**Шаг 1 — Установка Docker**

Дистрибутив Docker, доступный в официальном репозитории Ubuntu, не всегда является последней версией программы. Лучше установить последнюю версию Docker, загрузив ее из официального репозитория Docker. Для этого добавляем новый источник дистрибутива, вводим ключ GPG из репозитория Docker, чтобы убедиться, действительна ли загруженная версия, а затем устанавливаем дистрибутив.

Сначала обновляем существующий перечень пакетов:

* sudo apt update

Затем устанавливаем необходимые пакеты, которые позволяют apt использовать пакеты по HTTPS:

* sudo apt install apt-transport-https ca-certificates curl software-properties-common

Затем добавляем в свою систему ключ GPG официального репозитория Docker:

* curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Добавляем репозиторий Docker в список источников пакетов APT:

* sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

Затем обновим базу данных пакетов информацией о пакетах Docker из вновь добавленного репозитория:

* sudo apt update

Следует убедиться, что мы устанавливаем Docker из репозитория Docker, а не из репозитория по умолчанию Ubuntu:

* apt-cache policy docker-ce

Вывод получится приблизительно следующий. Номер версии Docker может быть иным:

Output of apt-cache policy docker-ce

docker-ce:

Installed: (none)

Candidate: 18.03.1~ce~3-0~ubuntu

Version table:

18.03.1~ce~3-0~ubuntu 500

500 https://download.docker.com/linux/ubuntu bionic/stable amd64 Packages

Обратите внимание, что docker-ce не устанавливается, но для установки будет использован репозиторий Docker для Ubuntu 18.04 (bionic).

Далее устанавливаем Docker:

* sudo apt install docker-ce

Теперь Docker установлен, демон запущен, и процесс будет запускаться при загрузке системы.  Убедимся, что процесс запущен:

* sudo systemctl status docker

Вывод должен быть похож на представленный ниже, сервис должен быть запущен и активен:

Output

● docker.service - Docker Application Container Engine

Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)

Active: active (running) since Thu 2018-07-05 15:08:39 UTC; 2min 55s ago

Docs: https://docs.docker.com

Main PID: 10096 (dockerd)

Tasks: 16

CGroup: /system.slice/docker.service

├─10096 /usr/bin/dockerd -H fd://

└─10113 docker-containerd --config /var/run/docker/containerd/containerd.toml

При установке Docker мы получаем не только сервис (демон) Docker, но и утилиту командной строки docker или клиент Docker. Использование утилиты командной строки docker рассмотрено ниже.

**Шаг 2 — Использование команды Docker без sudo (опционально)**

По умолчанию, запуск команды docker требует привилегий пользователя **root** или пользователя группы **docker**, которая автоматически создается при установке Docker. При попытке запуска команды docker пользователем без привилегий sudo или пользователем, не входящим в группу **docker**, выводные данные будут выглядеть следующим образом:

Output

docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.

See 'docker run --help'.

Чтобы не вводить sudo каждый раз при запуске команды docker, добавьте имя своего пользователя в группу docker:

* sudo usermod -aG docker ${USER}

Для применения этих изменений в составе группы необходимо разлогиниться и снова залогиниться на сервере или задать следующую команду:

* su - ${USER}

Для продолжения работы необходимо ввести пароль пользователя.

Убедиться, что пользователь добавлен в группу **docker** можно следующим образом:

* id -nG

Output

sammy sudo docker

Если вы хотите добавить произвольного пользователя в группу docker, можно указать конкретное имя пользователя:

* sudo usermod -aG docker username

Далее в этой статье предполагается, что вы используете команду docker как пользователь, находящийся в группе **docker**. Если вы не хотите добавлять своего пользователя в группу docker, в начало команд необходимо добавлять sudo.

Теперь рассмотрим команду docker.

**Шаг 3 — Использование команды Docker**

Команда docker позволяет использовать различные опции, команды с аргументами. Синтаксис выглядит следующим образом:

* docker [option] [command] [arguments]

Для просмотра всех доступных подкоманд введите:

* docker

Полный список подкоманд Docker 18:

Output

attach Attach local standard input, output, and error streams to a running container

build Build an image from a Dockerfile

commit Create a new image from a container's changes

cp Copy files/folders between a container and the local filesystem

create Create a new container

diff Inspect changes to files or directories on a container's filesystem

events Get real time events from the server

exec Run a command in a running container

export Export a container's filesystem as a tar archive

history Show the history of an image

images List images

import Import the contents from a tarball to create a filesystem image

info Display system-wide information

inspect Return low-level information on Docker objects

kill Kill one or more running containers

load Load an image from a tar archive or STDIN

login Log in to a Docker registry

logout Log out from a Docker registry

logs Fetch the logs of a container

pause Pause all processes within one or more containers

port List port mappings or a specific mapping for the container

ps List containers

pull Pull an image or a repository from a registry

push Push an image or a repository to a registry

rename Rename a container

restart Restart one or more containers

rm Remove one or more containers

rmi Remove one or more images

run Run a command in a new container

save Save one or more images to a tar archive (streamed to STDOUT by default)

search Search the Docker Hub for images

start Start one or more stopped containers

stats Display a live stream of container(s) resource usage statistics

stop Stop one or more running containers

tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

top Display the running processes of a container

unpause Unpause all processes within one or more containers

update Update configuration of one or more containers

version Show the Docker version information

wait Block until one or more containers stop, then print their exit codes

Для просмотра опций использования определенной команды введите:

* docker docker-subcommand --help

Для просмотра всей информации о Docker используется следующая команда:

* docker info

Рассмотрим некоторые команды подробнее. Расскажем сначала о работе с образами.

**Шаг 4 — Работа с образами Docker**

Контейнеры Docker запускаются из образов Docker. По умолчанию Docker получает образы из хаба [Docker Hub](https://hub.docker.com), представляющего собой реестр образов, который поддерживается компанией Docker. Кто угодно может создать и загрузить свои образы Docker в Docker Hub, поэтому для большинства приложений и дистрибутивов Linux, которые могут потребоваться вам для работы, уже есть соответствующие образы в Docker Hub.

Чтобы проверить, можете ли вы осуществлять доступ и загружать образы из Docker Hub, введите следующую команду:

* docker run hello-world

Корректный результат работы этой команды, который означает, что Docker работает правильно, представлен ниже:

Output

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

9bb5a5d4561a: Pull complete

Digest: sha256:3e1764d0f546ceac4565547df2ac4907fe46f007ea229fd7ef2718514bcec35d

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

...

Изначально Docker не мог находить образ hello-world локально, поэтому загружал образ из Docker Hub, который является репозиторием по умолчанию. После загрузки образа Docker создавал из образа контейнер и запускал приложение в контейнере, отображая сообщение.

Образы, доступные в Docker Hub, можно искать с помощью команды docker и подкоманды search. Например, для поиска образа Ubuntu вводим:

* docker search ubuntu

Скрипт просматривает Docker Hub и возвращает список всех образов, имена которых подходят под заданный поиск. В данном случае мы получим примерно следующий результат:



В столбце **OFFICIAL** строка **OK** показывает, что образ построен и поддерживается компанией, которая занимается разработкой этого проекта. Когда нужный образ выбран, можно загрузить его на ваш компьютер с помощью подкоманды pull.

Чтобы загрузить официальный образ ubuntu на свой компьютер, запускается следующая команда:

* docker pull ubuntu

Результат будет выглядеть следующим образом:

Output

Using default tag: latest

latest: Pulling from library/ubuntu

6b98dfc16071: Pull complete

4001a1209541: Pull complete

6319fc68c576: Pull complete

b24603670dc3: Pull complete

97f170c87c6f: Pull complete

Digest: sha256:5f4bdc3467537cbbe563e80db2c3ec95d548a9145d64453b06939c4592d67b6d

Status: Downloaded newer image for ubuntu:latest

После загрузки образа можно запустить контейнер с загруженным образом с помощью подкоманды run.  Как видно из примера hello-world, если при выполнении docker с помощью подкоманды run образ еще не загружен, клиент Docker сначала загрузит образ, а затем запустит контейнер с этим образом.

Для просмотра загруженных на компьютер образов нужно ввести:

* docker images

Вывод должен быть похож на представленный ниже:

Output

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu latest 113a43faa138 4 weeks ago 81.2MB

hello-world latest e38bc07ac18e 2 months ago 1.85kB

Далее в инструкции показано, что образы, используемые для запуска контейнеров, можно изменять и применять для создания новых образов, которые, в свою очередь, могут быть загружены (технический термин *push*) в Docker Hub или другой Docker-реестр.

Рассмотрим более подробно, как запускать контейнеры.

**Шаг 5 — Запуск контейнера Docker**

Контейнер hello-world, запущенный на предыдущем этапе, является примером контейнера, который запускается и завершает работу после вывода тестового сообщения. Контейнеры могут выполнять и более полезные действия, а также могут быть интерактивными. Контейнеры похожи на виртуальные машины, но являются менее требовательными к ресурсам.

В качестве примера запустим контейнер с помощью последней версии образа Ubuntu. Комбинация параметров **-i** и **-t** обеспечивает интерактивный доступ к командному процессору контейнера:

* docker run -it ubuntu

Командная строка должна измениться, показывая, что мы теперь работаем в контейнере. Она будет иметь следующий вид:

Output

root@d9b100f2f636:/#

Обратите внимание, что в командной строке отображается идентификатор контейнера. В данном примере это d9b100f2f636. Идентификатор контейнера потребуется нам позднее, чтобы указать, какой контейнер необходимо удалить.

Теперь можно запускать любые команды внутри контейнера. Попробуем, например, обновить базу данных пакета внутри контейнера. Здесь перед командами не нужно использовать sudo, поскольку вы работаете внутри контейнера как пользователь с привилегиями **root**:

* apt update

Теперь в нем можно установить любое приложение. Попробуем установить Node.js:

* apt install nodejs

Данная команда устанавливает Node.js в контейнер из официального репозитория Ubuntu. Когда установка завершена, убедимся, что Node.js установлен:

* node -v

В терминале появится номер версии:

Output

v8.10.0

Все изменения, которые вы производите внутри контейнера, применяются только для этого контейнера.

Чтобы выйти из контейнера, вводим команду exit.

Далее рассмотрим, как управлять контейнерами в своей системе.

**Шаг 6 — Управление контейнерами Docker**

Через некоторое время после начала использования Docker на вашей машине будет множество активных (запущенных) и неактивных контейнеров. Просмотр \*\* активных контейнеров \*\*:

* docker ps

Результат получится примерно следующим:

Output

CONTAINER ID IMAGE COMMAND CREATED

По нашей инструкции вы запустили два контейнера: один из образа hello-world, второй из образа ubuntu. Оба контейнера уже не запущены, но существуют в системе.

Чтобы увидеть и активные, и неактивные контейнеры, запускаем docker ps с помощью параметра -a:

* docker ps -a

Результат получится примерно следующим:

d9b100f2f636 ubuntu "/bin/bash" About an hour ago Exited (0) 8 minutes ago sharp\_volhard

01c950718166 hello-world "/hello" About an hour ago Exited (0) About an hour ago festive\_williams

Чтобы увидеть последние из созданных контейнеров, задаем параметр -l:

* docker ps -l

* CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
* d9b100f2f636 ubuntu "/bin/bash" About an hour ago Exited (0) 10 minutes ago sharp\_volhard

Для запуска остановленного контейнера используем команду docker start, затем указываем идентификатор контейнера или его имя. Запустим загруженный из Ubuntu контейнер с идентификатором d9b100f2f636:

* docker start d9b100f2f636

Контейнер запускается. Теперь для просмотра его статуса можно использовать docker ps:

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

d9b100f2f636 ubuntu "/bin/bash" About an hour ago Up 8 seconds sharp\_volhard

Для остановки запущенного контейнера используем команду docker stop, затем указываем идентификатор контейнера или его имя. В этот раз мы используем имя, которое назначил контейнеру Docker, то есть sharp\_volhard:

* docker stop sharp\_volhard

Если вам контейнер больше не нужен, удаляем его командой docker rm с указанием либо идентификатора, либо имени контейнера. Чтобы найти идентификатор или имя контейнера, связанного с образом hello-world, используйте команду docker ps -a. Затем контейнер можно удалить.

* docker rm festive\_williams

Запустить новый контейнер и задать ему имя можно с помощью параметра --name. Параметр --rm позволяет создать контейнер, который самостоятельно удалится после остановки. Для более подробной информации о данных и других опциях используйте команду docker run help.

Контейнеры можно превратить в образы для построения новых контейнеров. Рассмотрим, как это сделать.

**Шаг 7 — Сохранение изменений в контейнере в образ Docker**

При запуске контейнера из образа Docker вы можете создавать, изменять и удалять файлы, как и на виртуальной машине.  Внесенные изменения применяются только для такого контейнера. Можно запускать и останавливать контейнер, однако как только он будет уничтожен командой docker rm, все изменения будут безвозвратно потеряны.

В данном разделе показано, как сохранить состояние контейнера в виде нового образа Docker.

После установки Node.js в контейнере Ubuntu у вас будет работать запущенный из образа контейнер, но он будет отличаться от образа, использованного для его создания. Однако вам может потребоваться такой контейнер Node.js как основа для будущих образов.

Затем подтверждаем изменения в новом образе Docker с помощью следующей команды.

* docker commit -m "What you did to the image" -a "Author Name" container\_id repository/new\_image\_name

Параметр **-m** позволяет задать сообщение подтверждения, чтобы облегчить вам и другим пользователям образа понимание того, какие изменения были внесены, а параметр **-a** позволяет указать автора. Идентификатор контейнера container\_id — этот тот самый идентификатор, который использовался ранее, когда мы начинали интерактивную сессию в контейнере Docker. Если вы не создавали дополнительных репозиториев в Docker Hub, имя репозитория (repository) обычно является вашим именем пользователя в Docker Hub.

Например, для пользователя **sammy** и идентификатора контейнера d9b100f2f636 команда будет выглядеть следующим образом:

* docker commit -m "added Node.js" -a "sammy" d9b100f2f636 sammy/ubuntu-nodejs

После подтверждения (*commit*) образа, новый образ сохраняется локально на вашем компьютере. Далее в этой инструкции мы расскажем, как отправить образ в реестр Docker (например, в Docker Hub) так, чтобы он был доступен не только вам, но и другим пользователям.

Если теперь просмотреть список образов Docker, в нем окажутся и новый образ, и исходный образ, на котором он был основан:

* docker images

Результат получится примерно следующим:

Output

REPOSITORY TAG IMAGE ID CREATED SIZE

sammy/ubuntu-nodejs latest 7c1f35226ca6 7 seconds ago 179MB

ubuntu latest 113a43faa138 4 weeks ago 81.2MB

hello-world latest e38bc07ac18e 2 months ago 1.85kB

В данном примере ubuntu-nodejs — это новый образ, созданный на основе существующего образа ubuntu из Docker Hub. Разница размеров отражает внесенные изменения. В данном примере изменение связано с установкой NodeJS. В следующий раз, когда потребуется запустить контейнер Ubuntu с предустановленным NodeJS, можно использовать этот новый образ.

Образы также могут строиться с помощью файла Dockerfile, который позволяет автоматизировать установку программ в новом образе. Однако в данной статье этот процесс не описывается.

Давайте теперь поделимся новым образом с другими пользователями, чтобы они могли создавать на его основе контейнеры.

**Шаг 8 — Отправка контейнеров Docker в репозиторий Docker**

Следующим логичным шагом после создания нового образа из существующего будет поделиться созданным образом с друзьями, со всеми в Docker Hub или в другом реестре Docker, к которому у вас есть доступ. Для отправки образов в Docker Hub или другой Docker-реестр, у вас должна быть в нем учетная запись.

В данном разделе показано, как отправлять образы Docker в Docker Hub. Научиться создавать собственный Docker-реестр можно с помощью статьи [How To Set Up a Private Docker Registry on Ubuntu 14.04](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-14-04).

Чтобы отправить свой образ, осуществляем вход на Docker Hub.

* docker login -u docker-registry-username

Для входа требуется ввести пароль Docker Hub. Если введен правильный пароль, вы будете успешно авторизованы.

**Примечание:** Если имя пользователя в Docker-реестре отличается от локального имени пользователя, которое использовалось для создания образа, необходимо привязать свой образ к имени пользователя в реестре. Чтобы отправить пример из предыдущего шага, вводим:

* docker tag sammy/ubuntu-nodejs docker-registry-username/ubuntu-nodejs

Затем можно отправлять собственный образ:

* docker push docker-registry-username/docker-image-name

Команда для отправки образа ubuntu-nodejs в репозиторий **sammy** выглядит следующим образом:

* docker push sammy/ubuntu-nodejs

Для загрузки образа может потребоваться некоторое время, но после завершения результат будет выглядеть следующим образом:

Output

The push refers to a repository [docker.io/sammy/ubuntu-nodejs]

e3fbbfb44187: Pushed

5f70bf18a086: Pushed

a3b5c80a4eba: Pushed

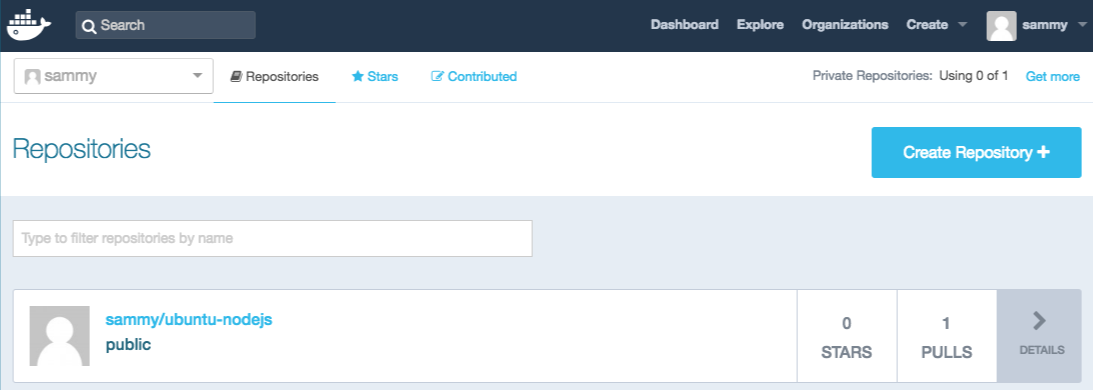
7f18b442972b: Pushed

3ce512daaf78: Pushed

7aae4540b42d: Pushed

...

После отправки образа в реестр, его имя должно появиться в списке панели управления вашей учетной записи, как показано ниже.



Если при отправке появляется ошибка, как показано ниже, значит, не выполнен вход в реестр:

Output

The push refers to a repository [docker.io/sammy/ubuntu-nodejs]

e3fbbfb44187: Preparing

5f70bf18a086: Preparing

a3b5c80a4eba: Preparing

7f18b442972b: Preparing

3ce512daaf78: Preparing

7aae4540b42d: Waiting

unauthorized: authentication required

Авторизуемся в реестре с помощью docker login и снова пытаемся отправить образ. Затем убедимся, что он появился на вашей странице в репозитории Docker Hub.

Теперь с помощью команды docker pull sammy/ubuntu-nodejs можно загрузить образ на новую машину и использовать его для запуска нового контейнера.