Лабораторна робота № 8

Установка Docker

Мета роботи - набути навичок встановлювати додаток Docker в ОС Linux, надати основи для роботи на постійній основі з образами та контейнерами, що дозволяє не засмічувати робочу машину локально встановленими різними версіями низки програмного забезпечення: *apache, mysql, virtualenv, python, mongodb, memchaced, redis, php,* а також подібного програмного забезпечення, яке використовується при розробці проектів та часто ще й конфліктує між собою від версії до версії.

Теоретичні відомості

Docker є найпопулярнішою платформою управління контейнерами. Це програмне забезпечення з відкритим кодом, принцип роботи якого найпростіше порівняти з транспортними контейнерами. Філософію Docker часто описують за допомогою метафори «доставки універсальних вантажних контейнерів», тобто стандартизованих розмірів контейнерів, які можна переміщувати між різними видами транспорту (вантажівками, поїздами, кораблями) з мінімумом ручної праці. Така ідея була перенесена на IT-сферу для переміщення коду між різними програмними середовищами з мінімальними обсягами роботи. Коли розробляється додаток, необхідно надати код разом з усіма його складовими, такими як бібліотеки, сервер, бази даних і т. д. Може мати місце така ситуація, коли додаток працює на вашому комп'ютері, але відмовляється працювати на комп’ютері іншого користувача. Ця проблема вирішується через створення програмного забезпечення, яке не залежить від системи.

Саме контейнери Docker спрощують перенесення програмних додатків.

Термінологія

Контейнери - це *технологія упаковки і запуску додатків* Windows, Linux, MacOS в різних локальних середовищах і в хмарі.

Контейнер - *це виконуваний екземпляр, який інкапсулює необхідну програмне забезпечення. Він складається з образів.* Його можна легко видалити і знову створити за короткий проміжок часу. Контейнери надають невимогливе до ресурсів ізольоване середовище, яке спрощує розробку, розгортання, запуск програмного забезпечення, особливо в динамічних і розподілених середовищах та керування додатками.

Образ - *базовий елемент кожного контейнера*. Залежно від способу, може знадобитися деякий час для його створення.

Порт - це *порт TCP/UDP* (протоколи транспортного рівня для передачі пакетів між комп’ютерами) в своєму первинному значенні. Щоб все було просто, припустимо, що порти можуть бути відкриті в зовнішньому світі або підключені до контейнерів (доступні тільки з цих контейнерів і невидимі для зовнішнього світу).

Том - описується як *загальна папка*. Тома ініціалізуються при створенні контейнера і призначені для збереження даних, незалежно від життєвого циклу контейнера.

Реєстр - це *сервер, на якому зберігаються образи*. Порівняємо його з GitHub: ви можете витягнути образ з реєстру, щоб розгорнути його локально, і так само локально можете вносити в реєстр створені образи.

Docker Hub - *публічний репозиторій* з інтерфейсом, що надається Docker Inc. Він *зберігає безліч образів*. Ресурс є джерелом «офіційних» образів, зроблених командою Докер або створених у співпраці з розробником програмного забезпечення. Для офіційних образів перераховані їх потенційні уразливості. Ця інформація відкрита для будь-якого зареєстрованого користувача. Доступні як безкоштовні, так і платні акаунти.

Контейнери створюються на основі ядра операційної системи сервера, але не отримують необмежений доступ до ядра. Наприклад, контейнер може звертатися до віртуалізованої версії файлової системи і реєстру, але будь-які зміни стосуються тільки контейнера і видаляються при його зупинці. Контейнер збирається поверх ядра, але ядро не надає всі інтерфейси API і служби, необхідні для запуску програми. Більшість з них надаються системними файлами (бібліотеками), які працюють на рівні вище ядра в режимі користувача. Оскільки контейнер ізольований від середовища режиму користувача сервера, контейнеру потрібно власна копія цих системних файлів режиму користувача, які упаковуються в базовий образ. Базовий образ виступає в якості основного рівня, на якому збирається контейнер, надаючи йому служби операційної системи, які не надаються ядром.

Таким чином, *Docker* використовує не віртуалізацію, а засоби ядра, які дозволяють створювати ізольовані групи процесів. При запуску *Docker* робить лише кілька системних викликів і ядро створює для нового процесу окремий простір PID-ів, окрему віртуальну мережу, окремий набір обмежень по ресурсах. Ядро асоціює *Docker* зі специфічним набором налаштувань.

На відміну від контейнера, віртуальна машина (ВМ) працює під управлінням повноцінної операційної системи, включаючи її власне ядро, і є повною емуляцією іншого програмного (операційного) середовища. Перевагами та метою створення контейнерів є прискорення розробки, інкапсуляція додатків (залежностей додатків, операційних систем) та переносимість програмного забезпечення.

Всі контейнери створюються з образів контейнерів. Образи контейнерів представляють собою набір файлів, організованих в стек шарів, розташованих на локальному комп'ютері або у віддаленому реєстрі контейнерів. Образ контейнера складається з файлів операційної системи режиму користувача, необхідних для підтримки додатку, будь-яких середовищ виконання або залежностей додатків, а також будь-якого іншого файлу конфігурації, необхідного для правильної роботи додатка.

Таким чином, Docker – це стандартизоване пакетне програмне забезпечення, призначене для розробки, розгортання проектів та використання розроблених додатків, які є переносимими та самодостатніми, у той час як метою віртуальної машини є емуляція іншого операційного середовища. Docker дозволяє відокремити ваш додаток від вашої інфраструктури і дозволяє запустити буд-який додаток, який безпечно ізольований у контейнері. Docker має особливі образи програмного забезпечення, що запускаються у віртуальному середовищі, не створюючи повну копію ОС). Слід зазначити, що розробники можуть одночасно запускати десятки контейнерів, що дає можливість імітувати роботу промислової розподіленої мережі.

Одним з найбільш поширених варіантів використання контейнерів є мікросервіси (microservices). Мікросервіси – це спосіб розробки та компонування програмних систем, при якому вони формуються з невеликих незалежних

компонентів, що взаємодіють один з одним через мережу. 64-бітовий Linux-контейнер працює тільки на хості з встановленою 64-бітовою версією ОС Linux.

Docker доступний для будь-якої з операційних систем: Windows, Linux, Mac OS. Docker ставиться на версію Ubuntu 18.04, на Ubuntu 19.10 не ставиться. Docker дозволяє запустити ОС Linux в ізольованому середовищі дуже швидко, протягом декількох хвилин. Платформа Docker складається з двох окремих компонентів:

- *Docker Engine*, механізму, що відповідає за створення і функціонування контейнерів,
- *Docker Hub,* хмарного сервісу для поширення контейнерів.

Механізм *Docker Engine* надає ефективний і зручний інтерфейс для запуску контейнерів. До цього для запуску контейнерів, що використовують таку технологію, як, наприклад, LXC (Linux Container), були потрібні неабиякий запас спеціальних знань в цій області і великий обсяг ручної роботи. *Docker Hub* надає величезну кількість образів контейнерів з відкритим доступом для завантаження, дозволяючи користувачам швидко почати роботу з ними і уникнути рутинної роботи, раніше вже виконану іншими людьми.

Трохи пізніше були розроблені ***інструментальні засоби для Docker***:

- ***Swarm*** - менеджер кластерів,
- ***Kinematic*** - графічний користувальницький інтерфейс для роботи з контейнерами;
- ***Machine*** - утиліта командного рядка для підтримки роботи Docker-хостів.

Установка Docker

1). Дистрибутив Docker, доступний в офіційному репозиторії Ubuntu, не завжди є останньою версією програми. Доцільно встановити останню версію Docker, завантаживши її з офіційного репозиторію Docker. Для цього додаємо новий джерело дистрибутива, вводимо ключ GPG з репозиторія Docker, щоб переконатися, чи дійсна завантажена версія, а потім встановлюємо дистрибутив.

Спочатку оновлюємо існуючий перелік пакетів: ***sudo apt update***

Далі встановлюємо необхідні пакети, які дозволяють менеджеру пакетів *apt* використовувати пакети по HTTPS:

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

Потім додаємо в свою систему ключ GPG офіційного репозиторію Docker:

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

Додаємо репозиторій Docker в список джерел пакетів *APT*:

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

Далі оновимо базу даних пакетів інформацією про пакети Docker зі знову доданого сховища: ***sudo apt update***

Слід переконатися, що ми встановлюємо Docker з репозиторію Docker, а не з репозиторію за замовчуванням Ubuntu: ***apt-cache policy docker-ce***

На екран буде виведена наступна інформація (номер версії Docker може бути іншим):

```
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
```

Зверніть увагу, що *docker-ce* не встановлюється, але для установки буде використаний репозиторій Docker для Ubuntu 18.04 (*bionic*).

Далі встановлюємо Docker: ***sudo apt install docker-ce***

Docker встановлений, демон запущений, і процес буде запускатися при завантаженні системи. Переконаємося, що процес запущений:

sudo systemctl status docker

На екран виводиться наступна інформація, сервіс повинен бути запущений і активний:

```
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 установці Docker ми отримуємо не тільки сервіс (демон) Docker, але і утиліту командного рядка docker або клієнт Docker. Використання утиліти командного рядка docker розглянуто нижче.

2). Використання команди Docker без sudo (необов'язково)

За замовчуванням команду ***docker*** може запустити користувач *root* або користувач з групи *docker*, яка автоматично створюється при установці *Docker*. Якщо ви хочете запустити команду *docker* без префікса *sudo* або від імені користувача, що не входять в групу *docker*, будуть виведені дані:

```
Output
docker: Cannot connect to the Docker daemon. Is the docker daemon
running on this host?.
See 'docker run --help'.
```

Щоб не вводити *sudo* кожний раз при запуску команди *docker*, додайте ім'я свого користувача у групу *docker*: ***sudo usermod -aG docker username.***

Наприклад, ***sudo usermod -aG docker leo.***

Для застосування цих змін у складі групи необхідно розлогінитися і знову залогінитися на сервері або задати наступну команду: ***su - username.***

Наприклад, ***su - leo.***

Для продовження роботи необхідно ввести пароль користувача.

Щоб переконатися, що користувач доданий у групу *docker*, слід набрати команду: ***id -nG.*** На екран виведеться:

```
Output
username sudo docker
```

3). Використання команди *Docker*

Команда *docker* дозволяє використовувати різні опції, команди з аргументами. Синтаксис команди наступний: ***docker [option] [command] [arguments].***

Для перегляду усіх опцій, доступних команд управління та підкоманд, введіть: ***docker***

Опції:

--config string Location of client config files (default "/home/userbody/.docker",
-c, --context string Name of the context to use to connect to the daemon (overrides DOCKER_HOST env var and default context set with "docker context use"),
-D, --debug Enable debug mode,
-H, --host list Daemon socket(s) to connect to,
-l, --log-level string Set the logging level ("debug" | "warn" | "error" | "fatal") (default "info"),
--tls Use TLS; implied by --tlsverify,
--tlscacert string Trust certs signed only by this CA (default "/home/userbody/.docker/ca.pem"),
--tlscert string Path to TLS certificate file (default "/home/userbody/.docker/cert.pem"),
--tlskey string Path to TLS key file (default "/home/userbody/.docker/key.pem"),
--tlsverify Use TLS and verify the remove
-v, --version Print version information and quit.

Команди управління:

builder Manage builds,
config Manage Docker config,
container Manage containers,
context Manage context,
engine Manage the docker engine,
image Manage images,
network Manage networks
node Manage Swarm nodes
plugin Manage plugins
secret Manage Docker secrets
service services
stack Manage Docker stacks
swarm Manage Swarm
system Manage Docker
trust Manage trust on Docker images
volume Manage volumes

Повний список підкоманд Docker:

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 statistic,
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 перегляду опцій використання певної команди введіть:

docker docker-subcommand --help

Наприклад, переглянемо список опцій команди ***images***

docker images –help

Options:

-a, --all Show all images (default hides intermediate images),

--digest Show digests,
-f, --filter filter Filter output based on conditions provided,
--format string Pretty-print images using a Go template,
--no-trunc Don't truncate output,
-q, --quit Only show numeric IDs.

Для перегляду всієї інформації про *Docker* використовується наступна команда:

docker info

4). *Робота з образами Docker*

Контейнери *Docker* запускаються з образів *Docker*. За замовчуванням *Docker* отримує образи з хаба *Docker Hub [https://hub.docker.com]*, який являє собою реєстр образів і він підтримується компанією *Docker*. Будь-хто може створити і завантажити свої образи *Docker* в *Docker Hub*, тому для більшості додатків і дистрибутивів *Linux*, які можуть знадобитися вам для роботи, вже є відповідні образи в *Docker Hub.*

Для перевірки чи ви маєте доступ до образів і можете завантажувати образи з Docker Hub, введіть наступну команду: ***docker run hello-world***

Коректний результат роботи цієї команди, який означає, що Docker працює правильно, наведений нижче:

```
Output
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest:
sha256:3e1764d0f546ceac4565547df2ac4907fe46f007ea229fd7ef2718514bcec3
5d
Status: Downloaded newer image for hello-world:latest
```

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

Спочатку *Docker* не міг знаходити образ hello-world локально, тому завантажував образ з *Docker Hub*, який є репозиторієм за замовчуванням. Після завантаження образу *Docker* створював з образу контейнер і запускав додаток в контейнері, відображаючи повідомлення. Образи, доступні в Docker Hub, можна шукати за допомогою команди *docker* і підкоманди *search*. Наприклад, для пошуку образу *Ubuntu* вводимо:

docker search ubuntu

Скрипт переглядає *Docker Hub* і повертає список всіх образів, імена яких підходять під заданий пошук. Ми отримаємо наступний результат:

Output

NAME DESCRIPTION

ubuntu Ubuntu is a Debian-based Linux operating sys… 917
dorowu/ubuntu-desktop-lxde-vnc Ubuntu with openssh-server and NoVNC 193
rastasheep/ubuntu-sshd Dockerized SSH service, built on top of offi… 156
ansible/ubuntu14.04-ansible Ubuntu 14.04 LTS with ansible 93
ubuntu-upstart Upstart is an event-based replacement for th… 87
neurodebian NeuroDebian provides neuroscience research s… 50
ubuntu-debootstrap debootstrap --variant=minbase --components=m… 38
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5 ubuntu-16-nginx-php-phpmyadmin-mysql-5 36
nuagebec/ubuntu Simple always updated Ubuntu docker images w… 23
tutum/ubuntu Simple Ubuntu docker images with SSH access 18
i386/ubuntu Ubuntu is a Debian-based Linux operating sys… 13
ppc64le/ubuntu Ubuntu is a Debian-based Linux operating sys… 12
1and1internet/ubuntu-16-apache-php-7.0 ubuntu-16-apache-php-7.0 10
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mariadb-10 ubuntu-16-nginx-php-phpmyadmin-mariadb-10 6
eclipse/ubuntu_jdk8 Ubuntu, JDK8, Maven 3, git, curl, nmap, mc, 6
codenvy/ubuntu_jdk8 Ubuntu, JDK8, Maven 3, git, curl, nmap, mc, 4
darksheer/ubuntu Base Ubuntu Image -- Updated hourly 4
1and1internet/ubuntu-16-apache ubuntu-16-apache 3
1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4 ubuntu-16-nginx-php-5.6-wordpress-4 3
1and1internet/ubuntu-16-sshd ubuntu-16-sshd 1
pivotaldata/ubuntu A quick freshening-up of the base Ubuntu doc
1and1internet/ubuntu-16-healthcheck ubuntu-16-healthcheck 0
pivotaldata/ubuntu-gpdb-dev Ubuntu images for GPDB development 0
smartentry/ubuntu ubuntu with smartentry 0
ossobv/ubuntu
...

Коли потрібний образ обраний, можна завантажити його на комп'ютер за допомогою підкоманди ***pull.*** Щоб завантажити офіційний образ *ubuntu* на комп'ютер, запускається наступна команда: ***docker pull ubuntu***

Отримуємо наступний результат:

```
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:
sha256:5f4bdc3467537cbbe563e80db2c3ec95d548a9145d64453b0693
9c4592d67b6d
Status: Downloaded newer image for ubuntu:latest
```

Після завантаження образу можна запустити контейнер із завантаженим образом за допомогою підкоманди ***run***. *Я*к видно з прикладу *hello-world*, якщо при виконанні *docker* за допомогою підкоманди *run* образ ще не завантажений, клієнт *Docker* спочатку завантажить образ, а потім запустить контейнер з цим образом.

Для перегляду завантажених на комп'ютер образів потрібно ввести:

docker images

Отримаємо виведення на екран:

Output

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	113a43faa138	4 weeks ago	81.2MB
hello-world	latest	e38bc07ac18e	2 months ago	1.85kB

Образи, які використовуються для запуску контейнерів, можна змінювати і застосовувати для створення нових образів, які, в свою чергу, можуть бути завантажені (технічний термін ***push***) в *Docker Hub* або інший *Docker*-реєстр.

5). ***Запуск контейнера Docker***

Контейнер *hello-world*, запущений на попередньому етапі, є прикладом контейнера, який запускається і завершує роботу після виведення тестового повідомлення. Контейнери можуть виконувати і більш корисні дії, а також можуть бути інтерактивними. Контейнери схожі на віртуальні машини, але є менш вимогливими до ресурсів.

*Я*к приклад запустимо контейнер за допомогою останньої версії образу Ubuntu. Комбінація параметрів ***-i*** та ***-t*** забезпечує інтерактивний доступ до командного процесора контейнера: ***docker run -it ubuntu***

Командний рядок повинен змінитися, показуючи, що ми тепер працюємо в контейнері. Вона буде мати наступний вигляд:

Output

root @ d9b100f2f636: / #

В командному рядку відображається ідентифікатор контейнера. В даному прикладі це *d9b100f2f636.* Ідентифікатор контейнера потрібно нам пізніше, щоб вказати, який контейнер необхідно видалити.

Тепер можна запускати будь-які команди всередині контейнера. Наприклад, оновити базу даних пакета всередині контейнера. Перед командами не потрібно використовувати *sudo*, оскільки ви працюєте всередині контейнера як користувач з привілеями ***root***: ***apt update.***

Тепер в контейнері можна встановити будь-який додаток. Спробуємо встановити ***Node.js*** (середовище виконання *JavaScript*): ***apt install nodejs.***

Ця команда встановлює *Node.js* в контейнер з офіційного репозиторію *Ubuntu*. Коли установка завершена, переконаємося, що *Node.js* встановлений: ***node -v***

У терміналі з'явиться номер версії:

Output

v8.10.0

Всі зміни, які ви здійснюєте всередині контейнера, застосовуються тільки для цього контейнера. Щоб вийти з контейнера, вводимо команду ***exit.***

6). ***Управління контейнерами Docker***

Через деякий час після початку використання *Docker* на вашій машині буде безліч активних (запущених) і неактивних контейнерів.

Перегляд ** активних контейнерів **: ***docker ps***

Результат перегляду:

```
Output
CONTAINER ID        IMAGE          COMMAND        CREATED
```

Було запущено два контейнери: один з образу *hello-world*, другий з образу *ubuntu*. Обидва контейнера вже не запущені, але існують в системі.

Щоб побачити і активні, і неактивні контейнери, запускаємо *docker ps* за допомогою параметра ***-a:*** ***docker ps -a***

Результат наступний:

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

Для перегляду останнього створеного контейнерів, задаємо 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 запуску зупиненого контейнера використовуємо команду ***docker start***, потім вказуємо ідентифікатор контейнера або його ім'я. Запустимо завантажений з *Ubuntu* контейнер з ідентифікатором *d9b100f2f636:*

docker start d9b100f2f636

Контейнер запускається. Тепер для перегляду його статусу можна використовувати ***docker ps***:

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
d9b100f2f636 ubuntu "/bin/bash" About an hour ago Up 8 seconds sharp_volhard

Для зупинки запущеного контейнера використовуємо команду ***docker stop***, потім вказуємо *ідентифікатор контейнера* або його *ім'я.* Цього разу ми використовуємо ***ім'я, яке призначив контейнеру Docker***, тобто ***sharp_volhard***:

docker stop sharp_volhard

Якщо вам ***контейнер*** більше не потрібен, ***видаляємо*** його командою ***docker rm*** із зазначенням або ідентифікатора, або імені контейнера.

Щоб знайти ідентифікатор або ім'я контейнера, пов'язаного з образом *hello-world*, використовуйте команду *docker ps -a.* Потім контейнер можна видалити:

docker rm festive_williams

Запустити новий контейнер і надати йому ім'я можна за допомогою параметра «***-name».*** Параметр «***-rm»*** дозволяє створити контейнер, який самостійно віддалиться після зупинки. Для більш докладної інформації про дані та інших опціях використовуйте команду ***docker run help***.

Контейнери можна перетворити в образи для побудови нових контейнерів. Розглянемо, як це зробити.

7). ***Збереження змін в контейнері в образ Docker***

При запуску контейнера з образу *Docker* ви можете створювати, змінювати і видаляти файли, як і на віртуальній машині. Внесені зміни застосовуються тільки для такого контейнера. Можна запускати і зупиняти контейнер, проте як тільки він буде знищений командою *docker rm*, всі зміни будуть безповоротно втрачені. В

даному розділі показано, як зберегти стан контейнера у вигляді нового образу *Docker*.

Після установки *Node.js* в контейнері *Ubuntu* у вас буде працювати запущений з образу контейнер, але він буде відрізнятися від образу, який ви використовували для його створення. Однак вам може знадобитися такий контейнер *Node.js* як основа для майбутніх образів. Далі підтверджуємо зміни в новому образі *Docker* за допомогою наступної команди.

docker commit -m "What you did to the image" -a "Author Name" container_id repository /new_image_name

Параметр «**-m**» дозволяє задати повідомлення про підтвердження, щоб полегшити користувачам образу розуміння того, які зміни були внесені, а параметр «**-a**» дає змогу вказати автора. Ідентифікатор контейнера *container_id* - цей ідентифікатор, який використовувався раніше, коли починали інтерактивну сесію в контейнері *Docker*. Якщо ви не створювали додаткових репозиторіїв в *Docker Hub*, ім'я сховища (*repository*) зазвичай є вашим ім'ям користувача в *Docker Hub*.

Наприклад, для *користувача* ***sammy*** та *ідентифікатора контейнера* d9b100f2f636 команда виглядає наступним чином:

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

Після підтвердження (*commit*) образу новий образ зберігається локально на вашому комп'ютері. Для того щоб розмітити образ в реєстр *Docker* (наприклад, в *Docker Hub)* так, щоб він був доступний не тільки вам, а й іншим користувачам, необхідно виконати наступні дії. Для перегляду списку образів *Docker*, в ньому з'являться і новий образ, і початковий образ, на якому він був заснований виконаємо команду: ***docker images***

Результат буде наступним:

OutputREPOSITORY	*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*

У цьому прикладі *ubuntu-nodejs* - це новий образ, створений на основі існуючого образу *ubuntu* з *Docker Hub*. Різниця розмірів відображає внесені зміни. В даному прикладі зміна пов'язана з установкою *NodeJS*. У випадку, коли буде потрібно запустити контейнер *Ubuntu* з передвстановленим *NodeJS*, можна використовувати цей новий образ. Образи також можна створювати за допомогою файлу *Dockerfile*, який дозволяє автоматизувати установку програм в новому образі.

Новим образом можна поділитися з іншими користувачами, щоб вони могли створювати на його основі контейнери.

8). ***Завантаження контейнерів Docker в репозиторій Docker***

Для завантаження образів в *Docker Hub* або інший *Docker-реєстр*, до якого у вас є доступ, ви повинні мати у ньому обліковий запис.

Для *створення власного Docker-реєстр*у та його налаштування можна скористатися статтею *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*. Для того щоб завантажити свій образ *Docker у Docker Hub*, треба увійти в *Docker Hub*: ***docker login -u docker-registry-username***

Наприклад, ***docker login -u sammy.***

Для входу у *Docker Hub* потрібно ввести пароль. Після введення правильного паролю ви будете успішно авторизовані. Якщо ім'я користувача в *Docker-реєстрі* відрізняється від локального імені користувача, яке використовувалося для створення образу, необхідно прив'язати свій образ до імені користувача в реєстрі. Для цього вводимо команду з урахуванням попереднього прикладу:

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

Наприклад, ***docker tag leo/ubuntu-nodejs sammy/ubuntu-nodejs***

Далі завантажуємо власний образ *Docker Hub*:

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

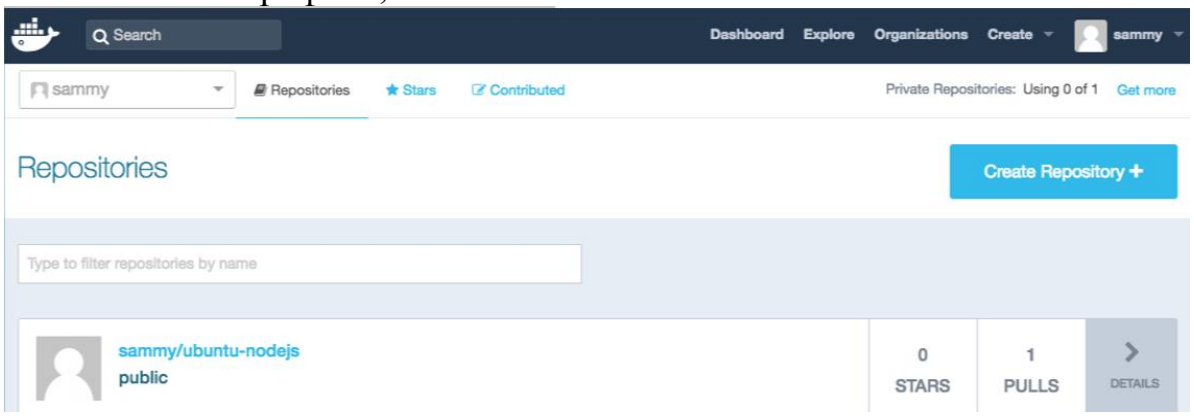
Наприклад, ***docker push sammy/ubuntu-nodejs***

Команда для завантаження образу *ubuntu-nodejs* в репозиторій виглядає наступним чином: ***docker push sammy/ubuntu-nodejs***

Для завантаження образу може знадобитися деякий час, але після завершення результат буде виглядати наступним чином:

Output
The push refers to a repository [docker.io/sammy/ubuntu-nodejs]
e3fbbfb44187: Pushed
5f70bf18a086: Pushed
a3b5c80a4eba: Pushed
7f18b442972b: Pushed
3ce512daaf78: Pushed
7aae4540b42d: Pushed

Після завантаження образу в реєстр його ім'я з'являється в списку панелі управління вашого профілю, як показано нижче:



Якщо при завантаженні з'являється наступна помилка (це означає, що не виконано вхід до реєстру):

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

необхідно повторити авторизацію.

Для авторизації в реєстрі повторюємо команду ***docker login*** та завантажуємо образ. Потім треба перевірити, що він з'явився на вашій сторінці в репозиторії *Hub.*

Далі за допомогою команди ***docker pull sammy ubuntu-nodejs*** можна завантажити образ на нову машину і використовувати його для запуску нового контейнера.

Для перевірки порту для *localhost* в ***nodejs*** можете використовувати ***netcat***, щоб перевірити чи працює ваша служба: ***nc example.com 8080.***

Завдання:

1. Ознайомитися з теоретичними матеріалом по лабораторній роботі.
2. Опанувати команди, які використовують при установці *Docker* та його встановити.
3. Підготувати звіт з описом процесу установки *Docker* з наведенням *screenshot*-ів екрану при виконанні кожної дії, надати його для викладача.

Хід виконання роботи

Встановити дистрибутив *Docker*.

Підготувати звіт

1. Описати хід виконання поставлених завдань, надаючи знімок екрану (*screenshot*).
2. Висновки по роботі.

Контрольні питання

1. Що таке Docker?
2. Що таке контейнер?
3. Що таке образ?
4. Що таке реєстр?
5. Що таке репозитарій?
6. Які відмінності між віртуальною машиною та контейнерами?
7. У яких операційних системах можна встановлювати Docker?
8. З яких компонентів складається платформа Docker?
9. Які вам відомі підкоманди Docker?
10. Як запустити команду *docker* без префікса *sudo?*
11. Як працювати з образами Docker?
12. Як запустити контейнер Docker, зупинити та видалити?
13. Як завантажити контейнер Docker в репозитарій Docker?

Література

1. Моэт Э. Использование Docker. Москва : ДМК Пресс, 2017. 354 с.
2. Сейерс Э. Х., Милл А. Docker на практике. Москва : ДМК. 2019. 516 с.
3. Парминдер Сингх Кочер. Микросервисы и контейнеры Docker. Москва : ДМК Пресс, 2019. 240 с.
4. Сайфан Джиджи. Осваиваем Kubernetes. Оркестрация контейнерных архитектур. Санкт-Петербург : Питер, 2016. 522 с.

5. https://dker.ru/docs/ - Docker документація російською.