

Университет ИТМО

**Облачные системы и услуги**

**Лабораторная работа №2\***

Выполнили: Данилов Владислав Андреевич K34201**,**

Ледванов Вадим Дмитриевич K34201,

Смирнов Кирилл Андреевич K34201

**Преподаватель:** Мигулаева Татьяна Алексеевна

Санкт-Петербург

2024

**Цель работы:**

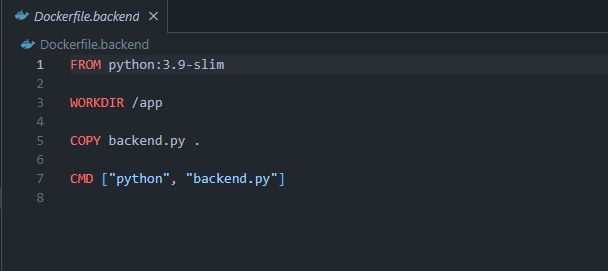
Изучить принципы написания Docker compose файлов и провести сравнительный анализ некорректных и оптимальных практик его разработки. Проанализировать неэффективные практики, выделить их недостатки и на основе этого сформулировать выводы, способствующие повышению эффективности работы с группами контейнеров, поднимаемых через compose.

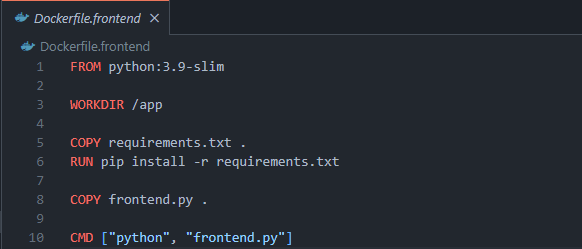
**Ход работы:**

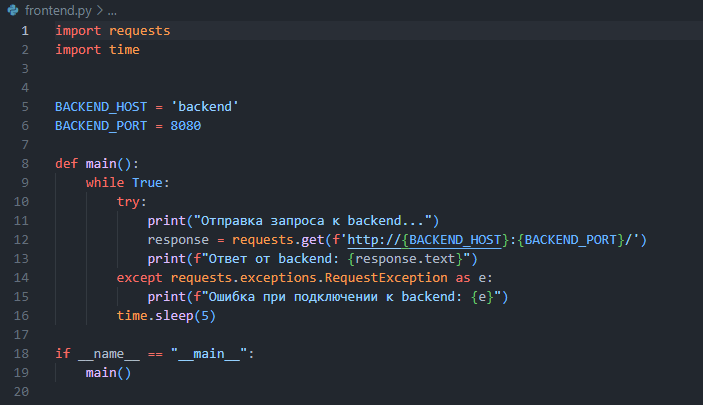
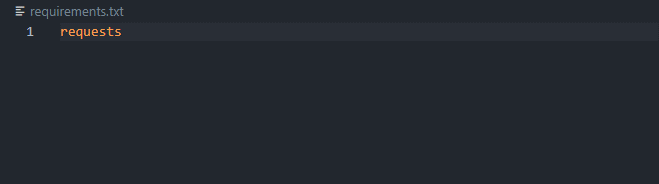
**Задание 1**.

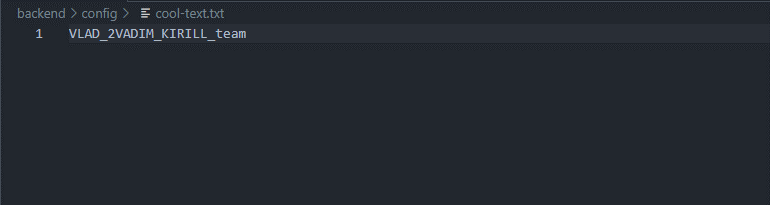
В работе было решено реализовать следующую задумку:

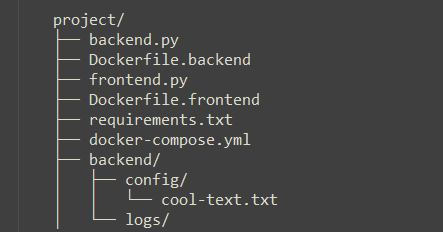
* Взять из предыдущей лабораторной работы наработки по backend-части простого клиент-серверного приложения, далее написать простую frontend-часть на Python.
* После этого производить запуск всего этого дела в отдельных контейнерах через *Docker Compose*. Основная часть реализации – использование **сетевых алиасов**, чтобы позволить контейнерам находить друг друга по имени.

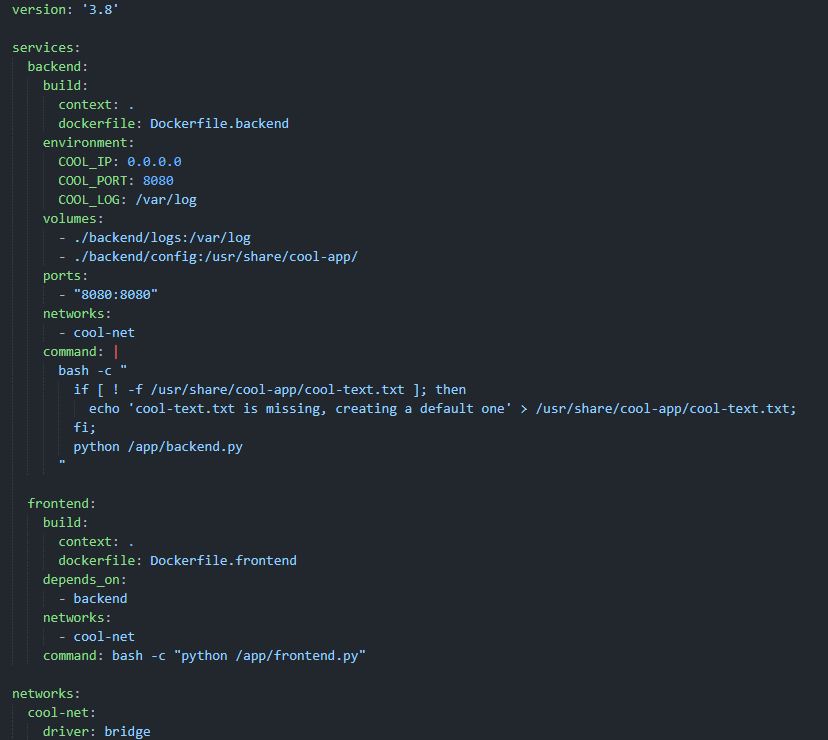
Ниже представлен полный код каждого файла проекта:

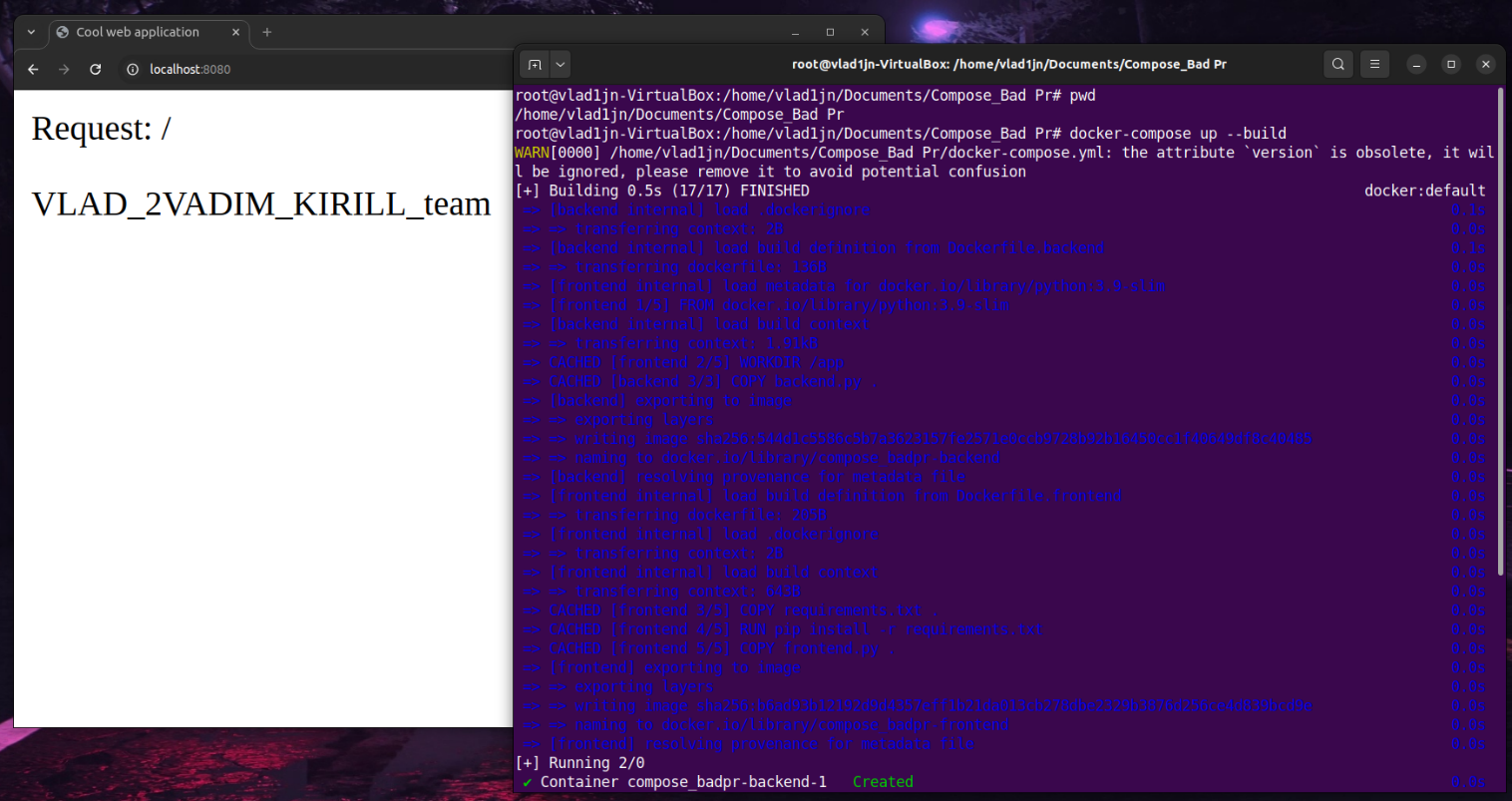


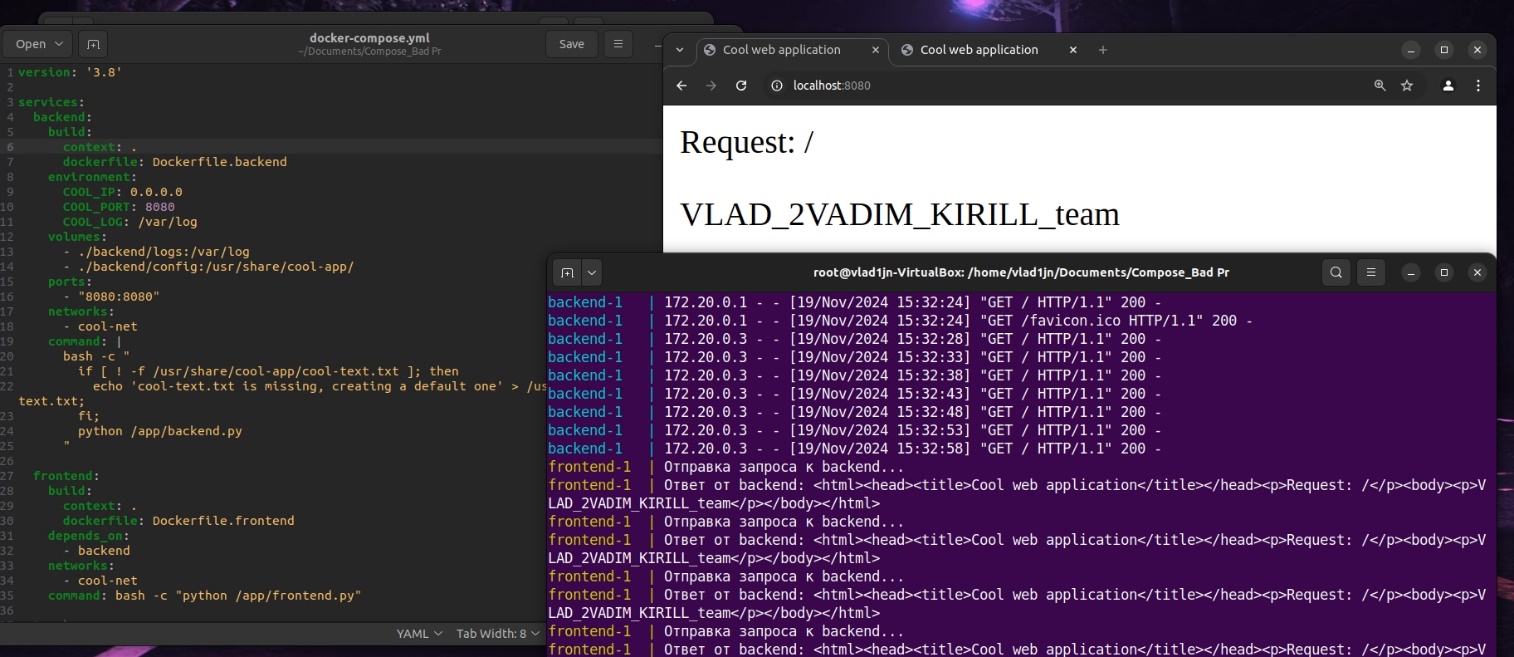


Код, который будет back отправлять в ответ вынесен в отдельный файл:

Структура проекта получается следующая:

**Файл Docker compose с bad practices:**

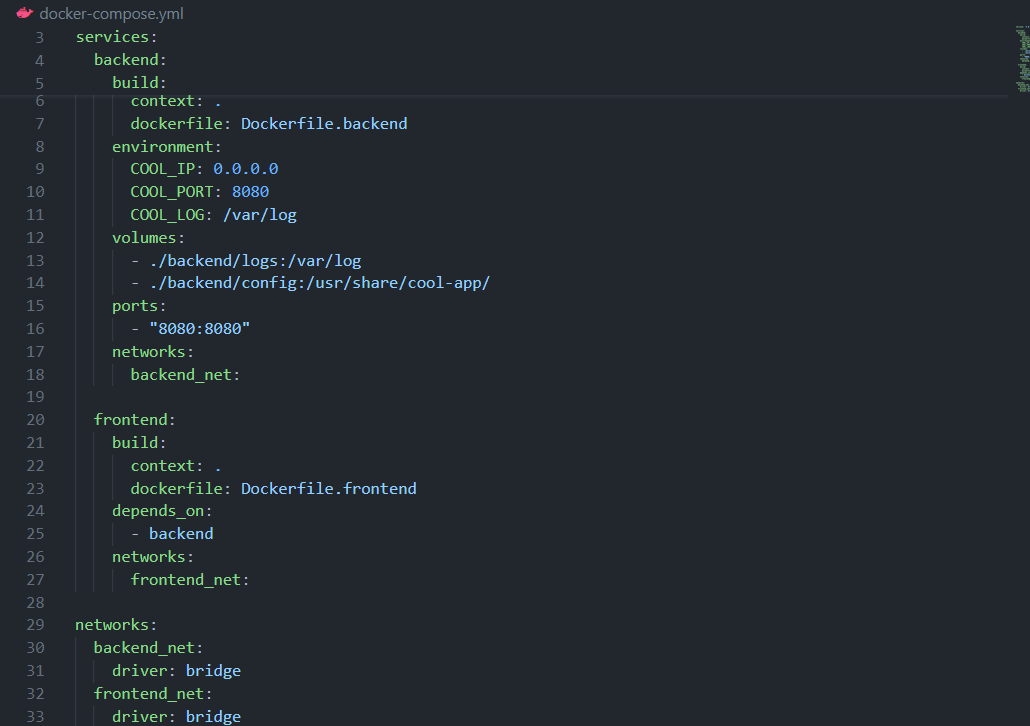
Запуск данной системы контейнеров через Docker compose инициируется следующим образом:

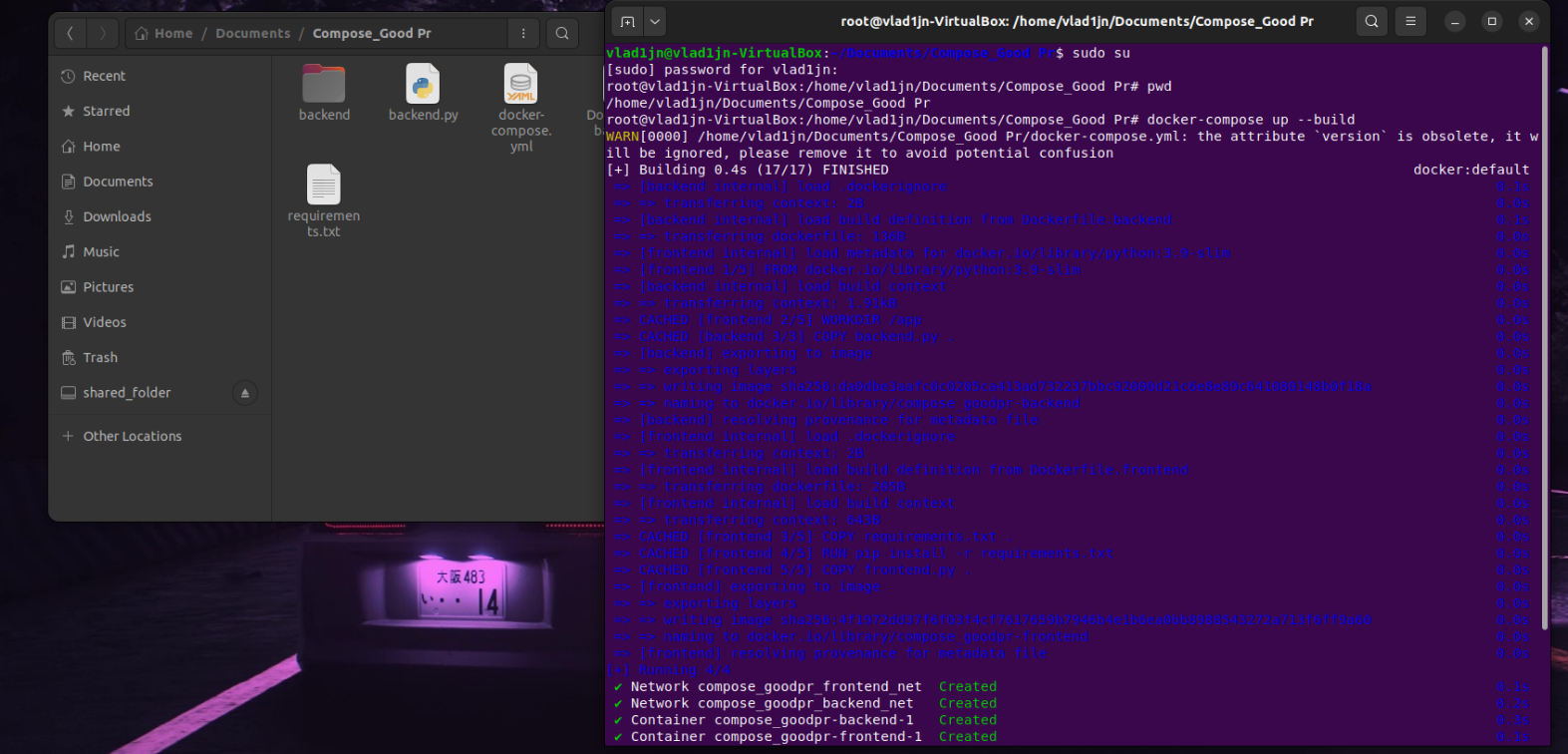
Далее вводя различные запросы и, манипулируя запросами к back, cookie-файлами и вкладками браузера (подключаясь с другого браузера), спустя время вынуждаем контейнеры общаться между собой. Это продемонстрировано ниже:

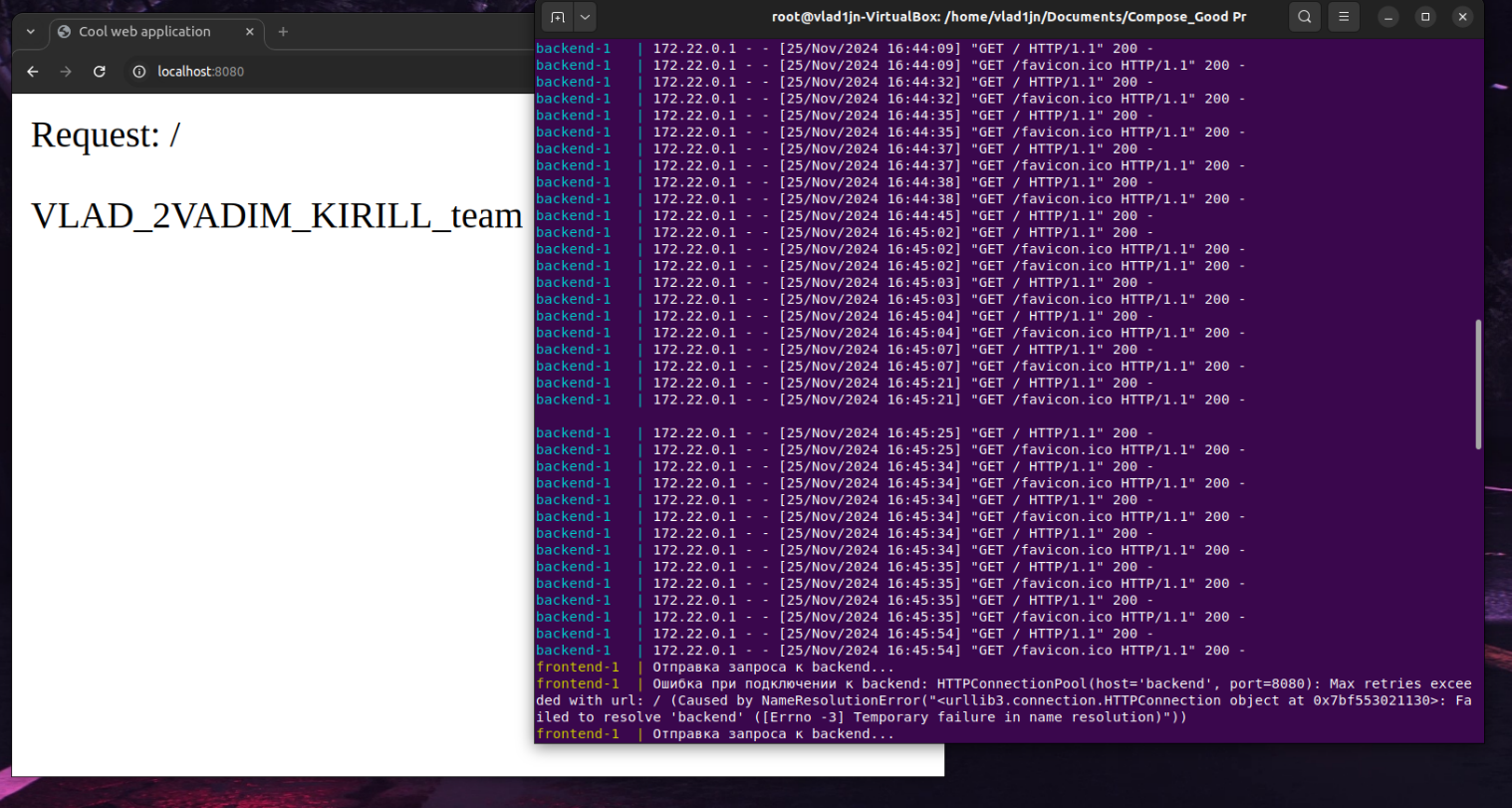
Таким образом, реализована система в которой контейнеры front и back Общаются через подсеть ‘cool-net’ на Localhost.

**Задание 2**.

Далее изменяем код в Docker compose. Остальной проект оставляем прежним.

**Полный код Файла Docker compose с good practices представлен ниже:**

Запуск инициируется следующим образом:

С помощью алиасов, Контейнеры взаимодействуют между собой:

**Задание 3**.

(Копия из Readme)

Плохие практики:

1. Выполнение сложной Bash-команды прямо в docker-compose (‘if [ ! -f /usr/share/cool-app/cool-text.txt ]’)

Команда с проверками и условиями, которая создаёт файл и запускает Python, делает код непонятным. Ещё в такой команде сложнее найти ошибки, если придется отлаживать. Такие команды не приняты в Compose.

1. Жесткая зависимость контейнера frontend от backend через ‘depends\_on’

Директива depends\_on не обеспечивает полную готовность бэкенда к работе до запуска фронтенда. Она лишь контролирует порядок запуска контейнеров. То есть система усложняется, без видимой необходимости

1. Отсутствие изоляции логов

Логи монтируются в общую директорию (‘./backend/logs’), что может привести к переполнению диска и конфликтам c Dockerfile’s

1. Использование одной подсети Localhost для всех контейн. (‘cool-net’)

Все сервисы работают в одной сети. Это может привести к потере данных и возможным проблемам. Например, при перехвате одного из Docker нарушителем, можно специально получить доступ к информации другого контейнера.

Как она (“bad practices”) была исправлена. И её влияние на результат:

1. Команда была вынесена специально вне проекта и параметризирована.

Это улучшило читаемость и обеспечило возможность повторного использования кода.

1. Реализованы 2 Localhost-подсети (‘networks:’)

То есть улучшена устойчивость системы: фронтенд продолжит корректно работать даже при временной недоступности бэкенда.

1. Логи перенесены в отдельную директорию (‘Compose\_Good Pr\backend\logs’)

Так еще и удалось улучшить управление логами

1. Bad Practice пункт 2 включал и это. Было исправлено: для каждого сервиса создана ’backend\_net’ и ‘frontend\_net’, соответственно.

Это изолировало их друг от друга еще и на уровне сетей.

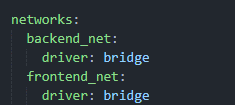
**Задание 4**. **Настройка изоляции между контейнерами**

Чтобы сервисы не "видели" друг друга по сети, были настроены два отдельных сетевых пространства (пункт 4 пред.).

Backend и frontend работают в разных местах.

Чтобы Docker’s могли общаться, нужен специальный путь – маппинг портов. В данном случае, frontend может связаться с backend только через “ports- "8080:8080"”.

Таким образом, в хорошем примере:

Каждая из 2 сетей использует мостовой драйвер (‘bridge’), который изолирует трафик между.

Контейнеры внутри одной сети могут общаться друг с другом, но контейнеры из разных сетей не видят друг друга, если явно не замаплены порты.

**Вывод:**

Изучены принципы написания Docker Compose файлов и проанализированы хорошие и плохие практики. Исправление последних, включало изоляцию сервисов, вынос сложных команд из Compose в арг., устранение прямой зависимости и централизованное управление переменными. В целом, это повысило стабильность и управляемость проекта.

Соблюдение хороших практик - чрезвычайно важно для обеспечения надежной и гибкой работы контейнерных приложений.