Инструкция по использованию Docker

СОДЕРЖАНИЕ

1 Теоретическая часть

1.1 Обзор понятий Docker образ и Docker контейнер

Контейнер Docker – это среда выполнения со всеми необходимыми компонентами, такими как код, зависимости и библиотеки, которые необходимы для запуска кода приложения без использования зависимостей хост-машины. Эта среда выполнения контейнера работает на движке на сервере, машине или облачном инстансе. Движок запускает несколько контейнеров в зависимости от доступных базовых ресурсов. Схема работы технологии контейнеризации представлена на рисунке 1.1.

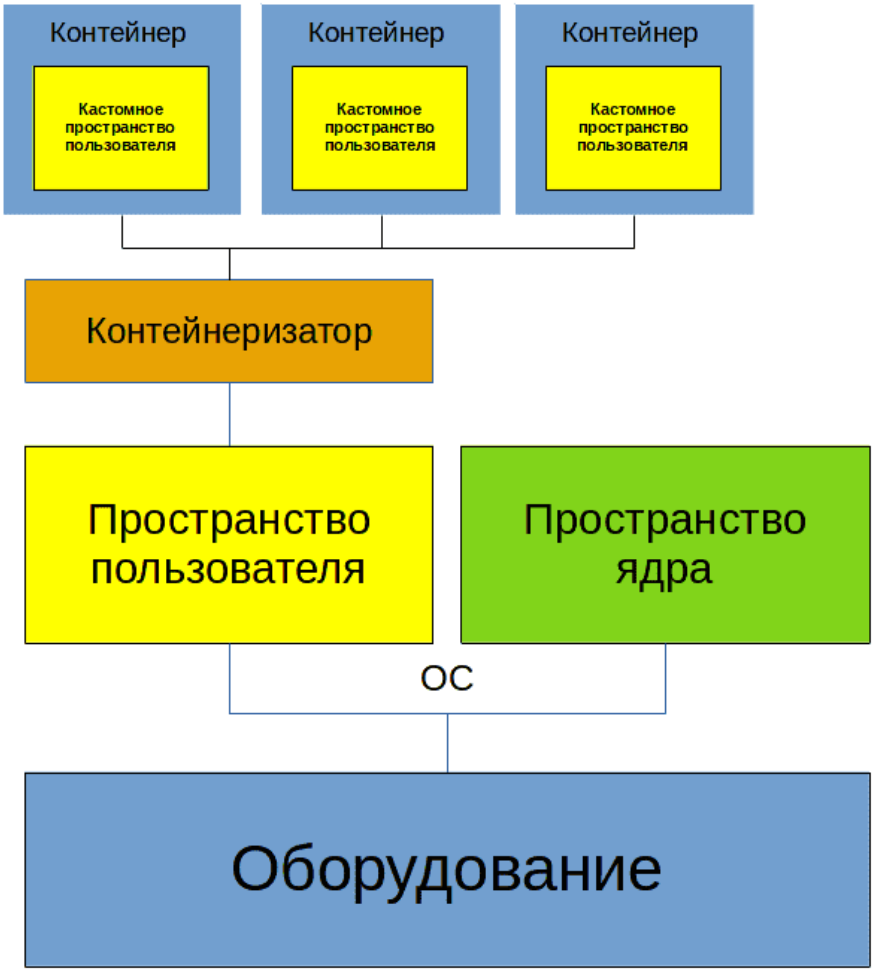

Рисунок 1.1 — Схема работы контейнеров

Образ Docker или образ контейнера – это отдельный исполняемый файл, используемый для создания контейнера. Этот образ контейнера содержит все библиотеки, зависимости и файлы, необходимые для запуска контейнера. Образ Docker можно использовать совместно и переносить, поэтому один и тот же образ можно развернуть сразу в нескольких местах – так же, как двоичный файл программного обеспечения.

Образы можно хранить в реестрах, чтобы отслеживать сложные архитектуры программного обеспечения, проекты, бизнес-сегменты и доступ к группам пользователей. Например, публичный реестр Docker Hub содержит такие образы, как операционные системы, фреймворки языков программирования, базы данных и редакторы кода.

1.2 Репозиторий Harbor

Harbor — это бесплатный реестр для хранения Docker образов с открытым исходным кодом, который предоставляет доступ к образам с помощью политик, а также умеет сканировать образы на наличие уязвимостей. Функциональные возможности репозитория Harbor:

— Пользовательский веб-интерфейс для просмотра репозиториев, поиска по ним, управления проектами;

— Наличие панели администратора;

— Управление доступом пользователей к проектам на основе их ролей;

— Возможность просмотра истории всех выполненных операций;

— Наличие возможности использования REST API запросов для управления репозиторием.

2 Практическая часть

2.1 Установка Docker

Чтобы установить *Docker* на локальную машину, необходимо запустить установочный скрипт для все операционных систем или запустить установочный скрипт в зависимости от версии операционной системы:

— Для нескольких ОС: docker-install.sh.

На рисунке 2.1 представлен алгоритм данного скрипта.

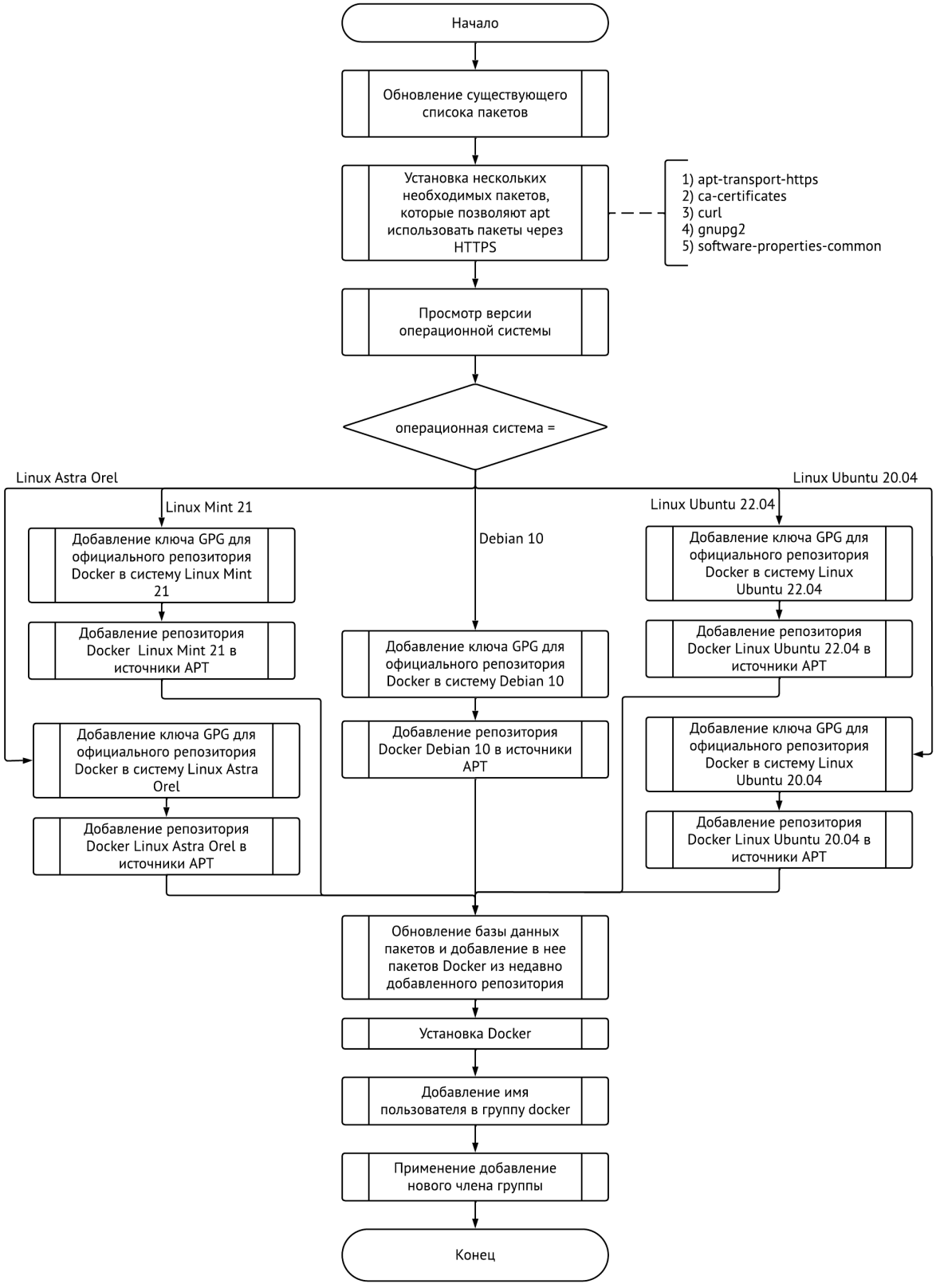

Рисунок 2.1 — Алгоритм работы блока ”Установка Docker на локальную машину”у скрипта docker-install.sh

```
1   #!/bin/bash
2   sudo apt −y update && apt −y upgrade
```

```
3   sudo apt install -y apt-transport-https ca-certificates curl gnupg2 software-properties-
        common
4   version=$(cat /etc/issue)
5
6   if [[ .*$version.* =~ .*"Astra".* ]]; then
7           curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
8           echo "deb [arch=amd64] https://download.docker.com/linux/debian stretch
                stable" | sudo tee -a /etc/apt/sources.list
9   fi
10
11  if [[ .*$version.* =~ .*"Mint".* ]]; then
12          curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
13          echo "deb [arch=amd64] https://download.docker.com/linux/debian stretch
                stable" | sudo tee -a /etc/apt/sources.list
14  fi
15
16  if [[ .*$version.* =~ .*"Debian".* ]]; then
17          curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
18          sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/
                linux/debian $(lsb_release -cs) stable"
19  fi
20
21  if [[ .*$version.* =~ .*"Ubuntu".* && .*$version.* =~ .*"20.04".* ]]; then
22          curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
23          sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/
                linux/ubuntu focal stable"
24  fi
25
26  if [[ .*$version.* =~ .*"Ubuntu".* && .*$version.* =~ .*"22.04".* ]]; then
27          curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
                dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
28          echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/
                docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $
                (lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/
                null
29  fi
30
31  sudo apt update
32
33  sudo apt install -y docker-ce
```

```
34
35          sudo usermod −aG docker ${USER}
36
37          su − ${USER}
```

— Для Astra Linux: docker-install-astra.sh

```
1           #!/bin/bash
2
3           sudo apt update && apt upgrade
4           sudo apt−get install apt−transport−https ca−certificates curl gnupg2
5           curl −fsSL https://download.docker.com/linux/debian/gpg | sudo apt−key add −
6           echo ”deb [arch=amd64] https://download.docker.com/linux/debian stretch stable” | sudo
                tee −a /etc/apt/sources.list
7           sudo apt update
8           sudo apt install docker−ce
9           sudo curl −L https://github.com/docker/compose/releases/download/1.23.2/docker−
                compose−Linux−x86_64 −o /usr/local/bin/docker−compose
10          sudo chmod +x /usr/local/bin/docker−compose
11          sudo usermod −aG docker ${USER}
12          su − ${USER}
```

— Для Debian-10-12: docker-install-debian-10.sh

```
1           #!/bin/bash
2
3           sudo apt update
4           sudo apt install −y apt−transport−https ca−certificates curl gnupg2 software−properties−
                common
5           curl −fsSL https://download.docker.com/linux/debian/gpg | sudo apt−key add −
6           sudo add−apt−repository ”deb [arch=amd64] https://download.docker.com/linux/debian
                $(lsb_release −cs) stable”
7           sudo apt update
8           sudo apt install docker−ce
9           sudo usermod −aG docker ${USER}
10          su − ${USER}
```

— Для Ubuntu 20: docker-install-ubuntu-20.sh

```
1           #!/bin/bash
2
3           sudo apt update
```

```
4       sudo apt install apt-transport-https ca-certificates curl software-properties-common
5       curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
6       sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
            focal stable"
7       sudo apt update
8       sudo apt install docker-ce
9       sudo usermod -aG docker ${USER}
10      su - ${USER}
```

— Для Ubuntu 22: docker-install-ubuntu-22.04.sh

```
1       #!/bin/bash
2
3       sudo apt update
4       sudo apt install apt-transport-https ca-certificates curl software-properties-common
5       curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/
            share/keyrings/docker-archive-keyring.gpg
6       echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
            archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs)
             stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
7       sudo apt update
8       sudo apt install docker-ce
9       sudo usermod -aG docker ${USER}
10      su - ${USER}
```

Если скрипт отработал без ошибок, то перезапустите компьютер.

После этого необходимо создать файл для Docker демона $/etc/docker/daemon.$ в которой нужно прописать строчку для исключения ip-адреса сервера.

/etc/docker/daemon.json:

```
1   {
2       "insecure-registries": ["192.168.0.168:4431"]
3   }
```

И перезапустить Docker с помощью команды:

```
1       sudo systemctl restart docker
```

После этого в пункте insecure-registries появится нужный ip-адрес. Для данной проверки необходимо ввести следующую команду:

```
1  docker info
```

2.2 Основные команды Docker

• Синтаксис имеет следующую форму:

```
1  docker [option] [command] [arguments]
```

• Чтобы просмотреть все доступные субкоманды, введите:

```
1  docker
```

Output :

```
1  attach Attach local standard input, output, and error streams to a running container
2  build Build an image from a Dockerfile
3  commit Create a new image from a container's changes
4  cp Copy files/folders between a container and the local filesystem
5  create Create a new container
6  diff Inspect changes to files or directories on a container's filesystem
7  events Get real time events from the server
8  exec Run a command in a running container
9  export Export a container's filesystem as a tar archive
10 history Show the history of an image
11 images List images
12 import Import the contents from a tarball to create a filesystem image
13 info Display system−wide information
14 inspect Return low−level information on Docker objects
15 kill Kill one or more running containers
16 load Load an image from a tar archive or STDIN
17 login Log in to a Docker registry
18 logout Log out from a Docker registry
19 logs Fetch the logs of a container
20 pause Pause all processes within one or more containers
21 port List port mappings or a specific mapping for the container
22 ps List containers
23 pull Pull an image or a repository from a registry
24 push Push an image or a repository to a registry
25 rename Rename a container
26 restart Restart one or more containers
```

```
27 rm Remove one or more containers
28 rmi Remove one or more images
29 run Run a command in a new container
30 save Save one or more images to a tar archive (streamed to STDOUT by default)
31 search Search the Docker Hub for images
32 start Start one or more stopped containers
33 stats Display a live stream of container(s) resource usage statistics
34 stop Stop one or more running containers
35 tag Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
36 top Display the running processes of a container
37 unpause Unpause all processes within one or more containers
38 update Update configuration of one or more containers
39 version Show the Docker version information
40 wait Block until one or more containers stop, then print their exit codes
```

• Чтобы просмотреть параметры, доступные для конкретной команды, введите:

```
1 docker docker−subcommand −−help
```

• Чтобы просмотреть общесистемную информацию о Docker, введите следующее:

```
1 docker info
```

2.3 Работа с образами Docker

Контейнеры Docker получают из образов Docker. По умолчанию Docker загружает эти образы из Docker Hub, реестр Docker, контролируемые Docker, т.е. компанией, реализующей проект Docker. Любой может размещать свои образы Docker на Docker Hub, поэтому большинство приложений и дистрибутивов Linux, которые вам потребуется, хранят там свои образы. Чтобы проверить, можно ли получить доступ к образам из Docker Hub и загрузить их, введите следующую команду:

```
1 docker run hello−world
```

Данный вывод говорит о том, что Docker работает корректно:

$Output$:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:6a65f928fb91fcfbc963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f44f1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

...
```

Вы можете выполнять поиск доступных на Docker Hub с помощью команды *docker* с субкомандой *search*. Например, чтобы найти образ Ubuntu, введите:

```
docker search ubuntu
```

Скрипт пробежится по Docker Hub и вернет список всех образов с именами, совпадающими со строкой запроса. В данном случае вывод будет выглядеть примерно следующим образом:

Output :

В столбце OFFICIAL значение OK указывает на образ, созданный и поддерживаемый компанией, реализующей проект. После того как вы определили образ, который хотели бы использовать, вы можете загрузить его на свой компьютер с помощью субкоманды *pull*.

```
docker pull ubuntu
```

Output :

```
Using default tag: latest
latest: Pulling from library/ubuntu
d51af753c3d3: Pull complete
fc878cd0a91c: Pull complete
6154df8ff988: Pull complete
fee5db0ff82f: Pull complete
Digest: sha256:747d2dbbaaee995098c9792d99bd333c6783ce56150d1b11e333bbceed5c54d7
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

После того как образ будет загружен, вы сможете запустить контейнер с помощью загруженного образа с помощью субкоманды run. Как вы уже видели на примере $hello - world$, если образ не был загружен, когда $docker$ выполняется с субкомандой run, клиент Docker сначала загружает образ, а затем запускает контейнер с этим образом. Чтобы просмотреть образы, которые были загружены на ваш компьютер, введите:

```
1 docker images
```

Вывод команды должен выглядеть примерно следующим образом:

2.4 Запуск контейнеров Docker

Контейнер $hello - world$, который вы запустили на предыдущем шаге, служит примером контейнера, который запускается и завершает работу после отправки тестового сообщения. Контейнеры могут быть гораздо более полезными, чем в примере выше, а также могут быть интерактивными. В конечном счете они очень похожи на виртуальные машины, но более бережно расходуют ресурсы. В качестве примера мы запустим контейнер с самым последним образом образ Ubuntu. Сочетание переключателей $-i$ и $-t$ предоставляет вам доступ к интерактивной командной оболочке внутри контейнера:

```
1 docker run -it ubuntu
```

Необходимо изменить приглашение к вводу команды, чтобы отразить тот факт, что вы работаете внутри контейнера, и должны иметь следующую форму:

```
1 root@d9b100f2f636:/#
```

Обратите внимание на идентификатор контейнер в запросе команды. В данном примере это $d9b100f2f636$. Вам потребуется этот идентификатор для определения контейнера, когда вы захотите его удалить. Теперь вы можете запустить любую команду внутри контейнера. Например, сейчас мы обновим базу данных пакетов внутри контейнера. Вам не потребуется

начинать любую команду с $sudo$, потому что вы работаете внутри контейнера как $root$-пользователь:

```
root@d9b100f2f636:/# apt update
```

Чтобы выйти из контейнера, введите $exit$.

```
root@d9b100f2f636:/# exit
```

2.5 Управление контейнерами Docker

После использования Docker в течение определенного времени, у вас будет много активных (запущенных) и неактивных контейнеров на компьютере. Чтобы просмотреть активные, используйте следующую команду:

```
docker ps
```

Чтобы просмотреть все контейнеры — активные и неактивные, воспользуйтесь командой $docker\ ps$ с переключателем $-a$:

```
docker ps -a
```

Чтобы запустить остановленный контейнер, воспользуйтесь $docker\ start$ с идентификатором контейнера или именем контейнера. Давайте запустим контейнер на базе Ubuntu с идентификатором $1c08a7a0d0e4$:

```
docker start 1c08a7a0d0e4
```

Контейнер будет запущен, а вы сможете использовать $docker\ ps$, чтобы просматривать его статус.

Чтобы остановить запущенный контейнер, используйте $docker\ stop$ с идентификатором или именем контейнера. На этот раз мы будем использовать имя, которое Docker присвоил контейнеру, т.е. $quizzical - mcnulty$.

```
docker stop quizzical_mcnulty
```

После того как вы решили, что вам больше не потребуется контейнер, удалите его с помощью команды $docker\ rm$, снова добавив идентификатор контейнера или его имя. Используйте команду $docker\ ps\ -a$, чтобы найти

идентификатор или имя контейнера, связанного с образом $hello - world$, и удалить его.

```
1 docker rm youthful_curie
```

Вы можете запустить новый контейнер и присвоить ему имя с помощью переключателя $- - name$. Вы также можете использовать переключатель $- -rm$ чтобы создать контейнер, который удаляется после остановки. Изучите команду $docker\ run\ help$, чтобы получить больше информации об этих и прочих опциях.

```
1 docker run help
```

2.6 Основные скрипты для работы с Docker

Чтобы упростить и ускорить работу с Docker было разработано несколько скриптов:

— docker-create-image-and-container-both.sh — скрипт, создающий Docker образ и его контейнер, используя или локальный репозиторий хост-машины или репозиторий Harbor.

— docker-delete-container-image.sh — скрипт удаляет указанный пользователем образ, контейнер, а также чистит кэш Docker.

— docker-open-container.sh — скрипт запускает контейнер и входит его командную строку данного контейнера.

— docker-push.sh — скрипт загружает на репозиторий Docker образы.

— docker-container-to-image.sh — скрипт преобразовывает Docker контейнер в Docker образ.

— test-is-equal-docker-images.sh — скрипт проверяет на совпадение контрольных сумм у образов.

Приведенные выше скрипты используют следующие скрипты:

— info-account.sh — скрипт, в котором прописаны данные пользователя для входа и загрузки образов в Harbor.

— Dockerfile-copy-file — файл с инструкцией для создания Docker образа.

— Dockerfile-create — файл с инструкцией для создания Docker образа. Данная инструкция подгружает графическую оболочку и файлы, которые пользователь положил рядом с этим скриптом.

— .dockerignore — инструкция для Docker. В этом скрипте перечислены файлы, которые нужно игнорировать при использовании команды ”COPY . /app”в скрипте Dockerfile*

Для запуска основных скриптов необходимо прописать следующую команду:

```
./<name−script>.sh
```

Скрипты можно запускать в любом порядке. Самый распространенный вариант:

— docker-create-image-and-container-both.sh — скрипт, создающий Docker образ и его контейнер, используя или локальный репозиторий хост-машины или репозиторий Harbor.

— docker-push.sh — скрипт загружает на репозиторий Docker образы.

— docker-delete-container-image.sh — скрипт удаляет указанный пользователем образ, контейнер, а также чистит кэш Docker.

Перечисленные ниже скрипты используются так часто, так как они используются для внештатных ситуациях или для проверки контрольных у образов.

— docker-open-container.sh — скрипт запускает контейнер и входит его командную строку данного контейнера.

— docker-container-to-image.sh — скрипт преобразовывает Docker контейнер в Docker образ.

— test-is-equal-docker-images.sh — скрипт проверяет на совпадение контрольных сумм у образов.

2.6.1 Загрузка данных пользователя

Название скрипта: info-account.sh

Цель: загрузить в переменные командной строки (bash) необходимые значения для входа в репозиторий Harbor.

Результат: переменные с необходимыми значениями.

ВНИМАНИЕ! На него ссылаются другие скрипты. Пользователю не надо самостоятельно его запускать. Данный скрипт необходимо изменить, если вы заходите в репозиторий не от admin.

```
1 #!/bin/bash
2
3 REGISTRY_URL=”192.168.0.168:4431”
4 USERNAME=”admin”
5 PASSWORD=”Vv12345678”
```

2.6.2 Предоставление прав образу и контейнеру на работу с графической оболочкой локальной машины

Название скрипта: Dockerfile-create

Цель: настроить графическую оболочку в образе.

Результат: можно запустить графическое приложение внутри контейнера.

```
1 ARG image
2
3 FROM $image
4
5 ENV QT_QPA_PLATFORM=xcb
```

2.6.3 Предоставление прав образу и контейнеру на работу с графической оболочкой локальной машины и загрузка в контейнер файлы с локальной машины

Название скрипта: Dockerfile-copy-file

Цель: настроить графическую оболочку в образе и загрузить файлы с хоста.

Результат: можно запустить графическое приложение внутри контейнера. Можно работать с файлами, которые расположены в корневом каталоге.

```
ARG image

FROM $image

ENV QT_QPA_PLATFORM=xcb

COPY . /
```

2.6.4 Игнорирование файлов в папке

Название скрипта: .dockerignore

Цель: игнорировать укачанные файлы в данном скрипте.

Результат: в контейнере не будут присутствовать указанные файлы.

```
Dockerfile_copy−file
Dockerfile_create
.dockerignore
docker−create−image−and−container−both.sh
docker−delete−container−image.sh
docker−push.sh
docker−open−container.sh
test_is−equal−docker−images.sh
docker−install−astra.sh
docker−install−ubuntu−22.04.sh
docker−install−ubuntu−20.sh
docker−install−debian−10.sh
docker−install.sh
info−account.sh
docker−container−to−image.sh
```

2.6.5 Создание Docker образа и его контейнера

Название скрипта: docker-create-image-and-container-both.sh.

Цель: создать образ и его контейнер.

Результат: командная строка внутри созданного контейнера.

Алгоритм работы скрипта представлен на рисунке 2.2 и на рисунке 2.3.

Примечание:

Dockerfile вынесен отдельным пунктом, так как данный файл пользователь может менять по своему усмотрению.

Dockerfile — это файл, содержащий команды для создания docker образа с необходимой наполняемостью. Например, какой образ лежит внутри создаваемого Dockerfile образом. Или какие команды можно автоматически запустить (например, скачивание библиотек).

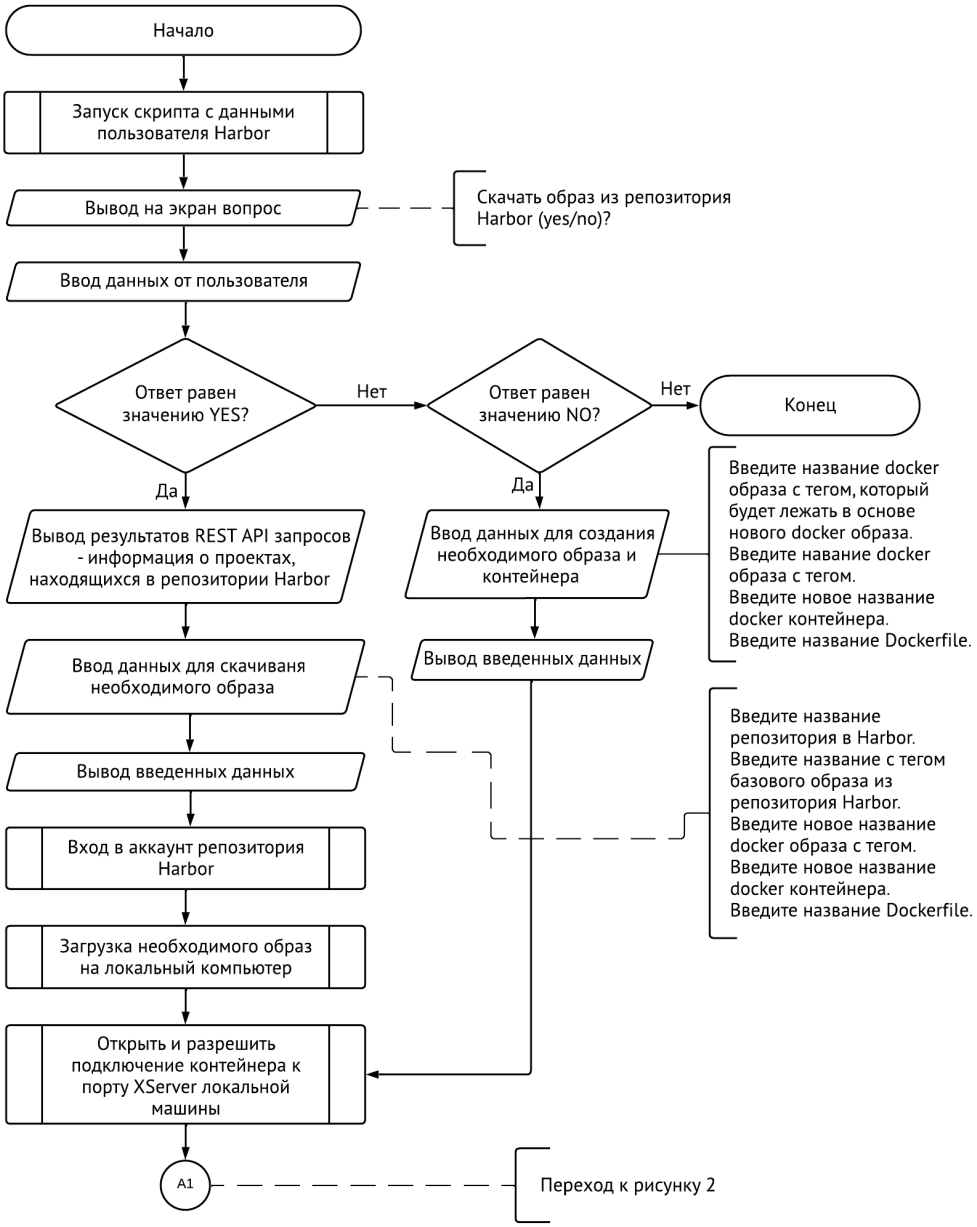

Рисунок 2.2 — Алгоритм работы скрипта docker-create-image-and-container-both.sh

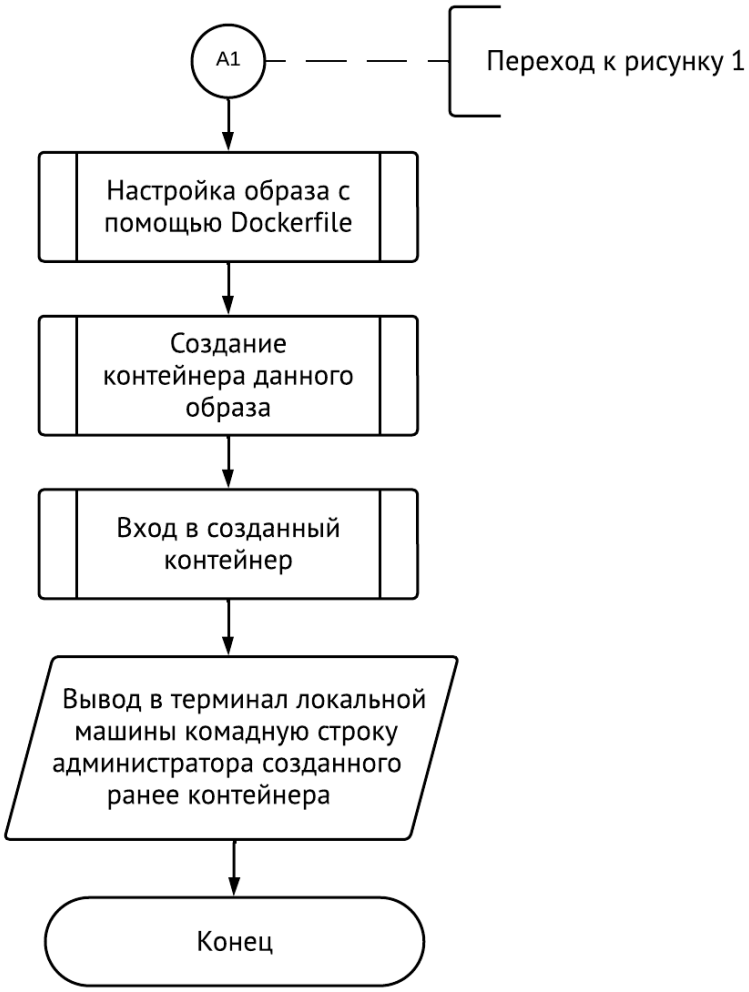

Рисунок 2.3 — Алгоритм работы скрипта docker-create-image-and-container-both.sh

```
1  #!/bin/bash
2
3  . ./info−account.sh
4
5  read −p ”Скачать образ из репозитория Harbor (yes/no): ” answer
6
7  if [[ ”$answer” == ”yes” || ”$answer” == ”y” ]]; then
8          echo ”Вывод репозиториев с проектами: ”
9          curl −u $USERNAME:$PASSWORD −k https://$REGISTRY_URL/v2/_catalog
10         read −p ”Введите название образа с его репозиторием в Harbor: ” name_repo_img
11         curl −u $USERNAME:$PASSWORD −k https://$REGISTRY_URL/v2/
               $name_repo_img/tags/list
12         read −p ”Введите тег образа $name_repo_img из репозитория Harbor: ” name_tag
```

```
13          read −p ”Введите новое название docker образа с тегом через : (по умолч. тег = :
                latest): ” name_work_image
14          read −p ”Введите новое название docker контейнера: ” name_work_container
15          find . −type f −name ”Dockerfile*”
16          echo ”Dockerfile_create − создать образ с графическим интерфейсом”
17          echo ”Dockerfile_copy−file − создать образ с графическим интерфейсом и загрузить
                папки и файлы, которые лежат рядом с этим скриптом”
18          read −p ”Введите название Dockerfile: ” DockerFile
19          var1=”${REGISTRY_URL}/”
20          var2=”$var1$name_repo_img”
21          var3=”${var2}:”
22          name_base_image=”$var3$name_tag”
23          echo ”Было введено:”
24          echo ”1) имя базового докер образа: ” $name_base_image
25          echo ”2) имя docker образа: ” $name_work_image
26          echo ”3) имя docker контейнера: ” $name_work_container
27          echo ”4) имя Dockerfile: ” $DockerFile
28
29          docker login $REGISTRY_URL −u $USERNAME −p $PASSWORD
30          docker pull $name_base_image
31
32  fi
33
34  if [[ ”$answer” == ”no” || ”$answer” == ”n” ]]; then
35          read −p ”Введите название docker образа с тегом, который будет лежать в основе
                нового docker образа: ” name_base_image
36          read −p ”Введите новое название docker образа с тегом (по умолч. тег = latest): ”
                name_work_image
37          read −p ”Введите новое название docker контейнера: ” name_work_container
38          read −p ”Введите название Dockerfile: ” DockerFile
39          echo ”Было введено:”
40          echo ”1) имя docker образа: ” $name_base_image
41          echo ”2) имя docker образа: ” $name_work_image
42          echo ”3) имя docker контейнера: ” $name_work_container
43          echo ”4) имя Dockerfile: ” $DockerFile
44  fi
45
46  if [[ ”$answer” != ”no” && ”$answer” != ”n” && ”$answer” != ”yes” && ”$answer” != ”y” ]];
        then
47          exit
```

```
48 fi
49
50 sudo xhost +local:docker
51
52 docker build −−tag $name_work_image −−build−arg image=$name_base_image −−file ./
       $DockerFile .
53
54 docker run −it −−name $name_work_container −e DISPLAY=unix$DISPLAY −v /tmp/.X11−
       unix:/tmp/.X11−unix −−privileged $name_work_image /bin/bash
```

2.6.6 Удаление Docker образа, Docker контейнера и кэш у Docker

Название скрипта: docker-delete-container-image.sh

Цель: удалить Docker контейнеры, Docker образы и Docker кэш.

Результат: очистка памяти ПК.

Алгоритм работы скрипта представлен на рисунке 2.4

Примечание:

Частая ошибка при удалении: ID у копий образов всегда одинаковый, и Docker не знает какой именно удалить.

Решение: Прописать вместо ID его название. Это решение касается не только образов (название с тегом), но и контейнеров (название без тега).

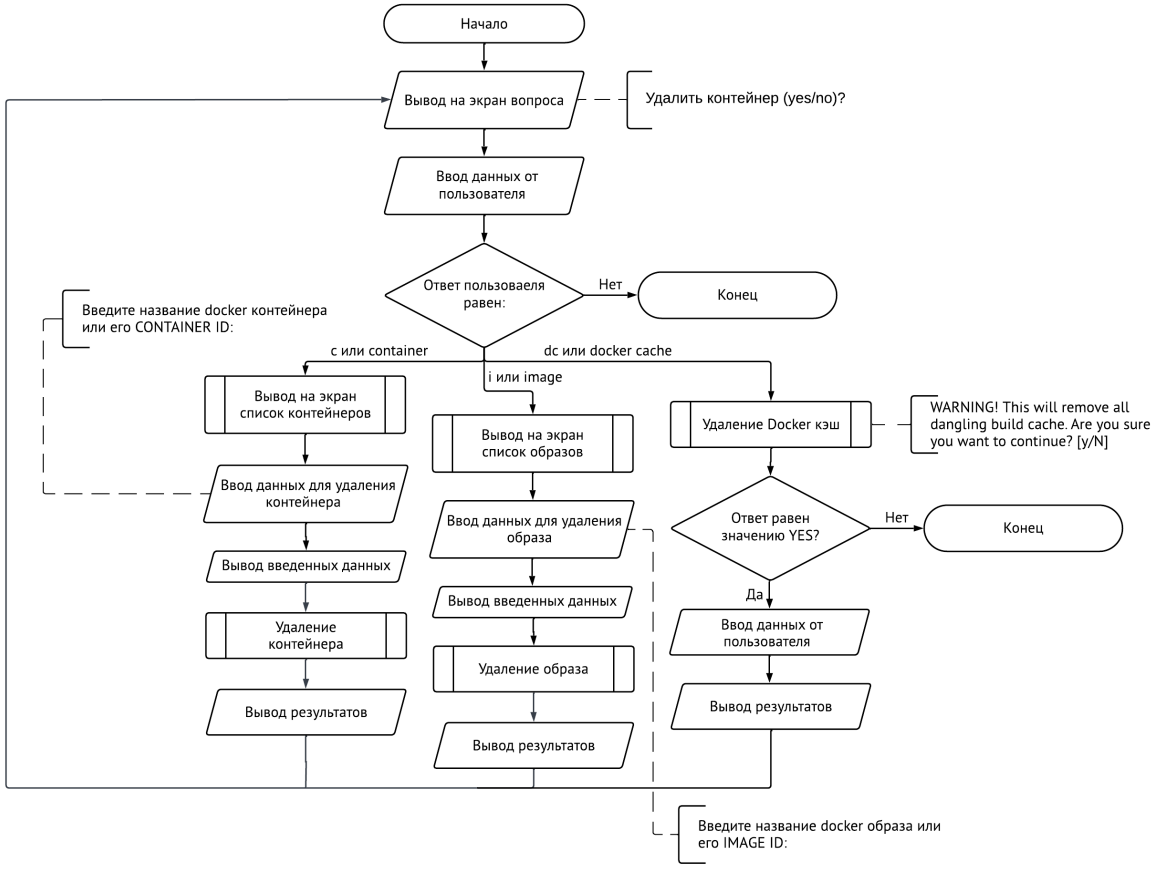

Рисунок 2.4 — Алгоритм работы блока ”удаление контейнера”скрипта docker-delete-container-image.sh

```
1  #!/bin/bash
2
3  answer=”i”
4
5  while [[ ”$answer” == ”container” || ”$answer” == ”c” || ”$answer” == ”image” || ”$answer” == ”
       i” || ”$answer” == ”docker−cache” || ”$answer” == ”dc” ]];
6  do
7          read −p ”Что нужно удалить (container(c)/image(i)/docker−cache(dc)/quit(q)): ”
                   answer
8
9          if [[ ”$answer” == ”docker−cache” || ”$answer” == ”dc” ]]; then
10                 docker builder prune
11         fi
12
13         if [[ ”$answer” == ”image” || ”$answer” == ”i” ]]; then
14                 docker images
```

```
15                  read −p ”Введите название docker образа или его IMAGE ID: ”
                        name_work_image
16                  echo ”Было введено:”
17                  echo ”1) имя docker контейнера: ” $name_work_image
18                  docker rmi $name_work_image
19          fi
20
21          if [[ ”$answer” == ”container” || ”$answer” == ”c” ]]; then
22                  docker ps −a
23                  read −p ”Введите название docker контейнера или его CONTAINER ID: ”
                        name_work_container
24                  echo ”Было введено:”
25                  echo ”1) имя docker контейнера или CONTAINER ID: ”
                        $name_work_container
26                  docker stop $name_work_container
27                  docker rm $name_work_container
28          fi
29  done
30  exit
```

2.6.7 Запуск созданного контейнера

Название скрипта: docker-open-container.sh

Цель: зайти в созданный нами контейнер.

Результат: командная строка внутри созданного контейнера.

Алгоритм работы скрипта представлен на рисунке 2.5.

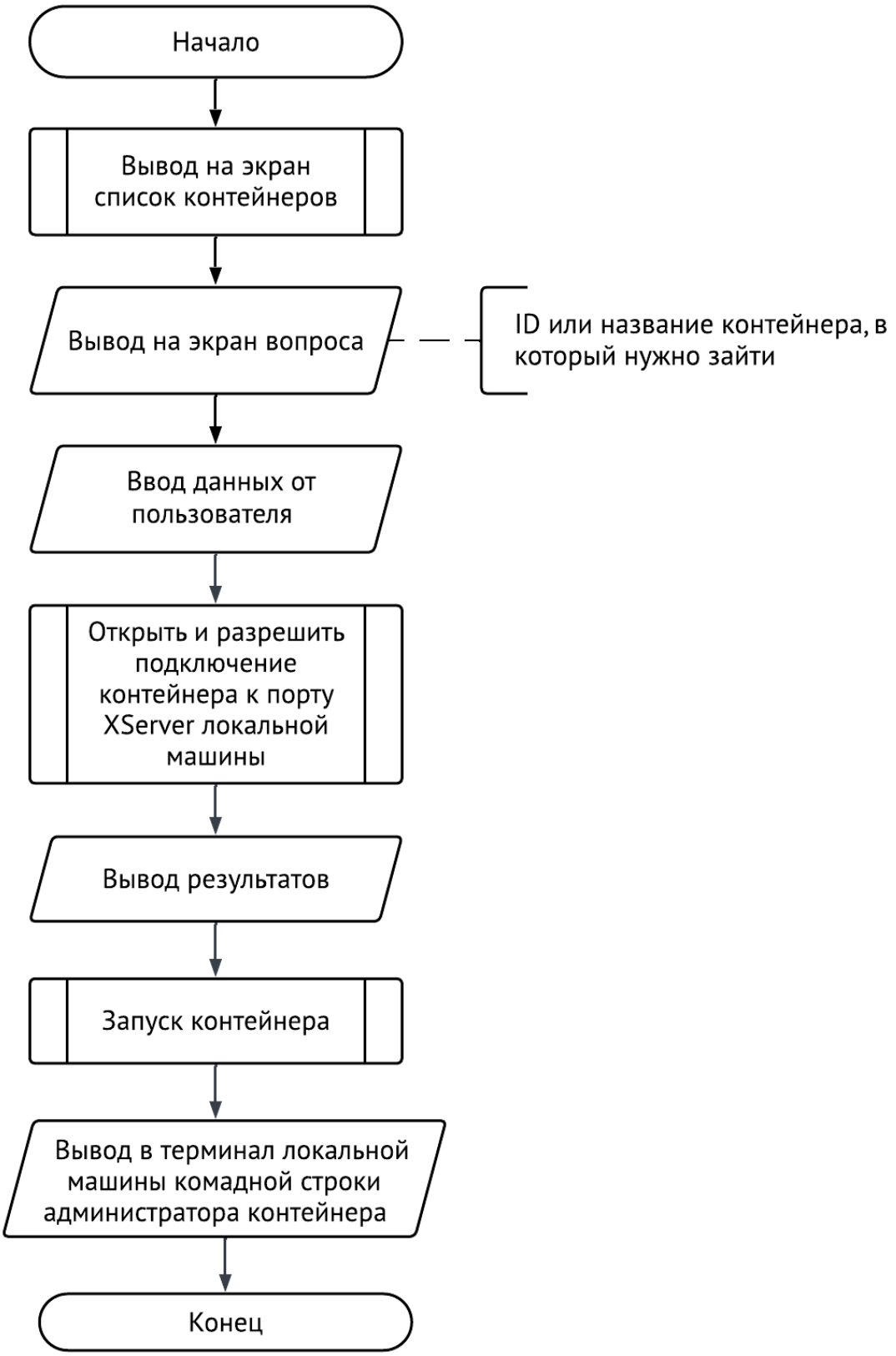

Рисунок 2.5 — Алгоритм работы скрипта docker-open-container.sh

```
1 #!/bin/bash
2
```

```
3  docker ps −a
4
5  read −p ”ID или название контейнера, в который нужно зайти: ” name_container
6  echo ”имя docker контейнера: ” $name_container
7
8  sudo xhost +local:docker
9
10 docker start $name_container
11
12 docker exec −it −e DISPLAY=unix$DISPLAY $name_container /bin/bash
```

2.6.8 Отправление Docker образа или Docker контейнера на репозиторий Harbor

Название скрипта: docker-push.sh

Цель: зайти в созданный нами контейнер.

Результат: командная строка внутри созданного контейнера.

Алгоритм работы скрипта представлен на рисунках 2.6 и 2.7.

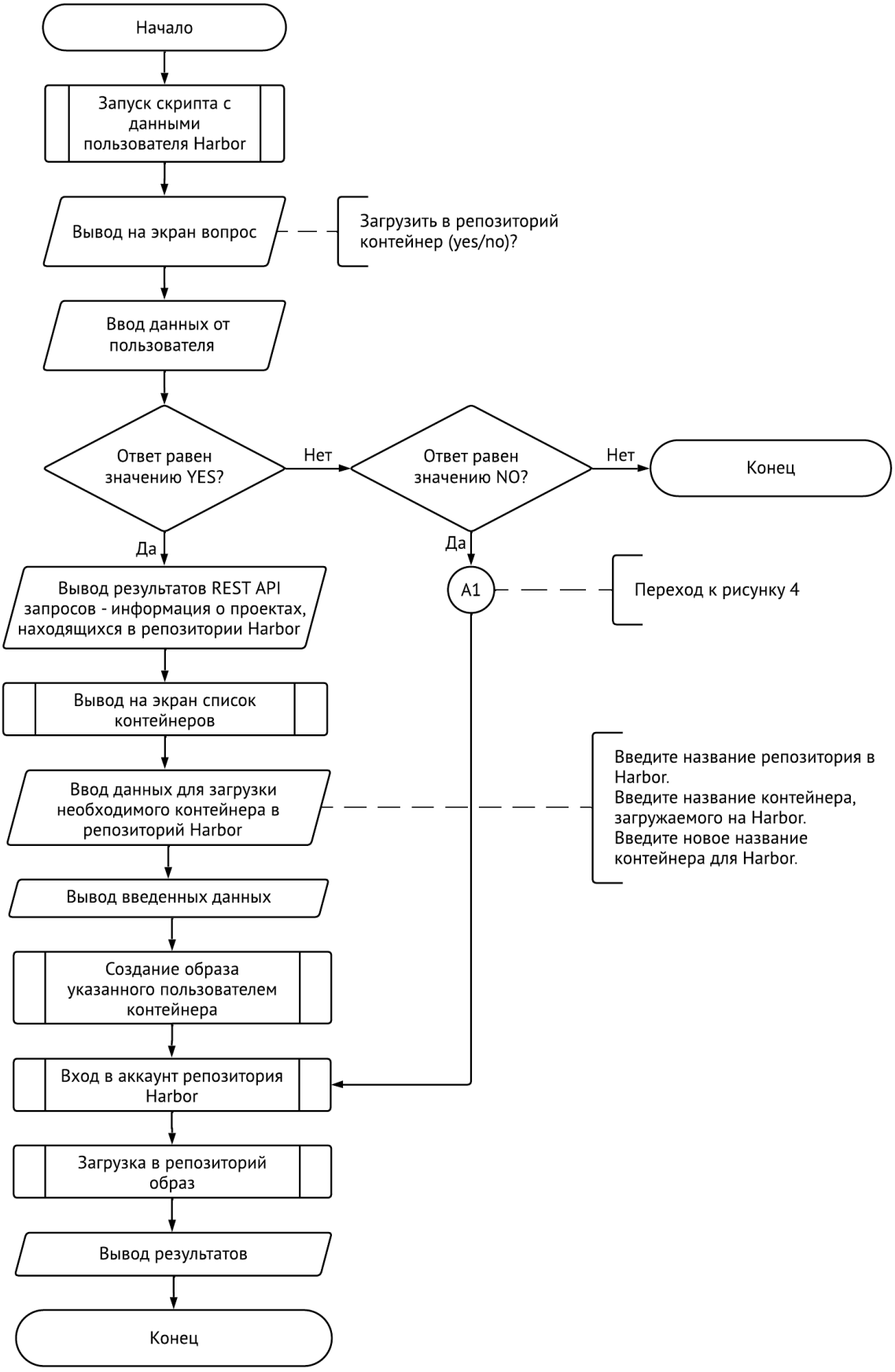

Рисунок 2.6 — Алгоритм работы блока ”Загрузка контейнера в репозиторий Harbor”у скрипта docker-push.sh

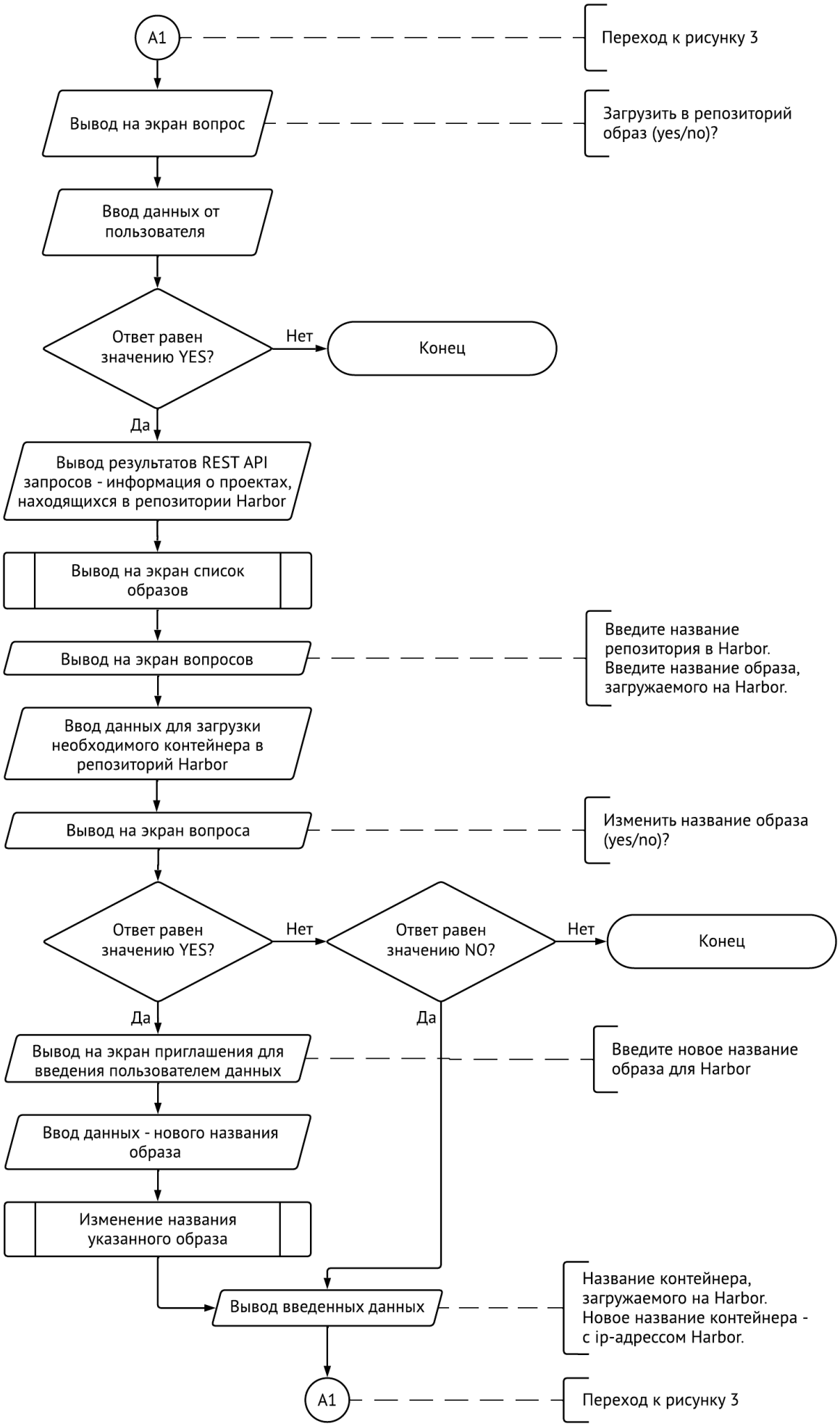

Рисунок 2.7 — Алгоритм работы ”Загрузка образа в репозиторий Harbor”у

```
1  #!/bin/bash
2
3  . ./info−account.sh
4
5  read −p ”Загрузить в репозиторий контейнер (yes/no)? ” answer
6
7  if [[ ”$answer” == ”yes” || ”$answer” == ”y” ]]; then
8          echo ”Вывод репозиториев с проектами: ”
9          curl −u $USERNAME:$PASSWORD −k https://$REGISTRY_URL/v2/_catalog
10         read −p ”Введите название репозитория в Harbor: ” name_repo
11         docker ps −a
12         read −p ”Введите название контейнера, загружаемого на Harbor: ”
               name_work_container
13         read −p ”Введите новое название контейнера для Harbor: ”
               new_name_work_container
14         read −p ”Введите тег для $new_name_work_container контейнера: ”
               new_name_work_container_tag
15         var1=”${REGISTRY_URL}/”
16         var2=”$var1$name_repo”
17         var3=”${var2}/”
18         var4=”$var3$new_name_work_container”
19         var5=”${var4}:”
20         name_push_image=”$var5$new_name_work_container_tag”
21         echo ”Было введено:”
22         echo ”1) название контейнера, загружаемого на Harbor: ” $name_work_container
23         echo ”2) новое название контейнера − с ip адрессом Harbor: ” $name_push_image
24
25         docker commit $name_work_container $name_push_image
26 fi
27
28 if [[ ”$answer” == ”no” || ”$answer” == ”n” ]]; then
29         read −p ”Загрузить в репозиторий образ (yes/no)? ” answer
30
31         if [[ ”$answer” != ”yes” && ”$answer” != ”y” ]]; then
32         exit
33         fi
34
35         if [[ ”$answer” == ”yes” || ”$answer” == ”y” ]]; then
36                 echo ”Вывод репозиториев с проектами: ”
```

```
37                  curl −u $USERNAME:$PASSWORD −k https://$REGISTRY_URL/v2/
                        _catalog
38                  read −p ”Введите название репозитория в Harbor: ” name_repo
39                  docker images
40                  read −p ”Введите название образа, загружаемого на Harbor: ” name_image
41
42                  read −p ”Изменить название образа (yes/no)? ” answer
43                  if [[ ”$answer” == ”yes” || ”$answer” == ”y” ]]; then
44                          read −p ”Введите новое название образа для Harbor: ”
                                new_name_image
45                          read −p ”Введите тег для $new_name_image образа: ”
                                new_name_image_tag
46                          var1=”${REGISTRY_URL}/”
47                          var2=”$var1$name_repo”
48                          var3=”${var2}/”
49                          var4=”$var3$new_name_image”
50                          var5=”${var4}:”
51                          name_push_image=”$var5$new_name_image_tag”
52                          docker tag $name_image $name_push_image
53                  fi
54
55                  if [[ ”$answer” == ”no” || ”$answer” == ”n” ]]; then
56                          name_push_image=$name_image
57                          echo ”ok”
58                  fi
59
60                  if [[ ”$answer” != ”no” && ”$answer” != ”n” && ”$answer” != ”yes” && ”
                        $answer” != ”y” ]]; then
61                          exit
62                  fi
63          fi
64
65          echo ”Было введено:”
66          echo ”1) название образа, загружаемого на Harbor: ” $name_image
67          echo ”2) новое название образа − с ip адрессом Harbor: ” $name_push_image
68  fi
69
70  if [[ ”$answer” != ”no” && ”$answer” != ”n” && ”$answer” != ”yes” && ”$answer” != ”y” ]];
        then
71          exit
```

```
fi

docker login $REGISTRY_URL −u $USERNAME −p $PASSWORD

docker push $name_push_image
```

2.6.9 Преобразование Docker контейнера в Docker образ

Название скрипта: docker-container-to-image.sh

Цель: преобразовать Docker контейнер в образ, для дальнейшей работы с образом данного контейнера (скачать tar архив или создать новый контейнер на базе данного образа).

Результат: Скрипт выведет на экран название контейнера и контрольную сумму у нового образа.

```
#!/bin/bash

docker ps −a
read −p ”Введите название контейнера, из которого нужно получить образ: ”
    name_work_container
read −p ”Введите название создаваемого образа: ” new_name_work_container
read −p ”Введите тег для образа $new_name_work_container: ”
    new_name_work_container_tag

if [[ $new_name_work_container_tag == ”” ]]; then
        new_name_work_container_tag=”latest”
fi

var1=”$new_name_work_container”
var2=”${var1}:”
name_image=”$var2$new_name_work_container_tag”

docker stop $name_work_container
docker commit $name_work_container $name_image
```

2.6.10 Проверка контрольных сумм у Docker образов

Название скрипта: test-is-equal-docker-images.sh

Цель: проверить образы на соответствие контрольных сумм.

Результат: вывод строки — совпадение/не совпадение.

Алгоритм работы скрипта представлен на рисунке 2.8

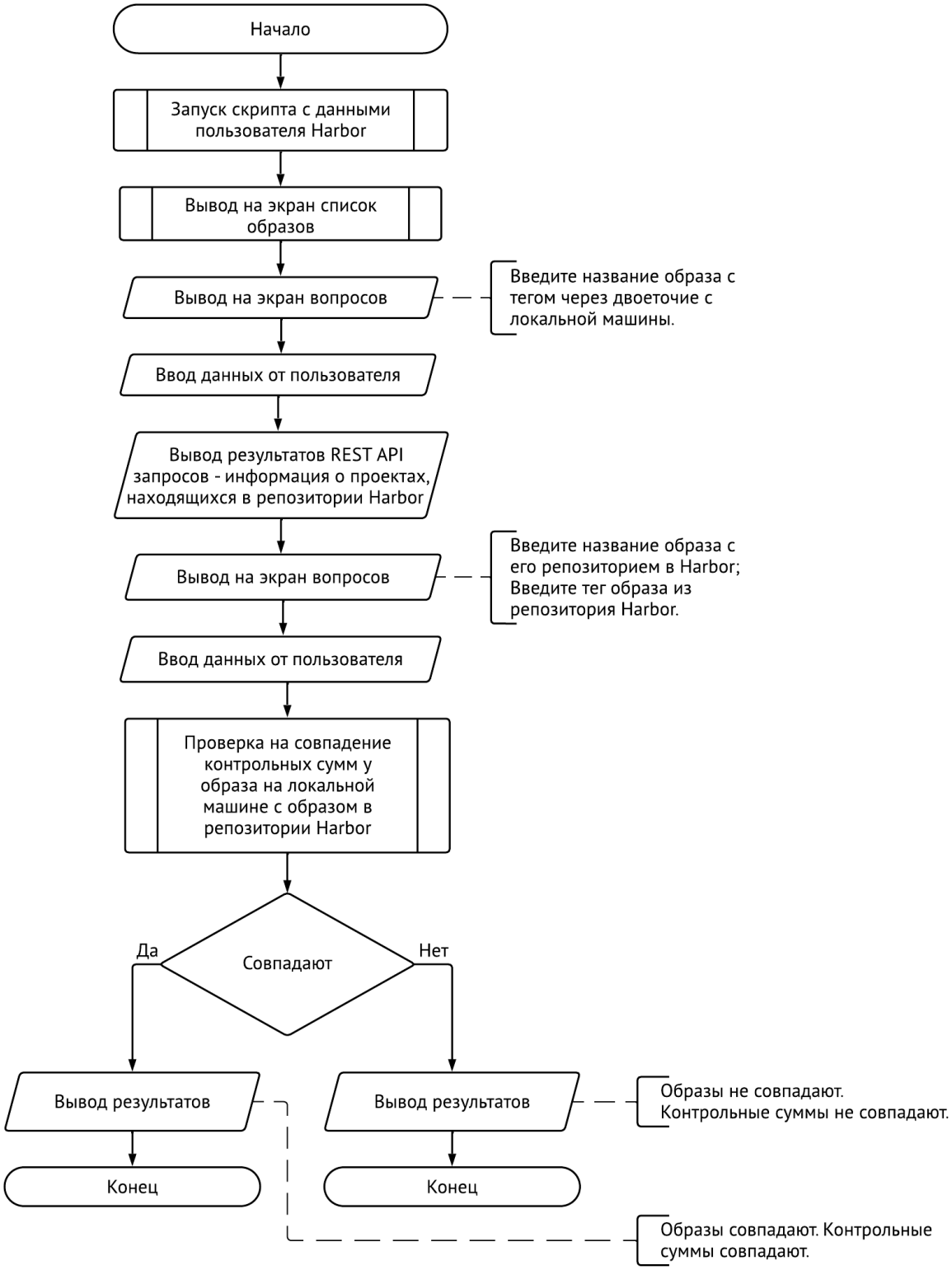

Рисунок 2.8 — Алгоритм работы скрипта test-is-equal-docker-images.sh

```
1 #!/bin/bash
2
3 . ./info−account.sh
4
```

```
docker images
read −p ”Введите название образа с тегом через двоеточие с локальной машины: ” name_local_image

curl −u $USERNAME:$PASSWORD −k https://$REGISTRY_URL/v2/_catalog
read −p ”Введите название образа с его репозиторием в Harbor: ” name_repo_img

curl −u $USERNAME:$PASSWORD −k https://$REGISTRY_URL/v2/$name_repo_img/tags/list
read −p ”Введите тег образа $name_repo_img из репозитория Harbor: ” name_tag

var1=”${REGISTRY_URL}/”
var2=”$var1$name_repo_img”
var3=”${var2}:”
name_repo_image=”$var3$name_tag”

echo ”Было введено:”
echo ”1) имя базового докер образа: ” $name_local_image
echo ”2) имя docker образа из репозитория Harbor: ” $name_repo_image

docker image inspect −−format ”{{.RepoDigests}}” $name_local_image > ./all−info.txt
line_1=$(cat ./all−info.txt)
digests=$(echo ${line_1#*@})
digests_local_image=${digests::−1}

curl −u $USERNAME:$PASSWORD −i −k https://$REGISTRY_URL/v2/$name_repo_img/manifests/$name_tag > ./all−info.txt
head −n7 ./all−info.txt > ./digests.txt
line=$(grep ’Docker−Content−Digest: ’ ./digests.txt)
digests_image=$(echo ${line#*Docker−Content−Digest: })
digests_repo_image=${digests_image::−1}

rm ./all−info.txt ./digests.txt

if [[ ”$digests_repo_image” == ”$digests_local_image” ]]; then
echo ”Образы совпадают. Контрольные суммы совпадают.”
else
echo ”Образы не совпадают. Контрольные суммы не совпадают.”
fi
```