Министерство цифрового развития, связи и
массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Отчёт
по лабораторной работе №2
по дисциплине «**Архитектура Вычислительных Систем**»

Выполнил:
студент гр. ИС-142
«__» декабря 2023 г. _____ /Наумов А.А./

Проверил:
старший преподаватель
кафедры ВС
«__» декабря 2023 г. _____ /Ревун А.Л./

Оценка « _____»

Новосибирск 2023

ЗАДАНИЕ

а) Разработать программу, принимающую аргументы командной строки и/или читает параметры из переменных среды окружения. Функционал программы на данном этапе не ограничивается. Например, на вход программы передается путь к файлу и его содержимое выводится в терминал. Программа не должна завершать свою работу до получения сигнала остановки.

б) Создать контейнер. Скомпилировать и запустить внутри контейнера программу, разработанную в п. а). Передать программе аргумент или переменную окружения при запуске контейнера.

в) Показать уровни изоляции: Файловая система, процессы, сеть.

ВЫПОЛНЕНИЕ РАБОТЫ

EXAMPLE

Dockerfile

```
FROM amytabb/docker_ubuntu16_essentials
ENV NAME VAR1
COPY run.sh /run.sh
COPY main.cpp /main.cpp
WORKDIR /
RUN g++ -o main main.cpp
CMD ["/bin/sh", "/run.sh"]
```

Main.cpp

```
#include <iostream>

int main(int argc, char **argv) {
  for(int i = 0; i < argc; ++i) {
    std::cout << "Argument " << i << ": " << argv[i] << '\n';
  }
  return 0;
}
```

Run.sh

```
#!/bin/sh
./main $VAR1
```

- `FROM amytabb/docker_ubuntu16_essentials`: Этот Dockerfile начинается с базового образа `amytabb/docker_ubuntu16_essentials`.

- `ENV NAME VAR1`: Определение переменной окружения `VAR1` с

именем `NAME`. Это означает, что переменная `VAR1` будет использоваться в контейнере с именем `NAME`.

- `COPY run.sh /run.sh` и `COPY main.cpp /main.cpp`: Копируют файлы `run.sh` и `main.cpp` из контекста сборки в корень файловой системы контейнера.

- `WORKDIR /`: Устанавливает рабочий каталог в корень файловой системы.

- `RUN g++ -o main main.cpp`: Компилирует исходный код `main.cpp` с помощью компилятора g++ и создает исполняемый файл `main` в корне файловой системы контейнера.

- `CMD ["/bin/sh", "/run.sh"]`: Устанавливает команду по умолчанию для выполнения при запуске контейнера. В данном случае, запускается скрипт `/run.sh` с использованием оболочки `/bin/sh`.

Простое C++ приложение, которое выводит аргументы командной строки.

Скрипт, который запускает исполняемый файл `main` с аргументом `$VAR1`. Значение переменной окружения `VAR1` будет передано в качестве аргумента приложению `main`.

Таким образом, этот код создает Docker-образ, включающий в себя компиляцию исходного кода C++, и запускает контейнер с приложением, которое выводит аргументы командной строки. Значение переменной окружения `VAR1` используется как аргумент при запуске контейнера.

Readme.txt

```
docker build -t example .
docker run -it -e VAR1='Hi!' example
```

NETWORK.SH

```
docker run -it --rm -d --name nginx nginx
```

Эта команда Docker используется для запуска контейнера с веб-сервером Nginx.
- `docker run`: Команда для запуска контейнера.
- `-it`: Опция для создания интерактивного терминала (stdin и stdout привязаны).
- `--rm`: Опция, которая автоматически удаляет контейнер после его завершения. Это полезно, чтобы избежать накопления неиспользуемых контейнеров.
- `-d`: Опция для запуска контейнера в фоновом режиме (detached mode).
- `--name nginx`: Опция для задания имени контейнера. В данном случае, контейнер будет назван "nginx".
- `nginx`: Имя образа, который будет использован для создания контейнера. В данном случае, это официальный образ Nginx из Docker Hub.

Таким образом, эта команда создает и запускает контейнер с веб-сервером Nginx в фоновом режиме, с интерактивным терминалом, и удаляет контейнер после его завершения.

PROCESSES:

Dockerfile

```
FROM ubuntu:latest
CMD ["top"]
```

Readme.txt

```
docker build -t processes .
docker run -it processes
```

1. `FROM ubuntu:latest`: Указывает базовый образ для создания нового образа. В данном случае используется официальный образ Ubuntu с тегом "latest" (последняя версия).
2. `CMD ["top"]`: Задает команду, которая будет выполнена при запуске контейнера. В данном случае, это команда "top", которая выводит список текущих процессов в системе.
Что касается изоляции процессов в Docker, Docker использует технологии контейнеризации, такие как cgroups (Control Groups) и namespaces, чтобы обеспечить изоляцию ресурсов и процессов между контейнерами и хостовой системой. Вот какие аспекты изоляции присутствуют в Docker:

1. Изоляция процессов (namespaces): Каждый контейнер имеет свой собственный namespace процессов, что означает, что процессы внутри контейнера видят только другие процессы в этом контейнере, а не в других контейнерах или на хостовой системе.
2. Изоляция ресурсов (cgroups): Cgroups позволяют ограничивать и управлять ресурсами, выделенными для каждого контейнера, такими как CPU, память, сеть и дисковое пространство. Это обеспечивает предсказуемое и контролируемое использование ресурсов.
3. Изоляция файловой системы: Каждый контейнер имеет свое собственное файловое пространство, что обеспечивает изоляцию файловых систем между контейнерами и хостовой системой.
4. Изоляция сети: Docker позволяет создавать сети для контейнеров, что обеспечивает изоляцию сетевых интерфейсов и портов между контейнерами и хостовой системой.

Таким образом, процессы внутри контейнера выполняются в изолированной среде, что делает их видимыми только внутри этого контейнера и не влияет на процессы в других контейнерах или на хостовой системе.

Filesystem:

Dockerfile

```
FROM amytabb/docker_ubuntu16_essentials
WORKDIR /
```

1. `FROM amytabb/docker_ubuntu16_essentials`: Указывает базовый образ для создания нового образа. Он наследует функциональность и настройки из указанного образа.
2. `WORKDIR /`: Задает текущий рабочий каталог для следующих инструкций Dockerfile. В данном случае, устанавливает его в корень файловой системы (root directory).
Файловая система контейнера изолирована от файловой системы хостовой системы. Каждый контейнер имеет свое собственное пространство файлов, что означает, что изменения внутри контейнера не влияют на файлы хостовой системы и наоборот. Это достигается с использованием технологии namespaces, которая обеспечивает изоляцию процессов и файловых систем. Каждый контейнер видит свою собственную файловую систему, и изменения, внесенные внутри контейнера, ограничиваются этим контейнером.

Readme.txt

```
docker build -t filesystem .
docker run -it filesystem
```