

Департамент образования города Москвы
Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»

Институт цифрового образования
Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

«Интеграция и развертывание программного обеспечения с помощью
контейнеров»

Лабораторная работа №2.1

Тема: «Создание Dockerfile и сборка образа»

Выполнила:

Студентка группы АДЭУ-211

Кравцова Алёна Евгеньевна

Руководитель:

Босенко Т.М

Москва

2024

Цель работы: научиться создавать Dockerfile и собирать образы Docker для приложений.

Задачи:

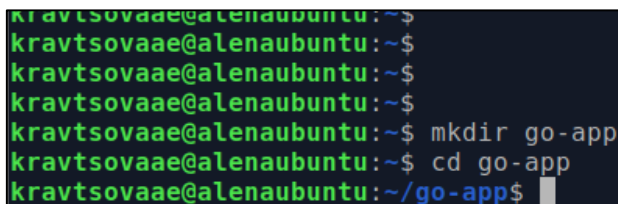
- Создать Dockerfile для указанного приложения;
- Собрать образ Docker с использованием созданного Dockerfile;
- Запустить контейнер из собранного образа и проверить его работоспособность;
- Выполнить индивидуальное задание.

Индивидуальное задание: Вариант 6. Создайте Dockerfile для приложения на Go, которое выводит "Hello, Go!" при запуске.

Шаги выполнения:

1) Создание проекта и подготовка файлов

Создадим новую директорию для проекта и перейдем в нее (Рис. 1).

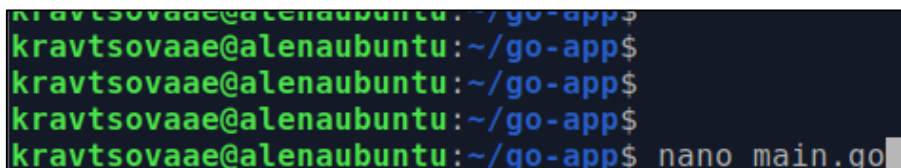


```
kravtsova@alenaubuntu:~$  
kravtsova@alenaubuntu:~$  
kravtsova@alenaubuntu:~$  
kravtsova@alenaubuntu:~$  
kravtsova@alenaubuntu:~$ mkdir go-app  
kravtsova@alenaubuntu:~$ cd go-app  
kravtsova@alenaubuntu:~/go-app$
```

Рис. 1 – Создание директории проекта

2) Создание файла main.go

Создадим файл main.go (Рис. 2) и добавим в него код, представленный на рисунке 3.



```
kravtsova@alenaubuntu:~/go-app$  
kravtsova@alenaubuntu:~/go-app$  
kravtsova@alenaubuntu:~/go-app$  
kravtsova@alenaubuntu:~/go-app$ nano main.go
```

Рис. 2 – Создание main.go



```
GNU nano 7.2 main.go *
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Hello, Go!")
}

func main() {
    http.HandleFunc("/", handler)
    fmt.Println("Server is running on port 8080...")
    http.ListenAndServe(":8080", nil)
}
```

Рис. 3 – Содержание main.go

```
package main

import (
    "fmt"
    "net/http"
)

func handler (w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Hello, Go!")
}

func main () {
    http.HandleFunc("/", handler)
    fmt.Println("Server is running on port 8080...")
    http.ListenAndServe(":8080", nil)
}
```

Этот код создает HTTP-сервер, который запускается на порту 8080 и при обращении к корневому URL (/) возвращает «Hello, Go!».

3) Создание Dockerfile

Создадим файл Dockerfile в той же директории (Рис. 4) и добавим в него код (Рис. 5).

```
kravtsovaee@alenaubuntu:~/go-app$  
kravtsovaee@alenaubuntu:~/go-app$  
kravtsovaee@alenaubuntu:~/go-app$ nano Dockerfile
```

Рис. 4 – Создание Dockerfile

```
GNU nano 7.2 Dockerfile *  
# Используем официальный образ Golang в качестве базового  
FROM golang:1.20-alpine  
  
# Устанавливаем рабочую директорию внутри контейнера  
WORKDIR /app  
  
# Копируем файлы проекта в контейнер  
COPY main.go .  
  
# Компилируем Go-приложение  
RUN go build -o app main.go  
  
# Указываем, что контейнер будет слушать 8080 порт  
EXPOSE 8080  
  
# Команда для запуска приложения  
CMD ["/app"]
```

Рис. 5 – Содержание dockerfile

```
# Используем официальный образ Golang в качестве базового  
FROM golang:1.20-alpine  
  
# Устанавливаем рабочую директорию внутри контейнера  
WORKDIR /app  
  
# Копируем файлы проекта в контейнер  
COPY main.go .  
  
# Компилируем Go-приложение  
RUN go build -o app main.go  
  
# Указываем, что контейнер будет слушать 8080 порт  
EXPOSE 8080  
  
# Команда для запуска приложения  
CMD ["/app"]
```

4) Сборка Docker-образа

Выполним команду для сборки образа *docker build -t go-app* (Рис. 6). После успешной сборки мы получим образ с наименованием go-app.

```
kravtsovaee@alenaubuntu:~/go-app$ docker build -t go-app .
[+] Building 29.9s (9/9) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 614B                                0.0s
=> [internal] load metadata for docker.io/library/golang:1.20-alpine 2.1s
=> [internal] load .dockerignore                                    0.0s
=> => transferring context: 2B                                       0.0s
=> [1/4] FROM docker.io/library/golang:1.20-alpine@sha256:e47f121850f4e276b2b210c56df3fda9191278dd84a3a442bfe0b09934 20.4s
=> => resolve docker.io/library/golang:1.20-alpine@sha256:e47f121850f4e276b2b210c56df3fda9191278dd84a3a442bfe0b09934 0.0s
=> => sha256:4abcf20661432fb2d719aaf90656f55c287f8ca915dc1c92ec14ff61e67fbaf8 3.41MB / 3.41MB 1.4s
=> => sha256:e8e7baba97f57fa5df2e96f78c627013fec3c450d844769a62de7f40cc5bbbed1 284.20kB / 284.20kB 0.6s
=> => sha256:3bc7f8f202272c1476692180b407ca56cc50f79b8b1859dcd5d579586b5cebce 101.16MB / 101.16MB 12.9s
=> => sha256:e47f121850f4e276b2b210c56df3fda9191278dd84a3a442bfe0b09934462a8f 1.65kB / 1.65kB 0.0s
=> => sha256:008f5b5d4645836f4074cbd9f44c513ba7eb00bc3859f08bbfdba24fd4dae65d 1.36kB / 1.36kB 0.0s
=> => sha256:71719a2da3d19db6340a72b90f937507cbcfcbcaf1fb12835a214d6e8c16a650 1.98kB / 1.98kB 0.0s
=> => sha256:027e8f7f47157b8e955bc20d9874e68eb427280f2b614af061d1f8011434f751 175B / 175B 0.9s
=> => sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 32B / 32B 1.1s
=> => extracting sha256:4abcf20661432fb2d719aaf90656f55c287f8ca915dc1c92ec14ff61e67fbaf8 0.1s
=> => extracting sha256:e8e7baba97f57fa5df2e96f78c627013fec3c450d844769a62de7f40cc5bbbed1 0.1s
=> => extracting sha256:3bc7f8f202272c1476692180b407ca56cc50f79b8b1859dcd5d579586b5cebce 6.9s
=> => extracting sha256:027e8f7f47157b8e955bc20d9874e68eb427280f2b614af061d1f8011434f751 0.0s
=> => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 0.0s
=> [internal] load build context                                    0.0s
=> => transferring context: 320B                                       0.0s
=> [2/4] WORKDIR /app                                              0.5s
=> [3/4] COPY main.go .                                           0.1s
=> [4/4] RUN go build -o app main.go                               6.4s
=> exporting to image                                              0.3s
=> => exporting layers                                                0.3s
=> => writing image sha256:490e49aelfefb6c650db2235da0720d53679a6088e139b07ac5cb086c3c42eb 0.0s
=> => naming to docker.io/library/go-app                             0.0s
kravtsovaee@alenaubuntu:~/go-app$
```

Рис. 6 – Сборка образа

5) Запуск контейнера

Запустим контейнер на основе созданного образа *docker run -d --name my-go-app -p 8080:8080 go-app* (Рис. 7).

```
kravtsovaee@alenaubuntu:~/go-app$
kravtsovaee@alenaubuntu:~/go-app$
kravtsovaee@alenaubuntu:~/go-app$ docker run -d --name my-go-app -p 8080:8080 go-app
00e8d2b1f527ad7e9c45bb3bb3666f74556f4c83bed36cf0c409536e4988477f
kravtsovaee@alenaubuntu:~/go-app$
```

Рис. 7 – Запуск контейнера

Проверим работоспособность, открыв в браузере или выполнив команду *curl http://localhost:8080* (Рис. 8).

```
kravtsovaee@alenaubuntu:~/go-app$
kravtsovaee@alenaubuntu:~/go-app$ curl http://localhost:8080
Hello, Go!
kravtsovaee@alenaubuntu:~/go-app$
```

Рис. 8 – curl localhost

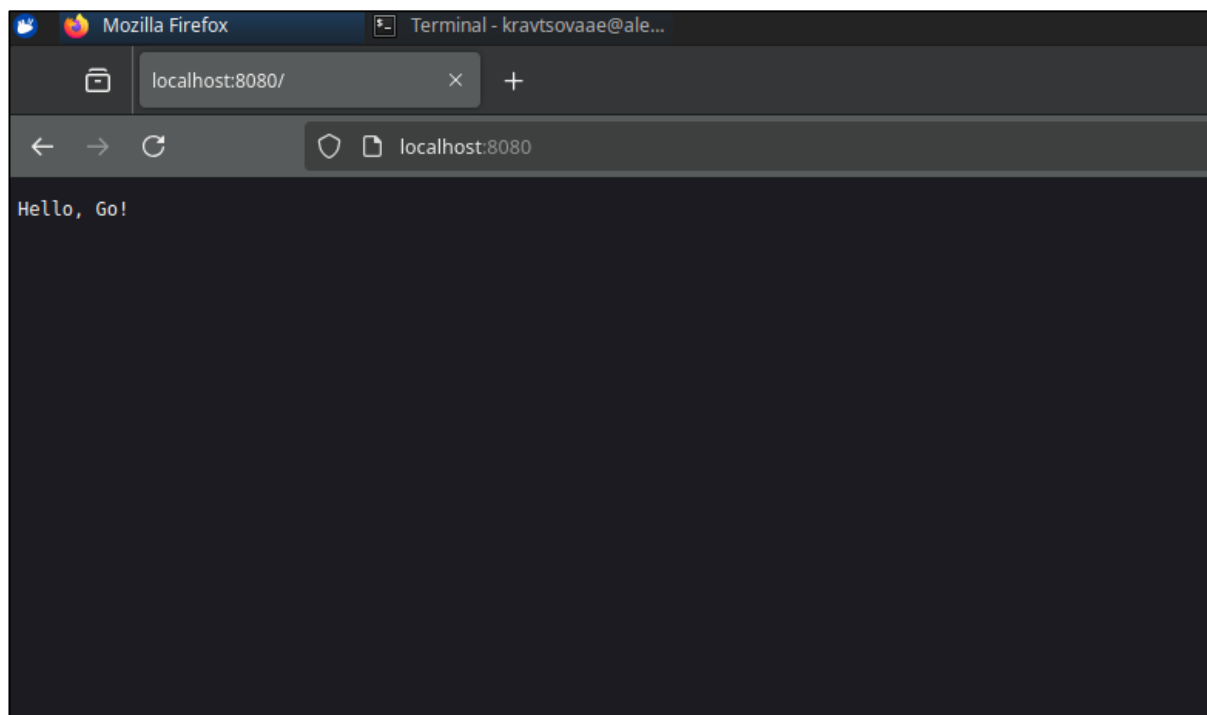


Рис. 9 – Запущенное приложение

Контрольные вопросы:

1) Что такое Dockerfile и для чего он используется?

Dockerfile – это текстовый файл, содержащий набор инструкций для автоматизированной сборки Docker-образа. Он описывает, какие команды нужно выполнить, чтобы создать образ с нужным программным обеспечением и настройками.

2) Какие основные инструкции используются в Dockerfile?

Основные инструкции:

- FROM – задаёт базовый образ.
- WORKDIR – устанавливает рабочую директорию внутри контейнера.
- COPY/ADD – копирует файлы в образ.
- RUN – выполняет команды в процессе сборки.
- EXPOSE – указывает порты, которые будут использоваться.
- CMD/ENTRYPOINT – задаёт команду, которая выполнится при запуске контейнера.

3) Как выполняется сборка образа Docker с использованием Dockerfile?

Сборка образа выполняется командой: *docker build -t имя_образа*.

Эта команда читает Dockerfile из текущей директории и строит образ согласно его инструкциям.

4) Как запустить контейнер из собранного образа?

Контейнер запускается с помощью команды: *docker run [опции] имя_образа*.

5) Каковы преимущества использования Dockerfile для создания образов Docker?

Преимущества:

- Автоматизация – процесс сборки образа автоматизирован и воспроизводим;
- Документированность – все шаги сборки описаны в одном файле;
- Удобство обновлений – можно легко вносить изменения и пересобирать образ;
- Портативность – образы легко перемещаются между разными системами и окружениями.