День 2

Отложенные задачи

Тестирование

Личный опыт

Давайте подумаем, в каком случае и для каких классов задач нам может потребоваться использовать отложенное выполнение

Сталкивались ли вы с такими задачами? Если нет, то давайте попробуем представить для чего нам это может быть нужно?

Основные цели

- Улучшить отзывчивость нашего интерфейса или api — то, что может быть выполнено без участия пользователя, должно быть выполнено отдельно
от сеанса взаимодействия с ним

- Как следствие, мы получаем уменьшение времени ответа и довольного пользователя

- Также мы можем, используя горизонтальное масштабирование, обработать “внезапную” нагрузку, просто увеличив количество воркеров

- Наконец, мы можем позволить себе выполнять тяжёлые или долгие задачи, никак не влияя на пользователя

Примеры решаемых задач

Построение агрегатов для дальнейшего анализа

Получение данных из различных источников

Обработка заказа и платежа

Всевозможные пользовательские отчёты

Обучение ML моделей

Любые нотификации — SMS, email, push notifications

Фоновая обработка загружаемых данных (картинок, документов ...)

Личный опыт

Возможно кто-то сталкивался с вариантами, которые позволяют решать обозначенные выше проблемы?

Если да, напишите какие библиотеки или сервисы для этого использовались

Варианты решений

- Любое самописное решение, для примера на основе RQ
- Celery, который является уже промышленным стандартом
- Любые другие разнообразные библиотеки - ARQ, Dramatiq, ...

Личный опыт

Давайте подумаем, чего нам хотелось бы от идеальной библиотеки, позволяющей выполнять отложенные или периодические задачи!

Напишите самые главные с вашей точки зрения возможности

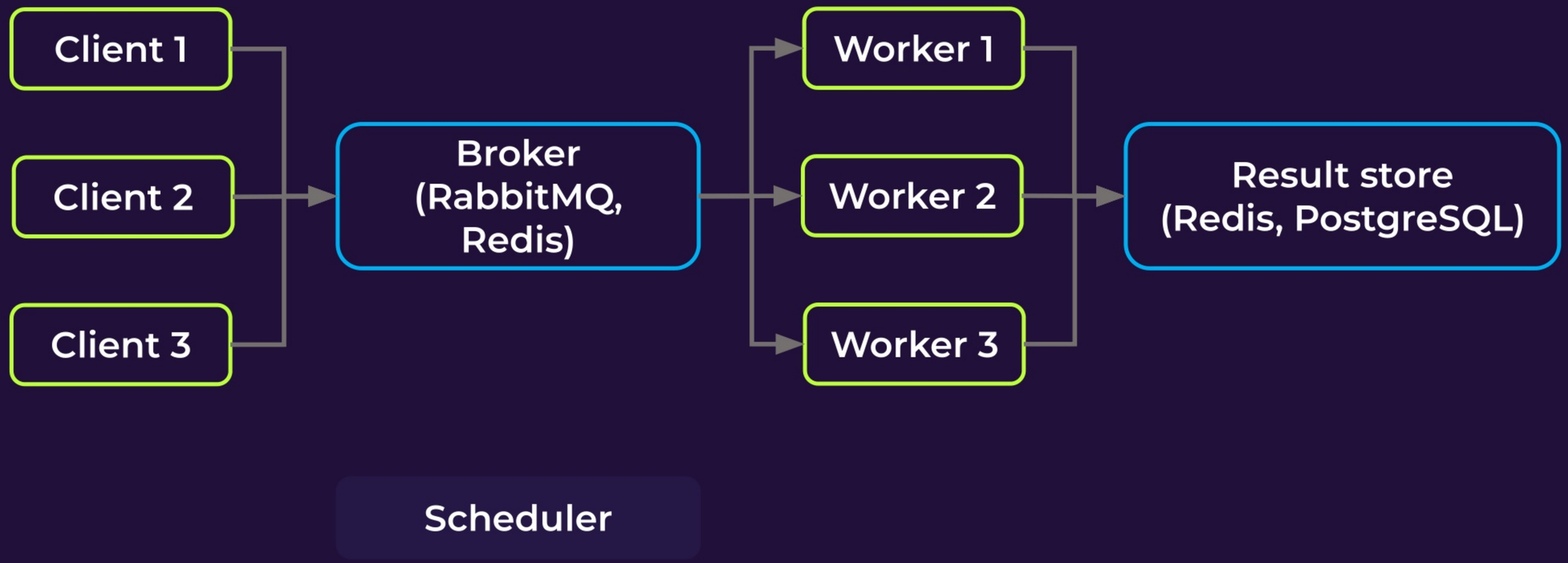
Celery: основные возможности

- Лёгкий старт — нужно написать минимум кода для простых задач
- Запуск задач по расписанию, а так же интервальное выполнение
- Гибкие политики по повтору неуспешных задач
- Наличие удобных инструментов для мониторинга состояния
- Гибкие настройки лимитов

Celery: дополнительные возможности

- Большой список очередей для реализации брокера задач
- Большой набор решений для сохранения результата обработки задачи
- Возможность сериализации задач и результата во множество форматов
- Можно гибко настроить запуск воркера (thread, solo, prefork)
- Есть механизм работы с несколькими очередями с разными приоритетами

Celery: схема взаимодействия



Личный опыт

Как мы обсуждали выше, важным моментом является возможность контролировать состояние celery, давайте подумаем, как это можно сделать и на какие моменты, метрики стоит обратить внимание

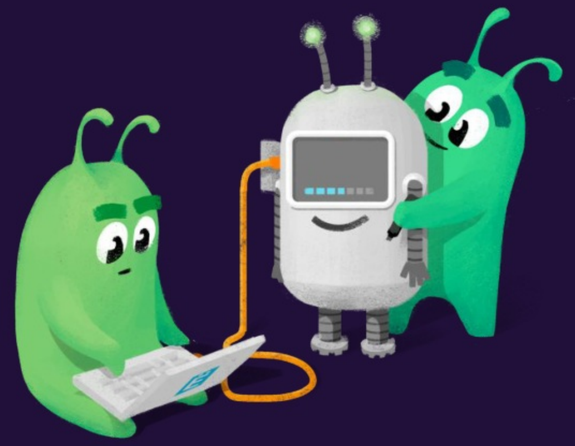
Ну и соответственно какой минимальный набор свойств должен быть у системы мониторинга celery?

Flower: основные возможности

- Просмотр детальной информации о запущенных задачах
- История выполненных задач
- Графики и статистика в разрезе очередь, класс задач
- Изменение лимитов очередей
- Остановка и перезагрузка воркеров
- Удаление запланированных задач и остановка запущенных

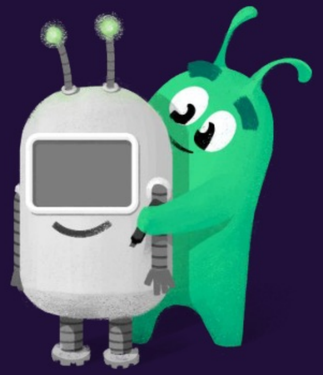
Практика: разбираемся вместе

- Пишем фоновую таску используя BackgroundTasks
- Устанавливаем Celery
- Устанавливаем Flower
- Добавляем простейшую таску
- Смотрим, как в интерфейсе Flower отображается информация о ней
- Разбираемся, какие данные можно получить с помощью Flower



Практика: реализуете сами

Добавляем фоновую таску - вывод текущего времени

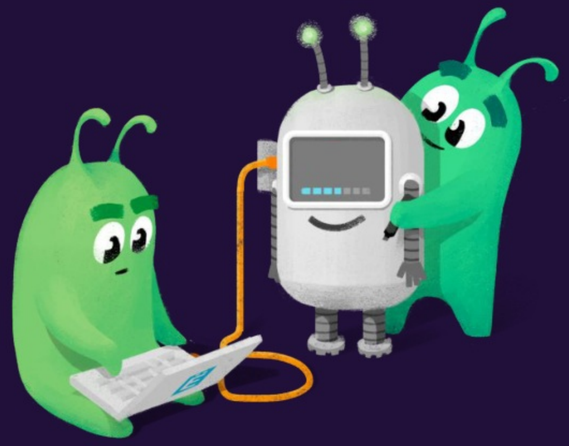


Добавляем фоновую таску - сложение двух timestamp

Переры
в

Практика: разбираемся вместе

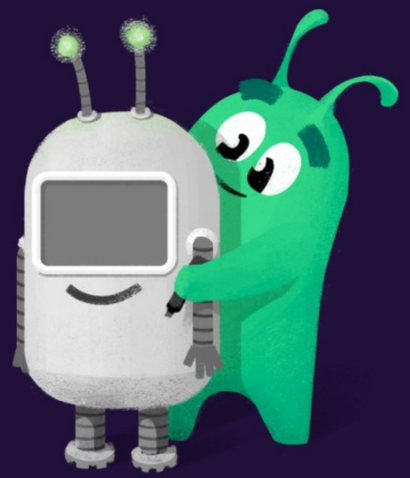
Добавляем интеграцию
с Celery в FastAPI



Пишем два api метода
которые позволяют создать
задачу
и проверить её статус

Практика: реализуете сами

Добавляем таску и endpoint для отправки email адресату



Отложенные таски: итоги

Итак, мы научились использовать Celery для написания отложенных и периодических задач и вызывать такие задачи из web приложения

Также мы узнали какими способами можно мониторить Celery и на что стоит
при этом обращать внимание

Большой перерыв

Личный опыт

Вопросы холиварные, но все же:

- Как вы сами относитесь к написанию тестов?
- Кто должен писать и поддерживать тесты?
- Какие плюсы и минусы от написания тестов?

Тесты: основные мотивы написания

- Проверить соответствия реальное и ожидаемое поведения программы
- Во многих случаях тесты являются хорошим подспорьем для понимания системы
- Наличие тестов облегчает процесс рефакторинга системы
- Тесты также облегчают добавление нового функционала
- Наличие нагрузочных тестов позволяет выявить узкие по производительности места

Личный опыт

А какие тесты вы пишете или пишут
у вас в команде?

Есть ли у вас какие-то жёсткие критерии к наличию тестов и покрытию ими кода?

Тесты: минимум который пишет разработчик

- Unit-тесты
- Функциональные тесты
- Помогает или пишет сам интеграционные тесты

Личный опыт

Какие вы использовали библиотеки или о каких библиотеках для тестирования вы слышали?

Тесты: основные библиотеки

Unittest

Pytest

Faker

Личный опыт

Существует множество вариантов разделения тестов на типы по различным свойствам

Напишите, какие виды тестов вы знаете и с какими чаще всего сталкиваетесь?

Тесты: основные виды

По объекту:

- Функциональное
- Тестирование производительности
- Тестирование безопасности
- ...

По знанию и доступности внутреннего устройства системы:

- Черный ящик
- Белый ящик
- Серый ящик

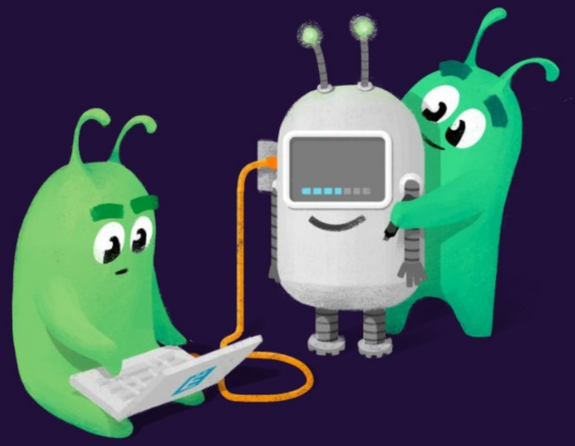
По степени автоматизации:

- Ручное
- Автоматизированное
- Полуавтоматизированное

Перер
ыв

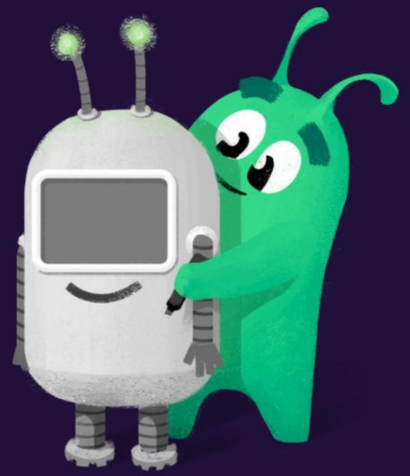
Практика: разбираемся вместе

- Настраиваем запуск тестов с помощью pytest
- Добавляем интеграционные тесты на основные endpoints
- Пишем unit-тесты на основную бизнес логику



Практика: реализуете сами

Добавляем тесты на 2-3 endpoints



Личный опыт

Как мы можем нагружать сервис
и какую информацию будем иметь по итогу?

Давайте подумаем для чего нам может пригодится нагрузочное тестирование?

На каких этапах разработки стоит использовать нагрузочное тестирование?

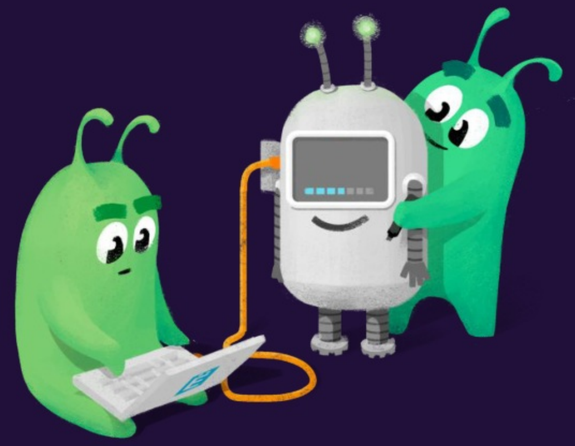
Нагрузочное тестирование

Существует множество всевозможных решений для проведения нагрузочного тестирования, вот только некоторые из них:

- Apache Jmeter
- Gatling
- Yandex Tank
- Locust

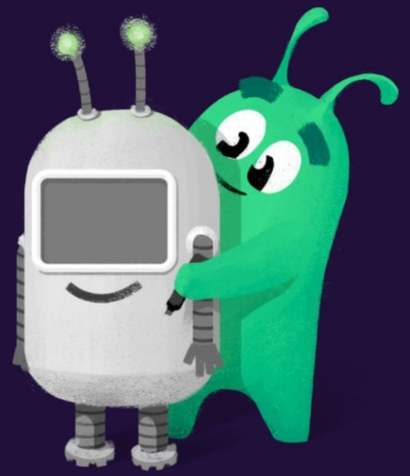
Практика: разбираемся вместе

Используя locust добавляем несколько нагрузочных тестов



Практика: реализуете сами

По подобию ранее написанных нагрузочных тестов реализуем тест на любой endpoint



Личный опыт

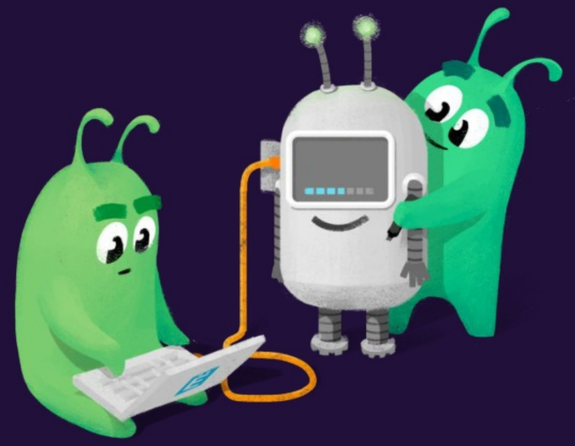
И, так как мы живём в реальном мире, мы должны уметь отлавливать такие ошибки и информировать о них для скорейшего исправления

Немного капитанства - в любом сервисе могут возникать непредвиденные ошибки, не бывает приложений без ошибок

Знаете ли вы инструменты для мониторинга таких ошибок, и если да, то о каких?

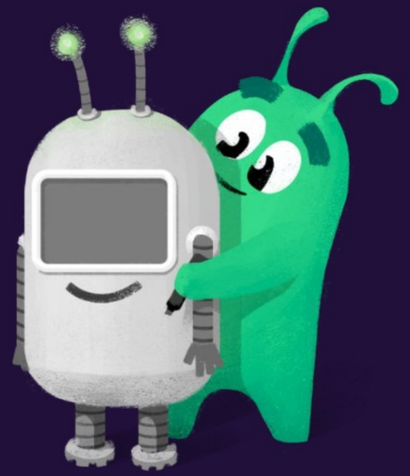
Практика: разбираемся вместе

- Поднимаем локально Sentry
- Регистрируем тестовый проект в Sentry
- Добавляем интеграцию с Sentry в написанное приложение
- Проверяем корректность работы связки



Практика: реализуете сами

Добавляем в любой метод выброс исключения по любому условию, например, ограничение на размер картинки или имя пользователя



Тесты: итоги

Мы рассмотрели основные виды тестирования и научились писать функциональные и unit тесты с помощью pytest

Также мы разобрались с написанием нагрузочных тестов с помощью locust

И в завершении сделали интеграцию с Sentry, которая позволит отлавливать непредвиденные ошибки

Вопросы?