Задание #1

- Зайти на Gitlab кафедры ВС. Далее будет подразумеваться именно этот Gitlab – https://git.csc.sibsutis.ru
- Создать репозиторий. Дальнейшая разработка будет вестись в этом репозитории.
- Сконфигурировать клиент git для использования имени и адреса email из учетной записи gitlab.
- Написать простое приложение, которое запускает http-сервер и отдает контент. Контент может быть любым – строка, текст, случайное число и тп. Язык программирования – любой.
- Написать Dockerfile для данного приложения. Результат сборки – docker image, при запуске docker-контейнера из которого, в контейнере запускается разработанное приложение. Файл разместить в репозитории проекта.

Задание #2

- Доработать Dockerfile – Для компилируемых языков сделать multistage сборку.
    a. Этап 1 – компиляция/сборка. Используется образ с наличием компилятора (Например `FROM gcc`)
    b. Этап 2 – копирование артефакта компиляции и запуск приложения. Используется образ ОС (Например `FROM alpine`)
    c. Проанализировать как изменится размер итогового образа. В случае использования интерпретируемых ЯП, например, python – создать в отдельной папке `Dockerfile` и использовать программу «Hello World» на компилируемом языке.
- Разработать конфигурацию Gitlab CI со следующим функционалом
    a. Сборка образа (Build, tag)
    b. Загрузка образа в Gitlab Registry (push)
    c. Удаление образа с Gitlab Runner-а

  Допускается опираться на template ci, предлагаемый gitlab.
- Запуск CI должен выполняться автоматически при установке тега для коммита (`git tag`). Так же для любого коммита должна быть возможность запуска «по кнопке» (manual). Тегом (docker tag) получившегося docker-образа является хэш коммита или git tag.
- Добавить в проект `.dockerignore` файл, включив в него список файлов, не требуемых для сборки проекта (Синтаксис аналогичен файлу `.gitignore`). Проанализировать как изменится размер итогового образа и время сборки (Для измерения времени и размера можно создать временный большой файл и добавить/убрать его из `.dockerignore`).
- **Временный файл добавлять в индекс репозитория и коммитить не нужно.**
- **При наличии программы Hello World собирать образ этой программы средствами CI не требуется. Сборка только основного проекта**

Задание #3

- Разработать `docker-compose.yml` файл со следующим функционалом:
  a. Запуск образа приложения, собранного через CI в задании #3
  b. Запуск готового docker-приложения с веб-сервером (например – traefik/whoami)
  c. Запуск основного прокси-сервера по варианту. Прокси-сервер должен реализовать отдачу контента двух приложений при обращении по домену/IP адресу на порт 80 хоста. Домен – любой. При обращении на `/app` запрос должен проксироваться приложению из пункта b. На остальные запросы должно отвечать приложение из пункта a. Варианты основных прокси-серверов:
     1. Nginx
     2. Apache
     3. Traefik
- `docker-compose.yml` файл, дополнительные файлы конфигурации разместить так же в репозитории. Файлы конфигурации поместить в отдельной директории.
- Сервисы, описанные в `docker-compose.yml` не должны иметь директиву build. Запуск производится из уже собранных/готовых образов задания #3. Для того, чтобы использовать образы из registry необходимо выполнить команду

```
docker login registry.csc.sibsutis.ru
```

Задание #4

- Доработать `Dockerfile` таким образом, чтобы приложение (Разработанное в задании #1) запускалось от имени непривилегированного пользователя (Не root). Необходимо в процессе сборки создать пользователя и:
    a. Используя entrypoint-скрипт, инструкцию `ENTRYPOINT` и `gosu` (`https://github.com/tianon/gosu`) при старте контейнера переключиться в контекст непривилегированного пользователя.
    b. Используя инструкцию USER установить для запуска созданного пользователя.
- **Результатом работы является две версии Dockerfile – с использованием entrypoint-скрипта и с использованием инструкции USER.**
- **Допускается оформить результат двумя отдельными файлами (CI для них перенастраивать не требуется).**
  Или допускается применить один из подходов в основной проект.
- Показать различия в процессе сборки и запуске контейнера при использовании двух указанных методов

Задание #5

Добавить в Docker-compose экземпляр базы данных по варианту. Допускается использование других баз данных (Далее – БД)

a. Mariadb
b. Postgresql
c. Mongodb

- Доработать приложение для работы с базой данных.
  a. Необходимо разработать модель данных (Например, `User` с полями `id`, `name`, `email`)
  b. При запуске контейнера БД должны создаваться необходимые для этой модели таблицы в базе данных.
  c. Реализовать в приложении набор следующих `endpoint`-ов (Хотя бы 2, на примере `/users`)
     1. `GET /users` – вывести список всех пользователей из БД (можно реализовать с поддержкой параметров start-end или иным вариантом пагинации: `/users?_start=0&_end=5`)
     2. `PUT /users` – создать пользователя с переданными параметрами
     3. `GET /users/{:id}` – вывести данные пользователя с указанным id
     4. `DELETE /users/{:id}` – удалить пользователя с указанным id
  d. Модель данных, передаваемых для `put`, `post`, `delete` оформляется в `json`, возвращаемые данные также в `json`.
  e. Продемонстрировать взаимодействие с БД
- Параметры подключения к базе данных необходимо передавать приложению при запуске через ключи командной строки или переменные окружения (Environment)
- Дополнительные файлы конфигурации разместить в отдельной директории.