственности-92840ac55baa>
- <https://melevir.medium.com/python-стиль-кода-1af07092836e>
- <https://melevir.medium.com/дизайн-функций-а63bc9588e11>
- [https://www.youtube.com/watch?v=o9pEzgHorH0 Stop Writing Classes](https://www.youtube.com/watch?v=o9pEzgHorH0)
- <https://leontrolski.github.io/mostly-pointless.html>
- <https://blog.guilatrova.dev/handling-exceptions-in-python-like-a-pro/>
- <https://mitelman.engineering/blog/python-best-practice/automating-python-best-practices-for-a-new-project/>
- <https://medium.com/@vanflymen/blazing-fast-ci-with-github-actions-poetry-black-and-pytest-9e74299dd4a5>

Deadline

Задание желательно сдать в течение недели. Код, отправленный на ревью в это время, рассматривается в первом приоритете. Нарушение делайна не карается. Пытаться сдать ДЗ можно до конца курсы. Код, отправленный с опозданием, когда по плану предполагается работа над более актуальным ДЗ, будет рассматриваться в более низком приоритете без гарантий по высокой скорости проверки.

Обратная связь

Студент коммитит все необходимое в свой github/gitlab репозитарий. Далее необходимо зайти в ЛК, найти занятие, ДЗ по которому выполнялось, нажать “Чат с преподавателем” и отправить ссылку. После этого ревью и общение на тему ДЗ будет происходить в рамках этого чата.