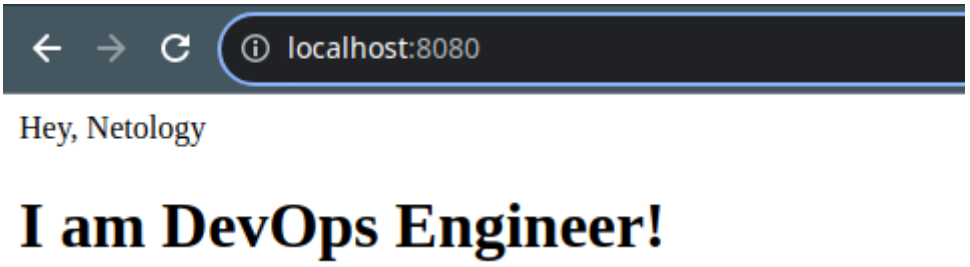
Домашнее задание к занятию "3. Введение. Экосистема. Архитектура. Жизненный цикл Docker контейнера"

Задача 1
Сценарий выполнения задачи:

создайте свой репозиторий на https://hub.docker.com;
выберете любой образ, который содержит веб-сервер Nginx;
создайте свой fork образа;
реализуйте функциональность: запуск веб-сервера в фоне с индекс-страницей, содержащей HTML-код ниже:
<html>
<head>
Hey, Netology
</head>
<body>
<h1>I'm DevOps Engineer!</h1>
</body>
</html>
Опубликуйте созданный форк в своем репозитории и предоставьте ответ в виде ссылки на https://hub.docker.com/username_repo.

Ссылка на решение: https://hub.docker.com/repository/docker/290381/alexg36
команда: docker pull 290381/alexg36:1.1
в браузере открыть http://localhost:8080/

скриншот



Задача 2

Посмотрите на сценарий ниже и ответьте на вопрос: "Подходит ли в этом сценарии использование Docker контейнеров или лучше подойдет виртуальная машина, физическая машина? Может быть возможны разные варианты?"
Детально опишите и обоснуйте свой выбор.
Сценарий:
Высоконагруженное монолитное java веб-приложение;
Nodejs веб-приложение;
Мобильное приложение с версиями для Android и iOS;
Шина данных на базе Apache Kafka;
Elasticsearch кластер для реализации логирования продуктивного веб-приложения - три ноды elasticsearch, два logstash и две ноды kibana;
Мониторинг-стек на базе Prometheus и Grafana;
MongoDB, как основное хранилище данных для java-приложения;
Gitlab сервер для реализации CI/CD процессов и приватный (закрытый) Docker Registry.

Высоконагруженное монолитное java веб-приложение - физическая или виртуальная машина с выделенными ресурсами, из-за монолитности не будет проблем с разворачиванием на разных машинах.
Nodejs веб-приложение - docker, для более простого воспроизведения зависимостей в рабочих средах.
Мобильное приложение с версиями для Android и iOS - виртуальная машина или физическое оборудование, контейнеризация не подходит для работы с UI.
Шина данных на базе Apache Kafka - docker, есть готовые образы для apache kafka, изолированность приложений, а также легкий откат на стабильные версии.
Elasticsearch кластер для реализации логирования продуктивного веб-приложения - три ноды elasticsearch, два logstash и две ноды kibana - в документации для размещения требуется использование выделенных хостов (виртуальных машин или контейнеров) некоторые функции Elasticsearch подразумевают что это единственное ресурсоемкое приложение на хосте.
Мониторинг-стек на базе Prometheus и Grafana - можно использовать контейнер или виртуальную машину для облегчения развертывания и масштабирования. Есть готовые образы, приложения не хранят данные, что удобно при контейнеризации.
MongoDB, как основное хранилище данных для java-приложения - можно использовать все три варианта, при этом для Docker потребуется смонтировать внешний том для хранения данных.
Gitlab сервер для реализации CI/CD процессов и приватный (закрытый) Docker Registry - могут быть применены все варианты, в зависимости от наличия соответствующих ресурсов. Но для большей изолированности лучше использовать docker. Gitlab север исходя из документации может быть размещён всеми тремя способами.

Задача 3

Запустите первый контейнер из образа centos с любым тэгом в фоновом режиме, подключив папку /data из текущей рабочей директории на хостовой машине в /data контейнера;
Запустите второй контейнер из образа debian в фоновом режиме, подключив папку /data из текущей рабочей директории на хостовой машине в /data контейнера;
Подключитесь к первому контейнеру с помощью docker exec и создайте текстовый файл любого содержания в /data;
Добавьте еще один файл в папку /data на хостовой машине;
Подключитесь во второй контейнер и отобразите листинг и содержание файлов в /data контейнера.

Скачаем образы centos и debian
вывод с терминала:

```
   ~
~  docker pull centos
✔
Using default tag: latest
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest:
sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
docker.io/library/centos:latest
   ~  docker pull debian
✔  19s
Using default tag: latest
latest: Pulling from library/debian
17c9e6141fdb: Pull complete
Digest:
sha256:bfe6615d017d1eebe19f349669de58cda36c668ef916e618be78071513c690e5
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
   ~
✔  15s
```

Запускаем первый контейнер из образа centos в фоновом режиме, подключив папку /data из текущей рабочей директории на хостовой машине в /data контейнера.
вывод с терминала:

```
   ~  docker run -v /data:/data --name centos-container -d -t centos
✔  15s
9fefc823667b89e6ebde3bc5764c11387db611a0f3ff775bb8d5829090f4409f
   ~
```

Запускаем второй контейнер из образа debian в фоновом режиме, подключив папку /data из текущей рабочей директории на хостовой машине в /data контейнера
вывод с терминала:

```
docker run -v /data:/data --name debian-container -d -t debian
3a868e48268c960e0418a186f36fe3d6cf84d578d25bc99495c335f2c4bc181b
```

Смотрим запущенные контейнеры:

~ docker ps

✔

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

3a868e48268c debian "bash" 2 minutes ago Up 2 minutes debian-container

9fefc823667b centos "/bin/bash" 5 minutes ago Up 5 minutes centos-container

~

✔

Подключаемся к первому контейнеру с помощью docker exec и создаем текстовый файл в /data

вывод с терминала:

~ docker exec centos-container /bin/bash -c "echo test>/data/readme.txt"

✔

~

Добавляем еще один файл в папку /data на хостовой машине

вывод с терминала:

~ sudo touch /data/readme_host.txt

✔

[sudo] пароль для alex:

~

Подключаемся во второй контейнер и отображаем листинг и содержание файлов в /data контейнера

вывод с терминала:

~ docker exec -it debian-container /bin/bash

✔ 18s

root@3a868e48268c:/# ls

bin boot data dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var

root@3a868e48268c:/# cd /data

root@3a868e48268c:/data# ls -l

total 4

-rw-r--r-- 1 root root 5 Nov 9 20:19 readme.txt

-rw-r--r-- 1 root root 0 Nov 9 20:22 readme_host.txt

root@3a868e48268c:/data#