УДК 004.056
В.М. Мельник, А.І. Нагорнюк
Луцький національний технічний університет

ВИКОРИСТАННЯ DOCKER ДЛЯ СТВОРЕННЯ ВІДОКРЕМЛЕНОГО ДОСЛІДНИЦЬКОГО СЕРЕДОВИЩА

Мельник В.М. Нагорнюк А.І. Використання DOCKER для створення відокремленого дослідницького середовища. В даній роботі було розглянуто застосування технології контейнеризації для створення незалежного середовища для проведення досліджень в напрямі порівняння швидкодії різних баз даних. Зокрема було висвітлено характерні особливості створення образів і контейнерів. Проведено порівняння з іншою технологією віртуалізації. Визначено основні переваги використання технології Docker.
Ключові слова: Docker, контейнеризація, CI/CD, особливості використання.

Melnyk V.M. Nahorniuk A.I. Use DOCKER to create a separate research environment. In this paper, the use of containerization technology to create an independent environment for conducting research in the direction of comparison of high-speed databases was considered. In particular, the nature of the features creating images and containers was highlighted. A comparison with another virtualization technology has been made. The main advantages of Docker technology are determined.
Keywords: Docker, Containerization, CI / CD, features of use.

Мельник В.М. Нагорнюк А.И. Использование DOCKER для создания обособленной исследовательской среды. В данной работе было рассмотрено применение технологии контейнеризации для создания независимой среды для проведения исследований в сфере сравнения быстродействия различных баз данных. В частности, были рассмотрены характерные особенности создания образов и контейнеров. Проведено сравнение с другой технологии виртуализации. Определены основные преимущества использования технологии Docker.
Ключевые слова: Docker, контейнеризация, CI / CD, особенности использования.

1. Постановка проблеми. Під час проведення досліджень у галузі програмної інженерії або розробки програмних продуктів виникає необхідність створення різноманітних програмних середовищ. Наприклад, існує серверний проект, розробку якого завершено і тепер його необхідно передати користувачеві. Під час розгортання проекту виникають складнощі, пов'язані з відмінністю апаратної і програмної складових між середовищем розробки та кінцевим серверним середовищем.[1, 2, 8] Все ускладнюється якщо продукт тиражований і замість одного клієнта наявні сотні або тисячі клієнтів з різними конфігураціями серверів. Часом інфраструктура проектів може бути дуже специфічною, а інструкція по налаштуванню - дуже мізерною або, як варіант, відсутньою зовсім. На налаштування витрачається невиправдано велика кількість корисного часу. У такі ситуації часто потрапляють нові члени команди (особливо, якщо вони працюють віддалено), компанії, які беруть на підтримку існуючий проект. Крім цього, проблеми з налаштуванням оточення зачіпають розробників, які працюють на декількох проектах з різною інфраструктурою. Також, при налаштуванні середовища необхідно налаштовувати велику кількість залежностей, особливо при необхідності одночасної підтримки різних версій цих залежностей. Це сильно «забруднює» систему, виникає можливість конфліктів між різними версіями застосунків. За допомогою docker є можливість відокремити власне додаток від інфраструктури і поводитися з інфраструктурою як з керованим додатком. Уявімо, що існує проект на 7-8 компонентів, нехай це буде frontend, backend, content-api, mysql, redis, mongodb, elastic, rabbitmq. Раніше всі ці сервіси безпосередньо встановлювалися і налаштовувалися на машині розробника, іноді на це могло піти від декількох годин до декількох днів. Більш того з'являлася проблема оновлень, додаток працює тільки з mongodb 2.4 а розробник випадково оновив до 4, погодьтеся, налаштовувати все це досить довго навіть для людей, які цим займаються постійно. Вихід було знайдено в docker, замість установки всіх компонентів ми просто можемо завантажити образи контейнерів і запустити їх. Це дуже зручно, прийшов новий розробник в команду і замість декількох днів налаштувань він просто запустив 2-3 команди і все, оточення готове. Аналогічно і з тестуванням, наприклад вам потрібно протестувати додаток на версіях PHP 5.6, 7.0, 7.1, 7.2. Більше вам не потрібно коженого разу заново інсталювати PHP або одночасно тримати кілька різних версій постійно змінюючи посилання, просто передаєте змінну з потрібною версією PHP в докер і він сам завантажує потрібний образ. На тестових середовищах вже все менше використовують системи підтримки релізів суть яких полягала в створенні папок з номером релізу, вивантаженням туди коду і лінкуванням в основну папку на яку дивився веб-сервер, зараз можна упакувати потрібний код в контейнер і просто його розгорнути на потрібному сервері в 2 команди.

Docker допомагає викладати код швидше, швидше тестувати, швидше викладати додатки і зменшити час між написанням коду і його запуском. Docker робить це за допомогою легкої платформи контейнерної віртуалізації, використовуючи процеси і утиліти, які допомагають керувати і викладати програми.[3, 5, 6]

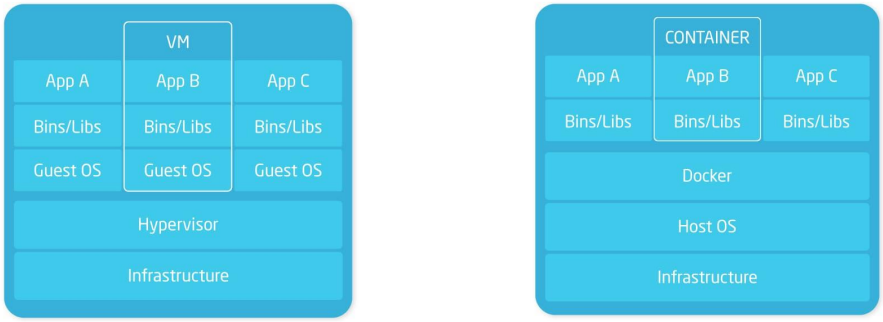
Платформа і засоби контейнерної віртуалізації можуть бути корисні в таких випадках:

- пакування застосунку (і так само використовуваних компонентів) в docker контейнери;
- роздача і доставка цих контейнерів командам для розробки і тестування;
- викладання контейнерів на продакшени, як в дата-центри так і в хмари.

2. Аналіз і порівняння схожих рішень. Протягом деякого часу Vagrant є рішенням для створення середовищ розробки, які можна налаштувати незалежно від вашої машини та спільно з командою. Існує багато переваг використання віртуальних машин над інсталяцією програмного забезпечення безпосередньо на вашому локальному комп'ютері (наприклад, LAMP):

- Програмні стеки повністю незалежні від машини, на якій ви працюєте.
- Програмні стеки можна спільно використовувати з іншими людьми та автоматично відтворювати з легкістю.
- Віртуальні машини можна запускати і зупиняти за потребою.[4, 8, 11]

Хоча Docker також працює на віртуальній машині, він працює інакше. Vagrant використовує набагато простішу архітектуру, ніж Docker. Він використовує віртуальні машини для запуску середовищ, незалежних від хост-машини. Це робиться за допомогою так званого програмного забезпечення для віртуалізації, такого як VirtualBox або VMware. Кожне середовище має власну віртуальну машину і налаштовується за допомогою Vagrantfile. Vagrantfile повідомляє Vagrant, як налаштувати віртуальну машину і які сценарії потрібно запускати, щоб забезпечити середовище. Недоліком цього підходу є те, що кожна віртуальна машина включає не тільки ваше додаток і всі його бібліотеки, але й всю гостьову операційну систему, яка може мати розміри в десятки гігабайт. Docker, однак, використовує "контейнери", які включають вашу програму і всі її залежності, але спільно використовують ядро (операційну систему) з іншими контейнерами. Контейнери виконуються як ізольовані процеси на операційній системі хоста, але не прив'язані до будь-якої конкретної інфраструктури (вони можуть працювати на будь-якому комп'ютері).



Малюнок: порівняння віртуалізації та контейнеризації.

Vagrant є інструментом, орієнтованим на забезпечення послідовного робочого середовища розробки в декількох операційних системах. Docker - це контейнерне управління, яке може послідовно запускати програмне забезпечення, доки існує система контейнеризації. Контейнери, як правило, більш легкі, ніж віртуальні машини, тому пуск і зупинка контейнерів надзвичайно швидкі. Docker використовує рідну функціональність контейнерів на MacOS, Linux і Windows. В даний час Docker не має підтримки для деяких операційних систем (наприклад, BSD). Якщо ваше цільове розгортання є однією з цих операційних систем, Docker не надаватиме однакового паритету виробництва як інструмент Vagrant. Vagrant дозволить вам також запускати середовище розробки Windows, Mac або Linux. Для мікросервісних важких середовищ, Docker може бути привабливим, оскільки ви можете легко запустити одну Docker VM і почати багато контейнерів вище, що дуже швидко. Це хороший випадок використання для Docker. Vagrant може зробити це також з постачальником Docker. Основною перевагою Vagrant є послідовний робочий процес, але є багато випадків, коли робочий процес чистого Docker має сенс. І Vagrant, і Docker мають велику бібліотеку "образів" або "коробок", що надаються спільнотою.[6, 7, 14]

Vagrant від Hashicorp - це рішення, яке дозволяє швидко конфігурувати та надавати віртуальні машини (VM), які допомагають ізолювати додаток у власному середовищі розробки. Віртуальні машини працюють на реальних серверах з різною апаратною конфігурацією, але емуляція віртуальної інфраструктури, якої потребує розробник - від операційної системи до бібліотек і бінарних файлів необхідного програмного забезпечення - гарантує, що програма працює одинаково незалежно від апаратного та програмного забезпечення серверів. Docker - це платформа з відкритим вихідним кодом, яка дозволяє ізолювати програми в кодових контейнерах, подібних до контейнерів Linux (LXC), хоча Docker перейшов з LXC до containerd, щоб забезпечити стандартизацію в галузі. Замість створення віртуального комп'ютера на вершині реального апаратного забезпечення, контейнер Docker є кодовим пакетом з усім необхідним для запуску коду програми всередині. Хоча для створення однієї програми створено єдиний контейнер, один Vagrant VM може одночасно запускати декілька взаємодіючих програм. Для контейнерів Docker це можливо за допомогою Kubernetes і docker-compose. Таким чином, контейнери, як правило, важать кілька десятків МБ, і кілька контейнерів з окремими додатками можуть працювати на вершині однієї віртуальної машини.[8, 12, 13]

Оскільки контейнери Docker виконуються в окремих пакетах коду, їх можна запускати за хвилину, не впливаючи на роботу інших компонентів системи. Vagrant VM повинні бути запущені на існуючих ОС, тому вони використовують певну кількість ресурсів для своєї підтримки. Docker зводить нанівець необхідність в гіпервізорі і всьому рівні віртуалізації, оскільки контейнери можна запускати безпосередньо на вершині 3 найпопулярніших ОС - Linux, Windows і Mac OS. Це дозволяє значно підвищити ефективність споживання ресурсів - хост, що працює під управлінням контейнерів, може похвалитися на 300% підвищеною ефективністю порівняно з запущеними віртуальними машинами.

Vagrant легший для розуміння і легше налаштовується, але може бути дуже ресурсомістким (з точки зору оперативної пам'яті та простору). Архітектуру Docker важче зрозуміти, але вона є набагато швидшою, використовує набагато менше ресурсів, в тому числі дискового простору.

Docker - це виробничо-готове середовище, яке забезпечує послідовний досвід використання програми на всьому конвеєрі доставки програмного забезпечення. Побудована за галузевими стандартами, Docker-контейнерізація стає важливим інструментом для забезпечення безперервності доставки програмного забезпечення, особливо в поєднанні з Kubernetes, Jenkins та іншими інструментами DevOps для CI/CD. З іншого боку, Vagrant спеціально розроблений для розробки програмного забезпечення, тому запуск його у виробництво не рекомендується через всі описані вище причини – надмірне споживання ресурсів, повільну швидкість запуску/перезавантаження, складнощі конфігурації безпеки.

3. Обговорення особливостей роботи з Docker.

Щоб створити власний образ, необхідно створити файл Dockerfile в кореневому каталозі проекту. Це буде виглядати так:

```
FROM ubuntu:18.04
MAINTAINER Andrii
RUN echo 'debconf debconf/frontend select Noninteractive' | debconf-set-selections
RUN apt-get update
RUN apt-get install -y python-pip python-dev python-lxml libxml2-dev libxslt1-dev libxslt-dev libpq-dev zlib1g-dev && apt-get build-dep -y python-lxml && apt-get clean
ADD requirements.txt requirements.txt
RUN pip install -r requirements.txt
WORKDIR /project
EXPOSE 80
```

Команда FROM визначає, який образ береться за основу. MAINTAINER – ім'я автора. Команда RUN виконує інструкції, інструкція ADD копіює нові файли, каталоги або віддалені URL-адреси файлів з <src> і додає їх до файлової системи зображення по шляху <dest>. Кілька ресурсів <src> можуть бути вказані, але якщо вони є файлами або каталогами, їх шляхи інтерпретуються як відносно джерела контексту побудови. Інструкція WORKDIR встановлює робочий каталог для будь-яких інструкцій RUN, CMD, ENTRYPOINT, COPY і ADD, які слідують за ним у файлі Docker. Якщо WORKDIR не існує, вона буде створена, навіть якщо вона не використовується в будь-якій наступній інструкції Dockerfile. Інструкція WORKDIR може використовуватися кілька разів у файлі Docker. Якщо заданий відносний

шлях, він буде відносно шляху попередньої команди WORKDIR. Інструкція EXPOSE повідомляє Docker про те, що контейнер слухає на вказаних мережевих портах під час виконання. Можна вказати, чи прослуховуватиме порт на TCP або UDP, а за замовчанням – TCP, якщо протокол не вказано. Інструкція EXPOSE не публікує порт. Він функціонує як тип документації між особою, яка створює зображення, і особою, яка керує контейнером, про які порти призначені для публікації. Щоб фактично опублікувати порт під час запуску контейнера, використовуйте прапорець -p для запуску докера для публікації та зіставлення одного або декількох портів, або прапора -P для публікації всіх відкритих портів і зіставлення їх з портами високого рівня.[10, 12, 15]

Після створення Dockerfile необхідно зібрати образ, використовуючи команду:

```
docker build -t username/image
```

Тут username - ваше ім'я користувача Dockerhub і image – назва вашого нового образу для даного проекту. Коли успішно вдалося створити основний образ, краще завантажити його в хмару DockerHub. Це можна зробити командою docker push username / image.

Отже, на даному етапі ми можемо запустити наш контейнер з додатків Django, але нам також потрібно запустити деякі інші контейнери с Redis, базою даних PostgreSQL і Celery Worker. Щоб спростити цей процес, ми скористаємося технологією Docker Compose, яка дозволяє нам створити простий файл YML з інструкціями про те, які контейнери запускати і як зв'язувати їх між собою. Створимо цей файл і назвемо його за замовчуванням docker-compose.yml:

```
django:
  image: username/image:latest
  command: python manage.py supervisor
  environment:
    RUN_ENV: "$RUN_ENV"
  ports:
   - "80:8001"
  volumes:
   - .:/project
  links:
   - redis
   - postgres

celery_worker:
  image: username/image:latest
  command: python manage.py celery worker -l info
  links:
   - postgres
   - redis

postgres:
  image: postgres:9.1
  volumes:
    - local_postgres:/var/lib/postgresql/data
  ports:
   - "5432:5432"
  environment:
    POSTGRES_PASSWORD: "$POSTGRES_PASSWORD"
    POSTGRES_USER: "$POSTGRES_USER"

redis:
  image: redis:latest
  command: redis-server --appendonly yes
```

Як ви бачите, ми запустимо чотири контейнери під назвами django, celery_worker, postgres і redis. Ці назви важливі для нас. Отже, по-перше, наш файл завантажить образ Redis з dockerhub і запустить з нього контейнер. По-друге, він завантажить образ Postgres і запустить контейнер із закріпленими даними з радела local_postgres. Потім, цей файл запустить контейнер з нашим додатком Django, направить 8001-й порт зсередини на 80-й зовні, зв'яже ваш поточний каталог прямо з папкою / project всередині контейнера, пролінкує його з контейнерами Redis і Postgres і запустить супервайзер. І останнє, але не менш важливе – наш контейнер з celery worker, який також з'єднаний з Postgres і Redis. Ви можете направити будь-яку кількість портів, якщо необхідно - для цього просто додайте нові записи до відповідного розділу. Також ви можете з'єднати будь-яку кількість контейнерів, наприклад, контейнер з вашою базою даних. Використовуйте тег: latest для автоматичної перевірки оновленого образу на DockerHub.[11, 12, 14]

Для того, щоб запустити контейнери, слід виконати команду

```
docker-compose up
```

Отже, образи Docker - це шаблони для ваших контейнерів. Вони розроблені для того, щоб бути ефективними і пропонувати максимальне повторне використання, використовуючи драйвер зберігання файлової системи накладення. Для початку роботи з Docker ви можете створити "інструкцію по збірці" певного оточення (наприклад, Debian + Apache + modPHP + PostgreSQL або Ubuntu + nginx + PHP-fpm + MySQL). За інструкцією створити "заготовку" системи і після використовувати її стільки разів, скільки буде потрібно. Можна навіть поділитися такою "заготовкою" з колегами, які працюють з вами в одному проекті і яким потрібне таке ж середовище. Таку "заготовку" називають образом або Docker image. Образ створюється за інструкціями, які записуються в спеціальний файл - Dockerfile. Після того, як у вас є образ, Docker може запустити на основі образу контейнер. Контейнер - це самостійна "одиниця" всередині поточної операційної системи. Він складається з операційної системи, встановлених розширень, призначених для користувача файлів, також контейнер може запускати процеси. Чимось Docker схожий на віртуальну машину, з тією різницею, що працює "поверх" операційної системи хост-машини.[2, 4, 6, 9]

4. Висновки. У даній роботі було розглянуто технологію контейнеризації Docker, яка часто використовується для створення програмних середовищ для розробки, відлагодження, тестування та впровадження програмних продуктів. Під час проведення дослідження було висвітлено особливості роботи з Dockerfile та docker-compose файлами, створення образів та контейнерів, налаштування взаємодії контейнерів між собою. Відмічається, що процес контейнеризації значно спрощує розгортання та підтримку різноманітних програмних оточень, а також значно зменшує накладні витрати на віртуалізацію.

1. James Turnbull The Docker Book: Containerization Is the New Virtualization
2. Jeff Nickoloff Docker in Action
3. Adrian Mouat Using Docker: Developing and Deploying Software with Containers
4. Nigel Poulton Docker Deep Dive
5. S. Goasguen Docker Cookbook: Solutions and Examples for Building Distributed Applications
6. Karl Matthias, Sean P. Kane Docker: Up and Running
7. https://www.docker.com/
8. https://www.vagrantup.com/
9. https://www.vagrantup.com/docs/vagrantfile/
10. https://proglib.io/p/docker/
11. https://habr.com/ru/post/353238/
12. https://habr.com/ru/post/309556/
13. https://vps.ua/blog/docker-and-linux-containers/
14. https://blog.maddevs.io/docker-for-beginners-a2c9c73e7d3d
15. https://dou.ua/lenta/articles/docker/