МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Встановлення та ознайомлення із Docker контейнерами

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи з курсу
“Операційні системи”
для студентів базового напрямку 6.050101
“Комп’ютерні науки”

Затверджено
на засіданні кафедри
“Системи автоматизації проектування”

Львів – 2020

Мета: Ознайомитись із основними особливостями запуску та налаштування Docker контейнерів та параметрами запуску

Короткі теоретичні відомості

Docker

Докер - це відкрита платформа для розробки, доставки і експлуатації додатків. Docker розроблений для більш швидкого викладання додатків. За допомогою docker можна відокремити додаток від інфраструктури і звертатися з інфраструктурою як керованим додатком. Docker допомагає викладати код швидше, швидше тестувати, швидше викладати додатки і зменшити час між написанням і запуском коду. Docker робить це за допомогою легкої платформи контейнерної віртуалізації, використовуючи процеси і утиліти, які допомагають керувати і викладати програми.

У своєму ядрі docker дозволяє запускати практично будь-який додаток, безпечно ізольований в контейнері. Безпечна ізоляція дозволяє запускати на одному хості багато контейнерів одночасно. Легка природа контейнера, який запускається без додаткового навантаження гіпервізора, дозволяє вам ефективніше використовувати можливості заліза.

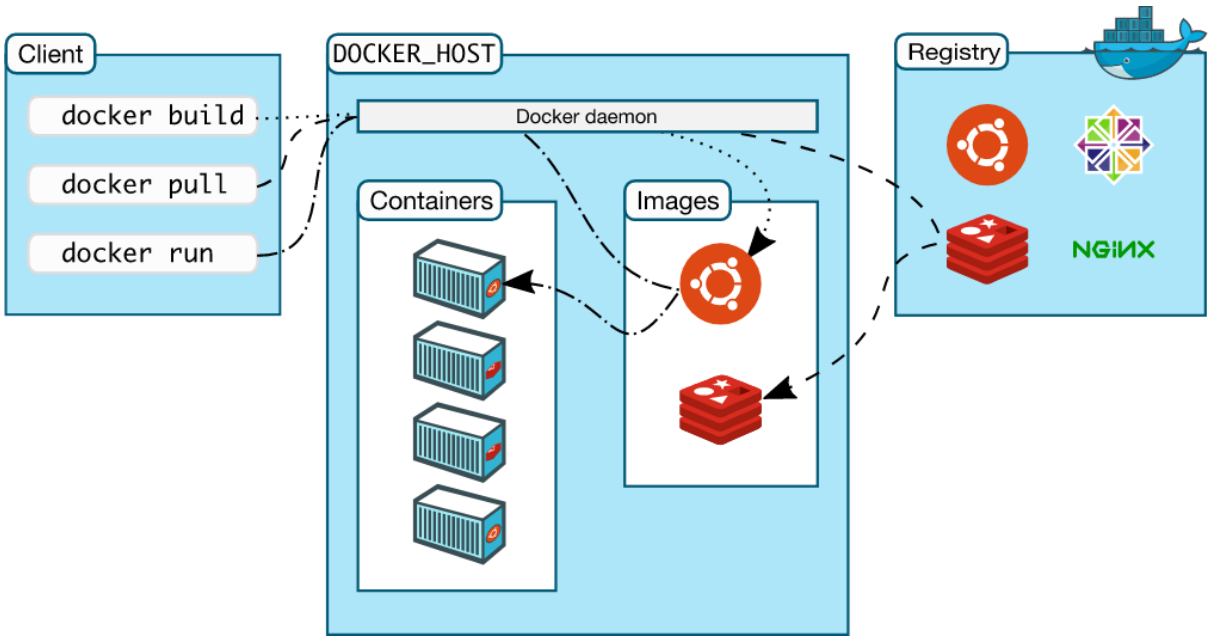
Архітектура Docker

Docker складається з двох головних компонентів:

- Docker: платформа віртуалізації з відкритим кодом;
- Docker Hub: платформа-як-сервіс для поширення і управління docker контейнерами.

Docker використовує архітектуру клієнт-сервер. Docker клієнт спілкується з демоном Docker, який бере на себе створення, запуск, розподіл контейнерів.

Обидва, клієнт і сервер можуть працювати на одній системі, також можна підключити клієнт до віддаленого демона docker. Клієнт і сервер спілкуються через сокет або через RESTful API.



Архітектура Docker

Головні компоненти Docker

Щоб розуміти, з чого складається docker, потрібно знати про три його компоненти:

- образи (images)
- реєстр (registries)
- контейнери

Docker-образ - це read-only шаблон. Наприклад, образ може містити операційну систему Ubuntu з Apache і додатком на ній. Образи використовуються для створення контейнерів. Docker дозволяє легко створювати нові образи, оновлювати існуючі, або можна завантажити образи створені іншими людьми. Образи - це компонента збірки docker-а.

Docker-реєстр зберігає образи. Є публічні і приватні реєстри, з яких можна скачати або завантажити образи. Публічний Docker-реєстр - це Docker Hub. Там зберігається величезна колекція образів. Образи можуть бути створені вами або можна використовувати образи створені іншими користувачами. Реєстри - це компонента поширення. Контейнери схожі на директорії. У контейнерах міститься все, що потрібно для роботи програми. Кожен контейнер створюється з образу. Контейнери можуть бути створені, запущені, зупинені, перенесені або видалені. Кожен контейнер ізольований і є безпечною платформою для додатка. Контейнери - це компонента роботи.

Виходячи з цих трьох компонентів в Docker можна:

- створювати образи, в яких знаходяться додатки;
- створювати контейнери з образів, для запуску додатків;
- розповсюдженню через Docker Hub або інший реєстр образів.

Принцип роботи Docker

Отже образ - це read-only шаблон, з якого створюється контейнер. Кожен образ складається з набору рівнів. Docker використовує union file system для поєднання цих рівнів в один образ. Union file system дозволяє файлам директоріями з різних файлових систем (різних гілок) прозоро накладатися, створюючи когерентну файлову систему.

Одна з причин, по якій docker легкий - це використання таких рівнів. Коли змінюється образ, наприклад, проходить оновлення додатку, створюється новий рівень. Так, без заміни всього образу або його перезібрання, як вам можливо доведеться зробити з віртуальною машиною, тільки рівень додається або оновлюється. І вам не потрібно роздавати весь новий образ, публікується тільки оновлення, що дозволяє поширювати образи простіше і швидше.

В основі кожного образу знаходиться базовий образ. Наприклад, ubuntu, базовий образ Ubuntu, або fedora, базовий образ дистрибутива Fedora. Так само можна використовувати готові образи як базу для створення нових образів. Наприклад, образ apache можна використовувати як базовий образ для веб-додатків. Docker зазвичай бере образи з реєстру Docker Hub.

Docker образи можуть створитися з цих базових образів, кроки опису для створення цих образів називаються інструкціями. Кожна інструкція створює новий образ або рівень. Інструкціями будуть наступні дії:

- запуск команди;
- додавання файлу або директорії;
- створення змінної оточення;
- вказівки що запускати коли запускається контейнер цього способу.

Ці інструкції зберігаються в файлі Dockerfile. Docker зчитує цей Dockerfile, коли збирається образ, виконує ці інструкції, і повертає кінцевий образ.

Реєстр - це сховище docker образів. Після створення образу ви можете опублікувати його на публічному реєстрі Docker Hub або на вашому особистому реєстрі. За допомогою docker клієнта ви можете шукати вже опубліковані образи і завантажувати їх на машину з docker для створення контейнерів.

Docker Hub надає публічні і приватні сховища образів. Пошук і скачування образів з публічних сховищ доступний для всіх. Вміст приватних сховищ не попадає в результат пошуку. І тільки ви і ваші користувачі можуть отримувати ці образи і створювати з них контейнери.

Принцип роботи контейнера

Контейнер складається з операційної системи, призначених для користувача файлів і метаданих. Відомо, що кожен контейнер створюється з образу. Цей образ говорить docker-у, що знаходиться в контейнері, який процес запустити, коли запускається контейнер та інші конфігураційні дані. Docker образ доступний тільки для читання. Коли docker запускає контейнер, він створює рівень для читання / запису зверху образу (використовуючи union file system, як було зазначено раніше), в якому може бути запущено додаток.

Або за допомогою програми docker, або за допомогою RESTful API, docker клієнт говорить docker-демону запустити контейнер.

```
$ sudo docker run -i -t ubuntu /bin/bash
```

Давайте розберемося з цією командою. Клієнт запускається за допомогою команди docker, з опцією run, яка говорить, що буде запущений новий контейнер. Мінімальними вимогами для запуску контейнера є такі атрибути:

- який образ використовувати для створення контейнера. У нашому випадку ubuntu;
- команду яку ви хочете запустити коли контейнер буде запущений. У нашому випадку /bin/bash;

Після запуску цієї команди Docker, по порядку, робить наступне:

- завантажує образ ubuntu: docker перевіряє наявність образу ubuntu на локальній машині, і якщо його немає - то викачує його з Docker Hub. Якщо ж образ є, то використовує його для створення контейнера;
- створює контейнер: коли образ отриманий, docker використовує його для створення контейнера;
- ініціалізує файлову систему і монтує read-only рівень: контейнер створений в файлової системі і read-only рівень доданий образ;
- ініціалізує мережу / міст: створює мережевий інтерфейс, який дозволяє docker-у спілкуватися хост машиною;
- Установка IP адреси: знаходить і задає адресу;
- Запускає вказаний процес: запускає програму;

- Обробляє та видає вихід додатку: підключається і залоговує стандартний вхід, вихід і потік помилок додатку, щоб можна було відслідковувати як працює програма.

Тепер у вас є робочий контейнер. Ви можете управляти своїм контейнером, взаємодіяти з вашим додатком. Коли вирішите зупинити додаток, видаліть контейнер.

Технології, використані у Docker

Докер написаний на мові Go і використовує деякі можливості ядра Linux, щоб реалізувати наведений вище функціонал. Docker використовує технологію namespaces для організації ізольованих робочих просторів, які називаються контейнерами. Коли запускається контейнер, docker створює набір просторів імен для даного контейнера. Це створює ізольований рівень, кожен контейнер запущений в своєму просторі імен, і не має доступ до зовнішньої системи.

Список деяких просторів імен, які використовує docker:

- **pid**: для ізоляції процесу;
- **net**: для управління мережевими інтерфейсами;
- **ipc**: для управління IPC ресурсами. (ICP: InterProccess Communication);
- **mnt**: для управління точками монтування;
- **utc**: для ізолювання ядра і контролю генерації версій (UTC: Unix timesharing system).

Control groups (контрольні групи). Docker також використовує технологію cgroups або контрольні групи. Ключ до роботи додатка в ізоляції, надання додатку тільки тих ресурсів, які йому потрібно. Це гарантує, що контейнери будуть добре співіснувати. Контрольні групи дозволяють розділяти доступні ресурси заліза і якщо необхідно, встановлювати межі і обмеження. Наприклад, обмежити можливу кількість пам'яті, що використовується контейнером.

Union File Sysem або UnionFS - це файлова система, яка працює створюючи рівні, що робить її дуже легкою і швидкою. Docker використовує UnionFS для створення блоків, з яких будується контейнер. Docker може використовувати кілька варіантів UnionFS включаючи: AUFS, btrfs, vfs і DeviceMapper.

Docker поєднує ці компоненти в обгортку, яку ми називаємо форматом контейнера. Формат, який використовується за умовчанням, називається

libcontainer. Так само docker підтримує традиційний формат контейнерів в Linux с допомогою LXC. В майбутньому Docker можливо буде підтримувати інші формати контейнерів. Наприклад, інтегруючись з BSD Jails або Solaris Zones.

Хід роботи

1. ***Встановлення Docker***

a) Docker розміщує сценарій запуску на вашому комп'ютері. Ми можемо просто запустити команду.

```
sudo curl -sSL https://get.docker.com/ | sh
```

b) Як тільки команда виконається, ви побачите встановлену версію, як вказано нижче (ваш варіант може бути новішим, це нормально) та інструкції для запуску без повноважень root/ без sudo.

```
Output    Client:
 Version:      1.8.3
API version:  1.20
Go version:   go1.4.2
Git commit:   f4bf5c7
Built:        Mon Oct 12 05:37:18 UTC 2015
OS/Arch:      linux/amd64

Server:
Version:      1.8.3
API version:  1.20
Go version:   go1.4.2
Git commit:   f4bf5c7
Built:        Mon Oct 12 05:37:18 UTC 2015
OS/Arch:      linux/amd64
```

c) Необов'язково: Запустіть контейнер `+ hello-world +` , для того щоб переконатися, що програма працює коректно.

```
sudo docker run hello-world
```

d) Ви побачите такий же результат, як показано нижче.

```
Output    $ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
535020c3e8ad: Pull complete
af340544ed62: Already exists
library/hello-world:latest: The image you are pulling has been
verified. Important: image verification is a tech preview feature and
should not be relied on to provide security.
```

```
   Digest:
sha256:d5fbd996e6562438f7ea5389d7da867fe58e04d581810e230df4cc073271ea52
   Status: Downloaded newer image for hello-world:latest

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

   To generate this message, Docker took the following steps:
    1. The Docker client contacted the Docker daemon.
    2. The Docker daemon pulled the "hello-world" image from the Docker
Hub.
    3. The Docker daemon created a new container from that image which
runs the
       executable that produces the output you are currently reading.
    4. The Docker daemon streamed that output to the Docker client,
which sent it
       to your terminal.

   To try something more ambitious, you can run an Ubuntu container
with:
    $ docker run -it ubuntu bash

   Share images, automate workflows, and more with a free Docker Hub
account:
    https://hub.docker.com

   For more examples and ideas, visit:
    https://docs.docker.com/userguide/
```

Docker успішно встановлено!

2. Перегляд основ контейнера: Виконати, Список, Видалити

Ми встановили Docker Client як частину встановлення Docker, тому у нас є доступ до інструменту командної стрічки, який дозволяє нам працювати з контейнерами.

a) Для того, щоб побачити деяку основну інформацію про контейнер, запускаємо наступну команду:
   `sudo docker ps -a`
b) Ми отримаємо приблизно такий вивід:

```
OutputCONTAINER ID  IMAGE  COMMAND  CREATED       STATUS    PORTS NAMES
a3b149c3ddea        hello-world  "/hello" 3 minutes ago Exited (0) 3 seconds ago
```

Так ми можемо побачити основну інформацію про наш контейнер.

Ви можете помітити, що в нього може бути дивна назва, наприклад `++;` Ці імена генеруються автоматично, якщо ви не вкажете їх при створенні контейнера.
Також можемо помітити, що приклад контейнера `+ hello-world +` був запущений 3 хвилини тому і завершений 3 хвилини тому.

c) Якщо ми знову запустимо контейнер за допомогою цієї команди(замінивши ім'я `++` на власне ім'я контейнера):
```
sudo docker start
```

d) Після чого виконаємо команду для виводу списку контейнерів:
```
sudo docker ps -a
```

e) Бачимо, що контейнер запущений недавно:
```
OutputCONTAINER ID  IMAGE  COMMAND  CREATED       STATUS    PORTS NAMES
a3b149c3ddea        hello-world  "/hello" 4 minutes ago Exited (0) 9 seconds ago
```

За замовчуванням контейнери Docker запускають команди, які їм задано, а опісля виконання, закінчують роботу.
Деякі контейнери будуть налаштовані для виконання списку завдань і закінчення, в той час, коли інші працюватимуть настільки довго, наскільки це потрібно.

f) Тепер, коли ми познайомилися з основами Docker, давайте видалимо зображення `+ hello-world +` , оскільки воно нам більше не потрібне(не забудьте змінити `++` на ім'я вашого контейнера):
```
sudo docker rm
```

3. Налаштування порту
Завантаження Nginx Docker, запуск контейнера в якості загальнодоступного веб-сервера.
За замовчуванням до контейнерів немає доступу через інтернет, тому нам потрібно зіставити *internal* порт контейнера з портом *Droplet.*

a) Виконайте наступну команду, для того щоб отримати образ Nginx Docker:
```
sudo docker pull nginx
```
Ця команда завантажує всі необхідні компоненти для контейнера. Docker буде їх кешувати, саме тому під час запуску контейнера нам вже не потрібно щоразу завантажувати образ або образи контейнера.

Docker підтримує сайт Dockerhub, загальнодоступний репозиторій файлів Docker (включаючи як офіційні так і користувацькі зображення). Зображення, яке ми завантажили, є офіційним Nginx, завдяки якому нам не потрібно створювати власний образ.

b) Давайте запустимо наш контейнер Nginx Docker за допомогою цієї команди.

```
sudo docker run --name docker-nginx -p 80:80 nginx
```

- `+ run +` команда для створення нового контейнера;
- Прапорець `+ - name +` так ми вказуємо ім'я контейнера(якщо його залишити пустим, воно буде призначено автоматично, як в кроці 2);
- `+ -p +` визначає порт, який ми представляємо в форматі `+ -p local-machine-port: internal-container-port +` . В такому випадку ми зіставляємо порт 80 в контейнері з портом 80 на сервері;
- `+ nginx` - це ім'я образу на Dockerhub (ми завантажували його раніше, за допомогою команди pull, але Docker зробить це автоматично, якщо образу немає);

Це все, що нам потрібно для інсталяції та підготовки до роботи Nginx. Для того, щоб перевірити роботу веб-сервера, вставте IP-адресу в браузер, і ви побачите вітальну сторінку Nginx.

Також в сеансі оболонки ви можете помітити, що журнал Nginx оновлюється, коли ви надсилаєте запити на ваш сервер, тому що ми запускаємо нас контейнер в інтерактивному режимі.

Натиснувши комбінацію клавіш `+ CTRL + C +` , ми повернемося до сеансу оболонки.

Якщо ви будете намагатися завантажити сторінку зараз, ви отримаєте "Відмовлено в з'єднанні". Це тому, що ми закрили контейнер. Ми можемо перевірити його за допомогою команди:

```
sudo docker ps -a
```

Вивід має бути приблизно такий:

```
OutputCONTAINER ID    IMAGE          COMMAND                  CREATED
05012ab02ca1          nginx    "nginx -g 'daemon off"   57 seconds ago

       STATUS                         PORTS               NAMES
Exited (0) 47 seconds ago                                docker-nginx
```

Ми бачимо, що наш контейнер Docker вийшов(завершив роботу).
Nginx не буде корисним, якщо нам потрібно приєднатися до образу контейнера, для того щоб він працював, тому на наступному кроці ми будемо від'єднувати контейнер, для того, щоб він працював незалежно.

c) Видаліть існуючий контейнер `+ docker-nginx +` за допомогою цієї команди:
   `sudo docker rm docker-nginx`

4. Робота в автономному режимі

a) Створіть новий окремий Nginx-контейнер за допомогою команди:
   `sudo docker run --name docker-nginx -p 80:80 -d nginx`

Ми додали прапорець `+ -d +` для запуску цього контейнера в фоновому режимі.
Вихідні дані повинні бути ідентифікатором нового контейнера.
Якщо ми запустимо команду list:
`sudo docker ps`
Ми побачимо дещо інше, ніж ми бачили раніше

```
OutputCONTAINER ID    IMAGE          COMMAND                  CREATED
b91f3ce26553          nginx    "nginx -g 'daemon off"   About a minute
ago

     STATUS                         PORTS                     NAMES
Up About a minute     0.0.0.0:80->80/tcp, 443/tcp        docker-nginx
```

Як можна помітити, замість `+ Exited (0) X минут назад +` тепер в нас є `+ Up около минуты +` , і ми також можемо бачити порти.
Якщо ми знову перейдемо на IP-адресу нашого сервера в браузері, ми знову побачимо вітальну сторінку Nginx. Цього разу він працює в фоновому режимі, тому що ми вказали прапорець `+ -d +` , який вказує Docker запускати цей контейнер в окремому режимі.

Тепер в нас є запущений екземпляр Nginx в окремому контейнері.

Але цього поки недостатньо, тому що ми не можемо редагувати файл конфігурації, і контейнер не має доступа до файлів нашого веб-сайту.

b) Зупиніть контейнер, виконавши наступну команду:
```
sudo docker stop docker-nginx
```

c) Тепер, коли контейнер зупинено(ви можете перевірити за допомогою `+ sudo docker ps -a +`), ми можемо видалити його, виконавши наступну команду:
```
sudo docker rm docker-nginx
```

5. Створення веб-сторінки для обслуговування на Nginx

Створення сторінки індекса для вашого сайту. Це налаштування дозволяє нам мати постійний контент веб-сайту, який розміщений поза(часового) контейнером.

a) Створюємо нову папку для файлів веб-сайту і перейдемо до неї, виконавши команди:
```
mkdir -p ~/docker-nginx/html
cd ~/docker-nginx/html
```

b) Створюємо HTML-файл(в прикладі команди для Vim, але ви можете використовувати будь-який редактор)
```
vim index.html
```

c) Зайдіть в режим вставки, натиснувши `+ i +`. Вставте текст, який показано нижче(або додайте власну HTML-розмінтку):
```
<html>
 <head>
   <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstr
ap.min.css" rel="stylesheet" integrity="sha256-
MfvZlkHCEqatNoGiOXveE8FIwMzZg4W85qfrfIFBfYc= sha512-
dTfge/zgoMYpP7QbHy4gWMEGsbsdZeCXz7irItjcC3sPUFtf0kuFbDz/ixG7ArTxm
DjLXDmezHubeNikyKGVyQ==" crossorigin="anonymous">
   <title>Docker nginx Tutorial</title>
 </head>
 <body>
   <div class="container">
     <h1>Hello Digital Ocean</h1>
     <p>This nginx page is brought to you by Docker and Digital
Ocean</p>
   </div>
 </body>
```

```
</html>
```

d) Зберігаємо цей файл, натиснувши + ESC +, а після +: wq + і + ENTER +:
   - write(+ w +)записати зміни;
   - quit(+ q +)вийти;

6. Зв'язок контейнера з локальною файловою системою

Запуск контейнера Nginx, щоб він був доступний через інтернет через порт 80, і підключення його до файлів веб-сайту на сервері.

Довідкова інформація про об'єми

Docker дозволяє нам зв'язати папки локальної файлової системи віртуальної машини з контейнерами.

В нашому випадку, ми хочемо обслуговувати веб-сторінки, тому нам потрібно надати нашому контейнеру файли для рендерингу.

Ми б могли скопіювати файли в контейнер як частину dockerfile або скопіювати їх в контейнер після факту, але ці методи залишають сайт в статичному стані всередині контейнера. Використовуючи функцію томів даних в Docker, ми можемо створити символічний зв'язок між файловою системою Droplet і файловою системою контейнера. Це дозволяє нам редагувати існуючі файли веб-сторінок і додавати нові в папку, і контейнер автоматично отримає до них доступ.

Контейнер Nginx за замовчуванням налаштований на пошук індексної сторінки в + / usr / share / nginx / html +, тому в новому контейнері Docker нам потрібно надати йому доступ до наших файлів в цьому місці.

Оформлення посилання

Для цього ми використовуємо прапорець + -v +, для того щоб зіставити папку з локального комп'ютера (+ ~ / docker-nginx / html +) з відносним шляхом в контейнері(+ / usr / share / nginx /). HTML +).

a) Ми можемо зробити це, виконавши наступну команду:
```
sudo docker run --name docker-nginx -p 80:80 -d -v ~/docker-nginx/html:/usr/share/nginx/html nginx
```

Ми бачимо, що новий додаток до команди + -v ~ / docker-nginx / html: / usr / share / nginx / html + є нашим посиланням на том.

- + -v + вказує, що ми зв'язуємо том;

- частина зліва від `+: +` - це розташування вашого файлу/папки на вашій віртуальній машині(`+ ~ / docker-nginx / html`);
- частина справа від `+: +` - це розташування, на яке ми посилаємося в нашому контейнері(`+ / usr / share / nginx / html +`);

Після виконання цієї команди , якщо ви тепер в браузері вкажете IP-адресу DigitalOcean Droplet, ви повинні побачити перший заголовок “Hello Digital Ocean” (або будь-яку сторінку, яку ви створили на кроці 5).

Якщо вас влаштовують інші налаштування Nginx за замовчуванням, все готово.

Ви можете завантажити більше контенту в папку `+ ~ / docker-nginx / html / +`, і він з’явиться на вашому живому сайті.

Наприклад, якщо ми змінимо індексний файл і перезагрузимо вікно браузера, ми зможемо побачити його оновлення в режимі реального часу. Таким чином, ми могли б створити цілий сайт з плоских HTML-файлів, якщо б захотіли. Наприклад, якщо ми додали на сторінку `+ about.html +`, ми можемо отримати до неї доступ за адресою `+ http: /// about.html +` без необхідності взаємодії з контейнером.

7. Використання власного файла конфігурації Nginx

a) Повернемося до папки, тому ми не пишемо загальнодоступну папку HTML:
```
cd ~/docker-nginx
```

b) Якщо ви хочете відкрити файл конфігурації за замовчуванням, просто скопіюйте його за допомогою команди Docker copy:
```
sudo docker cp docker-nginx:/etc/nginx/conf.d/default.conf
default.conf
```

Оскільки ми будемо використовувати користувацький файл `+ .conf +` для Nginx, нам потрібно перебудувати контейнер.

c) Для початку зупиняємо контейнер:
```
sudo docker stop docker-nginx
```

d) Видаліть це з:
```
sudo docker rm docker-nginx
```

e) Тепер ви можете відредагувати файл за замовчуванням локально(для обслуговування нової папки або використовувати `+ proxy_pass +` для пересилання трафіку в інший додаток/контейнер, як ви б це робили під час звичайного встановлення Nginx).

f) Після того, як ви зберегли свій користувацький файл конфігурації, прийшов час створити контейнер Nginx. Просто додайте другий прапорець `+ -v +` з відповідними шляхами, для того щоб дати свіжостворюваному контейнеру Nginx потрібні посилання для запуску з власного файлу конфігурації.

```
sudo docker run --name docker-nginx -p 80:80 -v ~/docker-nginx/html:/usr/share/nginx/html -v ~/docker-nginx/default.conf:/etc/nginx/conf.d/default.conf -d nginx
```

g) Зверніть увагу, що вам потрібно перезапустити контейнер за допомогою команди `+ docker restart +` , якщо ви вносите будь-які зміни в файл конфігурації після запуску контейнера, оскільки Nginx не виконує екстрене перезавантаження, якщо його файл конфігурації було змінено:

```
sudo docker restart docker-nginx
```