С чем нужно быть знакомым, приступая к выполнению тестового задания:

1. python https://www.python.org/
2. Протокол HTTP https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
3. Asyncio
   3.1. Официальная документация asyncio https://docs.python.org/3/library/asyncio.html
   3.2. Хорошая статья для уже знакомых с python об asyncio с примерами: https://yeray.dev/python/asyncio/asyncio-for-the-working-python-developer
4. Используемый у нас веб-фреймворк “FastAPI”
   4.1. Репозиторий на github: https://github.com/fastapi/fastapi. Там можно найти документацию и примеры.
   4.2. Пошаговый tutorial о том, как написать простое приложение на FastAPI https://fastapi.tiangolo.com/tutorial/first-steps/#interactive-api-docs
5. Асинхронный “драйвер” для работы с postgresql в asyncio https://magicstack.github.io/asyncpg/current/
6. docker-compose. оркестрация сервисов, создание песочниц
   6.1. официальная документация https://docs.docker.com/compose/gettingstarted/
   6.2. пример докеризации flask и mysql https://stavshamir.github.io/python/dockerizing-a-flask-mysql-app-with-docker-compose/

Тестовое задание “Сервис-балансировщик видео-трафика”

Нужно написать максимально _простой_ сервис-балансировщик пользовательских запросов, который должен отправлять пользователя с помощью 301-го HTTP редиректа смотреть фильм либо на корневые сервера, либо отправлять его в CDN по определённым правилам.

При этом сервис должен без проблем обрабатывать не менее 1000 запросов в секунду на среднем ноутбуке с core i5 7го поколения и 16GB RAM.

Требования к стеку технологий

Python 3.9 (или выше), aiohttp или FastAPI в роли веб-фреймворка, PostgreSQL в качестве хранилища конфигов.

Входящие данные

Ожидается, что сервис будет обрабатывать входящие запросы вида:
GET http://balancer-domain/?video=http://s1.origin-cluster/video/1488/xcg2djHckad.m3u8

Где:

1. "balancer-domain " - хостнейм сервиса балансировки,
2. http://s1.origin-cluster/video/1488/xcg2djHckad.m3u8 - URL видео-файла на сервере оригиналов s1, который нужно отдать пользователю либо через CDN, либо напрямую с севера оригиналов.

Алгоритм обработки запросов

1. Каждый N-ый запрос, формата описанного выше, отправляем в оригинальный урл(query arg video) 301 редиректом. Конфигурация соотношения редиректов CDN:origin-сервера хранится в настройках окружения или в postgresql.
2. Остальные отправляем на http://$CDN_HOST/s1/video/1488/xcg2djHckad.m3u8, где:

- s1 - сервер в кластере оригиналов
- $CDN_HOST - настраиваемый base-url сервиса CDN, хранится в настройках окружения или в postgresql.

В сухом остатке

1. Настройки CDN_HOST, распределения CDN:origin-servers хранятся в postgresql. Хотелось бы иметь возможность изменять настройки через REST-подобное HTTP API. Схема хранения в БД – как угодно разработчику.
2. Балансировщик принимает запросы вида: 'http://balancer-domain/?video=http://s1.origin-cluster/video/1488/xcg2djHckad.m3u8'
3. http://s1.origin-cluster/video/1488/xcg2djHckad.m3u8 - url ведущий на сервер оригиналов, где "s1" - поддомен, указывающий на сервер s1 в кластере файловых серверов, в кластере по аналогии с s1 есть другие сервера, s2,s3,s4,...sN
4. video/1488/xcg2djHckad.m3u8 - location ведущий на файл с видео-плейлистом
5. 'http://{app.config.CDN_HOST}/s1/video/1488/xcg2djHckad.m3u8' - адрес, ведущий на CDN сервер, который кэширует ответы файловых серверов, описанных в п.2. Часть URL'а "s1" как раз указывает с какого сервера кэшировать данные, там может быть указан s2,s3,s4 и т.д.
6. Т.е. если мы решили отправить запрос из в CDN, то отправляем на адрес f'http://{app.config.CDN_HOST}/s1/video/1488/xcg2djHckad.m3u8', а если решили отправить напрямую на файл-сервер, минуя CDN, то на 'http://s1.origin-cluster/video/1488/xcg2djHckad.m3u8'

Что ожидаем от кандидата

0. Вопросов, если что-то непонятно в задании
1. Репозиторий в github или bitbucket с написанным кодом
2. docker-compose.yml файл для запуска этого проекта
3. README.md с описанием того как запустить сервис

О чём хотелось бы пообщаться

В таком виде сервис не совсем пригоден для продакшена, но является хорошей отправной точкой для размышлений. В свете этого было бы интересно услышать идеи о том, как этот сервис можно(и нужно) доработать, чего в нём _очевидно_ не хватает.