

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

Кафедра вычислительных систем

Лабораторная работа № 2

Выполнил:

студент группы ИВ-121:

Ермаков А. В.

Работу проверил:

Романюта А.А.

Новосибирск 2023 г.

Содержание

Задание.....	3
Реализация программы.....	4
Листинг программы:	5
Создание контейнера	6
Листинг Dockerfile.....	10
Уровни изоляции	11
Файловая система.....	11
Процессы	12
Сеть.....	13

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 передаётся 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.

Реализация программы

Для начала разработаем программу, которая будет работать в бесконечном цикле, принимать аргументы, из которых на путь файла отвечать выводом его содержимого, а на «Exit» заканчивать свое выполнение.

Код будем писать на языке C#, для этого подключим необходимые библиотеки и определим класс, в котором пишем тернарный оператор, обрабатывающий аргументы командной строки:

```
1 string filePath = args.Length == 1
2   ? args[0]
3   : string.Empty;
```

Далее идет *if*, который выполняется, если пользователь не ввел ни одного аргумента, выводя соответствующее сообщение:

```
1 if (string.IsNullOrEmpty(filePath))
2 {
3     Console.WriteLine("Программа запущена без аргументов командной
4 строки. Для выхода введите 'exit'.");
5 }
```

После чего запускается основной цикл для обработки введенных пользователем данных: если он ввел белиберду (выводится сообщение с призывом ввести не белиберду), путь к файлу (выводит содержимое файла) и *exit* (завершает программу):

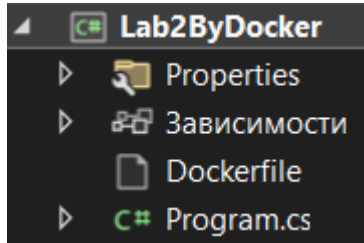
```
1 while (true)
2 {
3     if (string.IsNullOrEmpty(filePath))
4     {
5         Console.WriteLine("Введите путь к файлу:");
6         filePath = Console.ReadLine();
7     }
8
9     if (filePath?.ToLower() == "exit")
10    {
11        Exit();
12    }
13    else if (File.Exists(filePath))
14    {
15        string fileContent = File.ReadAllText(filePath);
16        Console.WriteLine($"Содержимое файла
17 '{filePath}':\n{fileContent}");
18    }
19    else
20    {
21        Console.WriteLine($"Файл не найден. Пожалуйста,
22 проверьте путь.");
23    }
24    filePath = string.Empty;
```

Листинг программы:

```
1 using System;
2 using System.IO;
3
4 class Program
5 {
6     static void Main(string[] args)
7     {
8         string filePath = args.Length == 1
9             ? args[0]
10            : string.Empty;
11
12         if (string.IsNullOrEmpty(filePath))
13         {
14             Console.WriteLine("Программа запущена без аргументов
15 командной строки. Для выхода введите 'exit'.");
16         }
17
18         while (true)
19         {
20             if (string.IsNullOrEmpty(filePath))
21             {
22                 Console.WriteLine("Введите путь к файлу:");
23                 filePath = Console.ReadLine();
24             }
25
26             if (filePath?.ToLower() == "exit")
27             {
28                 Exit();
29             }
30             else if (File.Exists(filePath))
31             {
32                 string fileContent = File.ReadAllText(filePath);
33                 Console.WriteLine($"Содержимое файла
34 '{filePath}':\n{fileContent}");
35             }
36             else
37             {
38                 Console.WriteLine($"Файл не найден. Пожалуйста,
39 проверьте путь.");
40             }
41
42             filePath = string.Empty;
43         }
44     }
45
46     static void Exit()
47     {
48         Console.WriteLine("Программа завершена.");
49         Environment.Exit(0);
50     }
51 }
52 }
```

Создание контейнера

В проекте создаем отдельный файл – *Dockerfile*:



В нем реализуем следующую функциональность:

```
# Используем образ .NET 6.0 SDK для сборки приложения
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
```

```
# Устанавливаем рабочую директорию
WORKDIR /app
```

```
# Копируем csproj-файл и воспроизводим только зависимости
COPY *.csproj ./
RUN dotnet restore
```

```
# Установка дополнительных утилит
RUN apt-get update && apt-get install -y procps net-tools
```

```
# Копируем все остальные файлы и собираем приложение
COPY . ./
RUN dotnet publish -c Release -o out
```

```
# Используем образ .NET 6.0 для запуска приложения
FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /app
COPY --from=build /app/out .
ENTRYPOINT ["dotnet", "Lab2ByDocker.dll"]
```

Далее переходим в *cmd*, где сначала переходим в нужную директорию:

```
1 cd C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker
```

После чего создаем *Docker* образ из *Dockerfile*, задавая ему имя:

```
1 docker build -t myapp-container-x .
```

-t указывает, что создаваемому *Docker*-образу будет присвоена метка *myapp-container-x*.

Docker - это команда для взаимодействия с *Docker*-демоном или *Docker*-сервером.

В результате создается *Docker image*:

```
C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>docker build -t myapp-container-x .
[+] Building 4.6s (16/16) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 1.14kB                             0.0s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                     0.0s
=> [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:6.0 0.4s
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:6.0   0.4s
=> [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:6.0@sha256:0f4f696537a786fbe9f3f98a0ceccccc054bd1a9b81b64aa06d6 0.0s
=> [stage-1 1/3] FROM mcr.microsoft.com/dotnet/aspnet:6.0@sha256:78db2415ff20e22d2f3fd6078a8fff5e17f327729d34e1d 0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 343.32kB                             0.0s
=> CACHED [build 2/7] WORKDIR /app                             0.0s
=> CACHED [build 3/7] COPY *.csproj ./                          0.0s
=> CACHED [build 4/7] RUN dotnet restore                        0.0s
=> CACHED [build 5/7] RUN apt-get update && apt-get install -y procps net-tools 0.0s
=> [build 6/7] COPY . ./                                       0.2s
=> [build 7/7] RUN dotnet publish -c Release -o out            3.6s
=> CACHED [stage-1 2/3] WORKDIR /app                             0.0s
=> [stage-1 3/3] COPY --from=build /app/out .                  0.1s
=> exporting to image                                           0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:4f49b582d68dbc51c1103b2334507dce6bda893d9502af42813417ecd7406e61 0.0s
=> => naming to docker.io/library/myapp-container-x            0.0s

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview

C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>
```

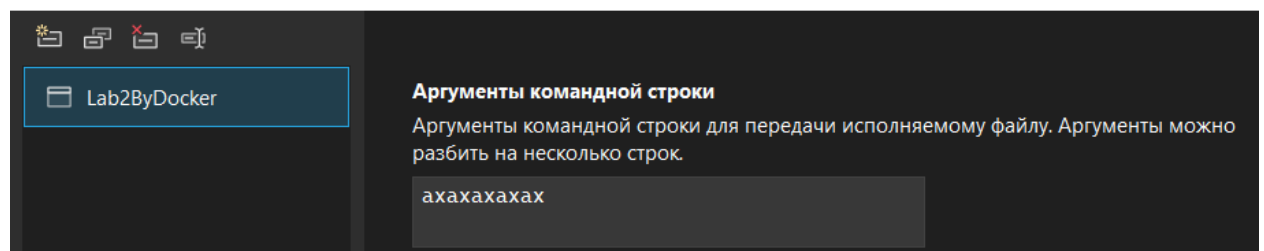
После чего создаем и запускаем контейнер с именем *myapp-instance* в интерактивном режиме (-it) следующей командой:

1	<code>docker run -it --name myapp-instance myapp-container-x</code>
---	---

```
C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>docker run -it --name myapp-instance myapp-container-x
Программа запущена без аргументов командной строки. Для выхода введите 'exit'.
Введите путь к файлу:
ddd
Файл не найден. Пожалуйста, проверьте путь.
Введите путь к файлу:
```

На момент запуска контейнера у нас в качестве аргумента командной строки использовалось следующее:

Профили запуска



Передадим при запуске аргумент *exit*:

```
C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>docker run -it --name myapp-instance-2 myapp-container-x  
exit  
Программа завершена.
```

Далее запускаем контейнер, поскольку мы его остановили:

1	<code>docker start myapp-instance-2</code>
---	--

```
C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>docker start myapp-instance-2  
myapp-instance-2
```

```
C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>
```

После этого погружаемся в файловую систему контейнера:

1	<code>docker exec -it myapp-instance-2 /bin/bash</code>
---	---

Exec - это подкоманда *Docker*, которая используется для выполнения команд внутри контейнера.

-it - это опции команды, которые выполняют два действия:

1) *-i* подключает ввод к контейнеру, позволяя вам взаимодействовать с командами внутри контейнера.

2) *-t* выделяет терминал, что делает командную строку более интерактивной и позволяет видеть вывод команд.

/bin/bash - это команда, которую вы хотите выполнить внутри контейнера. В данном случае, это команда *Bash*, что позволяет вам войти в контейнер и использовать интерактивную оболочку для выполнения дополнительных команд внутри контейнера.

```
C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>docker exec -it myapp-instance-2 /bin/bash  
root@c3023391773c:/app#
```

Теперь мы находимся внутри контейнера и можем взаимодействовать с ним через оболочку *Bash*.

Посмотрим на директории контейнера с помощью команды *ls*:

```
C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>docker exec -it myapp-instance-2 /bin/  
root@c3023391773c:/app# ls  
Lab2ByDocker Lab2ByDocker.deps.json Lab2ByDocker.dll Lab2ByDocker.pdb Lab2ByDocker.runtimeconfig.json  
root@c3023391773c:/app#
```

Далее нехитрыми манипуляциями создаем файл *a.txt*, который потом выведет контейнер при запуске на экран:

1	<code>touch a.txt</code>
---	--------------------------

После чего запикиваем в него содержимое:

1	<code>echo "Really??" > a.txt</code>
---	---

Пояснение: если использовать только одну «>» в команде `echo "Really??" > a.txt`, то текст вставляется с удалением предыдущего, а если использовать две галочки, то в конец файла, за тестом, что уже имеется.

```
root@c3023391773c:/app# echo "Really??" > a.txt
root@c3023391773c:/app#
```

По итогу получаем такой результат:

```
root@c3023391773c:/app# cat a.txt
Really??
root@c3023391773c:/app#
```

Теперь видим файл *a.txt* в файловой системе докера:

```
C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>docker exec -it myapp-instance-2 /bin/bash
root@c3023391773c:/app# ls
Lab2ByDocker  Lab2ByDocker.deps.json  Lab2ByDocker.dll  Lab2ByDocker.pdb  Lab2ByDocker.runtimeconfig.json  a.txt
root@c3023391773c:/app#
```

После чего выходим отсюда:

```
root@c3023391773c:/app# exit
exit
C:\Users\hasee\Desktop\5 семестр\авс\lab2\Lab2ByDocker\Lab2ByDocker>
```

И запускаем выполнение нашей программы, указав в качестве аргумента путь к файлу *a.txt*:

1	<code>docker exec -it myapp-instance-2 /app/Lab2ByDocker /app/a.txt</code>
---	--

/app/Lab2ByDocker - это путь к исполняемому файлу или команде, который вы хотите выполнить внутри контейнера. В данном случае, это */app/Lab2ByDocker* - это исполняемый файл внутри контейнера.

/app/a.txt - это аргумент команды. Он указывает на файл, который вы хотите передать в качестве аргумента.

```
C:\Users\hasee\Desktop\5sem\авс\lab2\Lab2ByDocker\Lab2ByDocker>docker exec -it myapp-instance-2 /app/Lab2ByDocker /app/a.txt
Содержимое файла '/app/a.txt':
Really??

Введите путь к файлу:
C:\Users\hasee\Desktop\5sem\авс\lab2\Lab2ByDocker\Lab2ByDocker>docker exec -it myapp-instance-2 /bin/bash
```

Ну и как можно заметить, содержимое файла успешно вывелось.

Листинг Dockerfile

1	FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
2	
3	WORKDIR /app
4	
5	COPY *.csproj ./
6	RUN dotnet restore
7	
8	RUN apt-get update && apt-get install -y procps net-tools
9	
10	COPY . ./
11	RUN dotnet publish -c Release -o out
12	
13	FROM mcr.microsoft.com/dotnet/aspnet:6.0
14	WORKDIR /app
15	COPY --from=build /app/out .
16	ENTRYPOINT ["dotnet", "Lab2ByDocker.dll"]

Уровни изоляции

Файловая система

Для начала посмотрим на файловую систему с помощью команды `ls`:

```
root@58348a51ed0a:/app# ls
Lab2ByDocker  Lab2ByDocker.deps.json  Lab2ByDocker.dll  Lab2ByDocker.pdb  Lab2ByDocker.runtimeconfig.json
```















Напоминаем, что до этого нужно использовать команду:

```
1 docker exec -it myapp-instance-2 /bin/bash
```

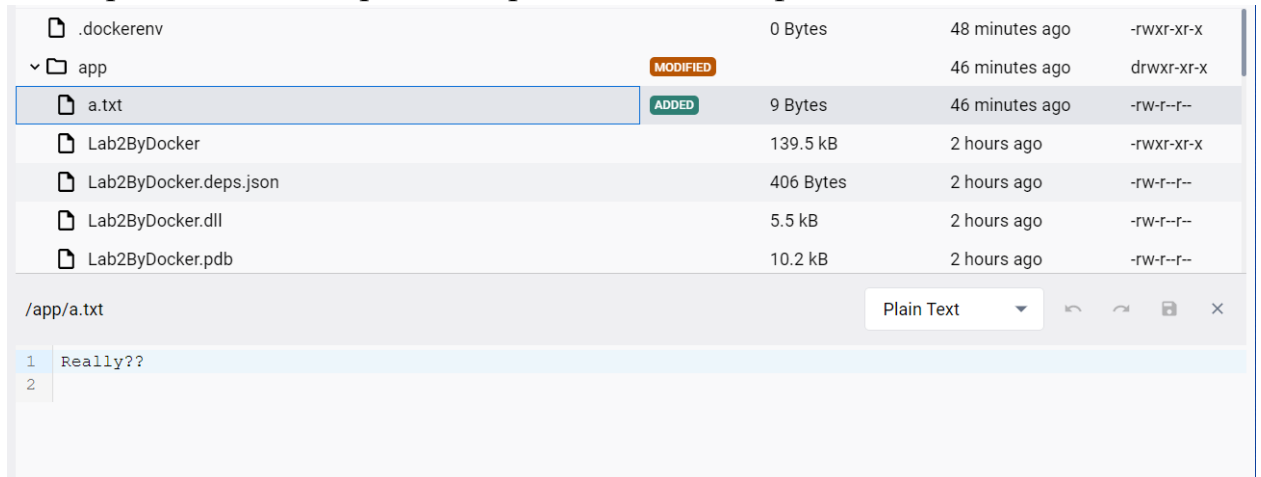
Можно попасть и в корневую папку:

```
root@efd814008a:/bin# ls
bash  date      echo  gunzip  ls      mountpoint  readlink  stty  umount  zcmp  zmore
cat   dd        egrep gzexe  lsblk  mv         rm        su    uname  zdiff  znew
chgrp df        false gzip   mkdir  nisdomainname  rmdir    sync  uncompress  zegrep
chmod dir      fgrep hostname  mknod  pidof    run-parts  tar      vdir   zfgrep
chown dmesg     findmnt  kill     mktemp  ps         sed      tempfile  wdctl  zforce
cp    dnsdomainname  fuser   ln       more    pwd       sh       touch  ypdomainname  zgrep
dash  domainname    grep    login    mount   rbash     sleep    true   zcat      zless
root@efd814008a:/bin#
```

Но гораздо проще и более подробно можно рассмотреть в программе *Docker Desktop*:

Logs Inspect Bind mounts Exec <u>Files</u> Stats					Open file editor	
Name ↑	Note	Size	Last modified	Mode		
 .dockerenv		0 Bytes	48 minutes ago	-rwxr-xr-x		
>  app	MODIFIED		46 minutes ago	drwxr-xr-x		
>  bin	MODIFIED		27 minutes ago	drwxr-xr-x		
>  boot			28 days ago	drwxr-xr-x		
>  dev			47 minutes ago	drwxr-xr-x		
>  etc	MODIFIED		27 minutes ago	drwxr-xr-x		
>  home			28 days ago	drwxr-xr-x		
>  lib	MODIFIED		19 days ago	drwxr-xr-x		
>  lib64			19 days ago	drwxr-xr-x		
>  media			19 days ago	drwxr-xr-x		
>  mnt			19 days ago	drwxr-xr-x		
>  opt			19 days ago	drwxr-xr-x		
>  proc			47 minutes ago	dr-xr-xr-x		
>  root	MODIFIED		46 minutes ago	drwx-----		

И как раз таки посмотреть содержимое нашего файла:



Здесь же его можно и отредактировать.

Процессы

Далее выполняем следующие команды, обновляя существующие и скачивая отсутствующие утилиты:

```
1 apt update
2 apt install procs
```

После чего выполняем команду `ps`, чтобы посмотреть на процессы:

```
root@efdfd814008a:/bin# ps
  PID TTY          TIME CMD
   42 pts/1        00:00:00 bash
  384 pts/1        00:00:00 ps
root@efdfd814008a:/bin#
```

Или использовать команду:

```
1 ps aux
```

```
root@efdfd814008a:/bin# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2 2929436 23360 pts/0    Ss+  17:45   0:00 dotnet Lab2ByDocker.dll
root        36  0.0  0.0   2480    584 pts/2    Ss+  17:50   0:00 /bin/sh
root        42  0.0  0.0   4028   3376 pts/1    Ss   17:51   0:00 /bin/bash
root       388  0.0  0.0   6756   2948 pts/1    R+   18:36   0:00 ps aux
root@efdfd814008a:/bin#
```

Aux состоит из:

a – отображение процессов всех пользователей.

u – вывод более подробной информации о пользователе.

x – показывать процессы, которые не связаны с терминалом.

Далее команда *top*, предоставляет интерактивное окно с отображением текущей активности системы:

```
root@efdfd814008a:/bin# top
top - 18:37:46 up 12:05, 0 users, load average: 0.29, 0.18, 0.15
Tasks: 4 total, 1 running, 3 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.1 us, 0.2 sy, 0.0 ni, 99.5 id, 0.2 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7845.1 total, 2056.9 free, 1611.3 used, 4176.9 buff/cache
MiB Swap: 2048.0 total, 2048.0 free, 0.0 used. 5931.7 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	2929436	23360	19592	S	0.3	0.3	0:00.06	dotnet
36	root	20	0	2480	584	512	S	0.0	0.0	0:00.00	sh
42	root	20	0	4028	3376	2892	S	0.0	0.0	0:00.02	bash
389	root	20	0	6988	3332	2860	R	0.0	0.0	0:00.00	top

Здесь выводится следующая информация:

- 1) Информация о загрузке процессора.
- 2) Список активных процессов.
- 3) Использование памяти.
- 4) Загрузка сети.
- 5) Загрузка диска.

Сеть

Теперь посмотрим на изоляцию сети, для этого выполним следующую команду:

```
1 | docker network ls
```

```
C:\Users\hasee>docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
6ed95412474a    bridge    bridge      local
c4257f0c8169    host      host        local
24a005dbba4c    none      null        local
```

Для более подробной информации используем следующую команду:

```
1 | docker inspect <номер id>
```

```
C:\Users\hasee>docker inspect 6ed95412474a
```

```
[
  {
    "Name": "bridge",
    "Id": "6ed95412474a99aef9a34c18fae715dbc11893a35d56a514b9d8c7a030a6b3d2",
    "Created": "2023-10-26T15:45:36.750418986Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "472e753d7955336307ef1c2791fbd27c8ebb8e802f555fe9dc0eac31d24bfd6d": {
        "Name": "myapp-instance",
        "EndpointID": "74e5fb508933c2a9471ff95cda94d069b3a20fa8e1e5e2c564c6bc1f29e5675e",
        "MacAddress": "02:42:ac:11:00:04",
        "IPv4Address": "172.17.0.4/16",
        "IPv6Address": ""
      },
      "78a13b7821d02253ba7b3f1b35dbe76f1846d67ba919e0e02ca77198b31793f4": {
        "Name": "myapp-container",
        "EndpointID": "80b766ffceb746223aa86223ad40e20f4b800e3ef636dbb2ac583c735d78870f",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
      },
      "a0e842a46d52466ec773a9dadbcfb909c7577e4e5aa5025e254f2eadfb4412": {
        "Name": "myapp-container-2",
        "EndpointID": "3883e7c713e5deb8db066cbe4f2faa422af12e61fe11c89b7e1a6ca8214294d4",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
      },
      "efdfd814008a9585adf6c7f79ba5d48a3db48e520c6a76a7c6944cc568946400": {
        "Name": "myapp-instance-2",
        "EndpointID": "68b6ef840440e52f34bfb220e230d14274856c052c958b99faed6eddd4777b31",
        "MacAddress": "02:42:ac:11:00:05",
        "IPv4Address": "172.17.0.5/16",
        "IPv6Address": ""
      }
    }
  }
],
```

```
    },  
    "Options": {  
      "com.docker.network.bridge.default_bridge": "true",  
      "com.docker.network.bridge.enable_icc": "true",  
      "com.docker.network.bridge.enable_ip_masquerade": "true",  
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",  
      "com.docker.network.bridge.name": "docker0",  
      "com.docker.network.driver.mtu": "1500"  
    },  
    "Labels": {}  
  }  
]  
  
C:\Users\hasee>
```

Как мы видим здесь предоставляется вся необходимая информация о сети, о конфигах, о контейнерах и их настройках.