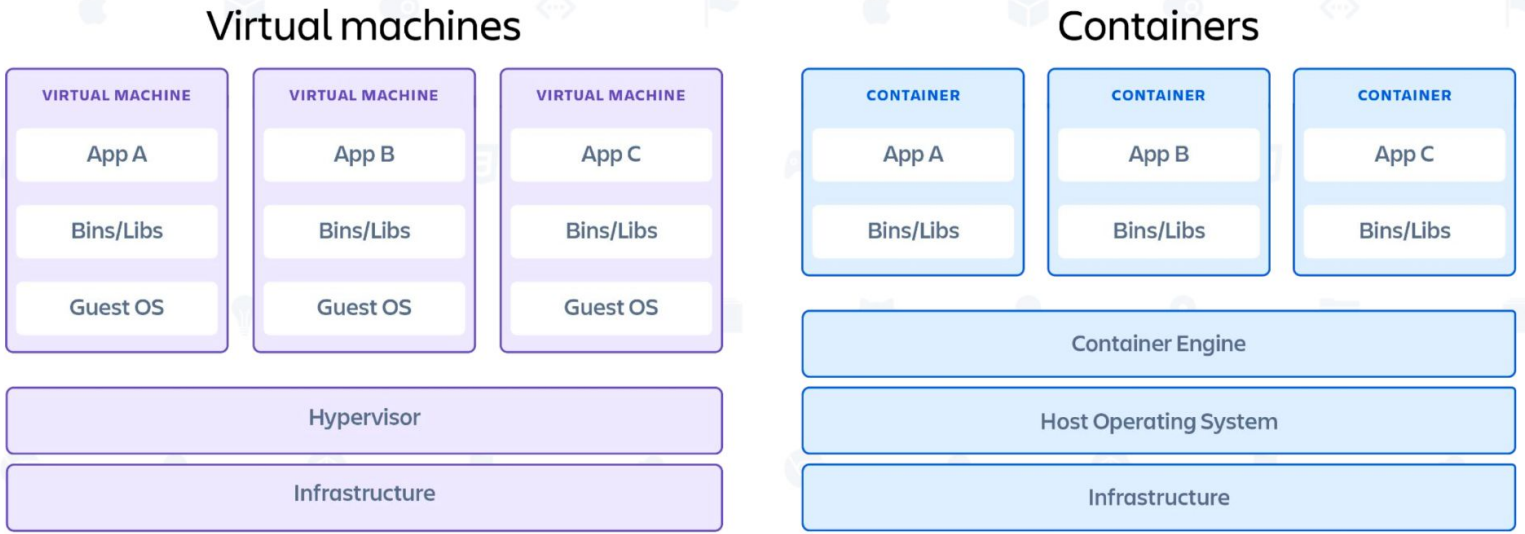
Lesson 38

24.04.2024

Контейнери та віртуальні машини – дуже схожі між собою технології віртуалізації ресурсів. Віртуалізація - це процес, при якому один системний ресурс, такий як оперативна пам'ять, ЦП, диск або мережа, може бути віртуалізований і представлений у вигляді багатьох ресурсів. Основна відмінність контейнерів і віртуальних машин полягає в тому, що віртуальні машини віртуалізують весь комп'ютер аж до апаратних рівнів, а контейнери — лише програмні рівні вище за рівень операційної системи.




ORACLE
VirtualBox


vmware


Microsoft
Hyper-V


docker

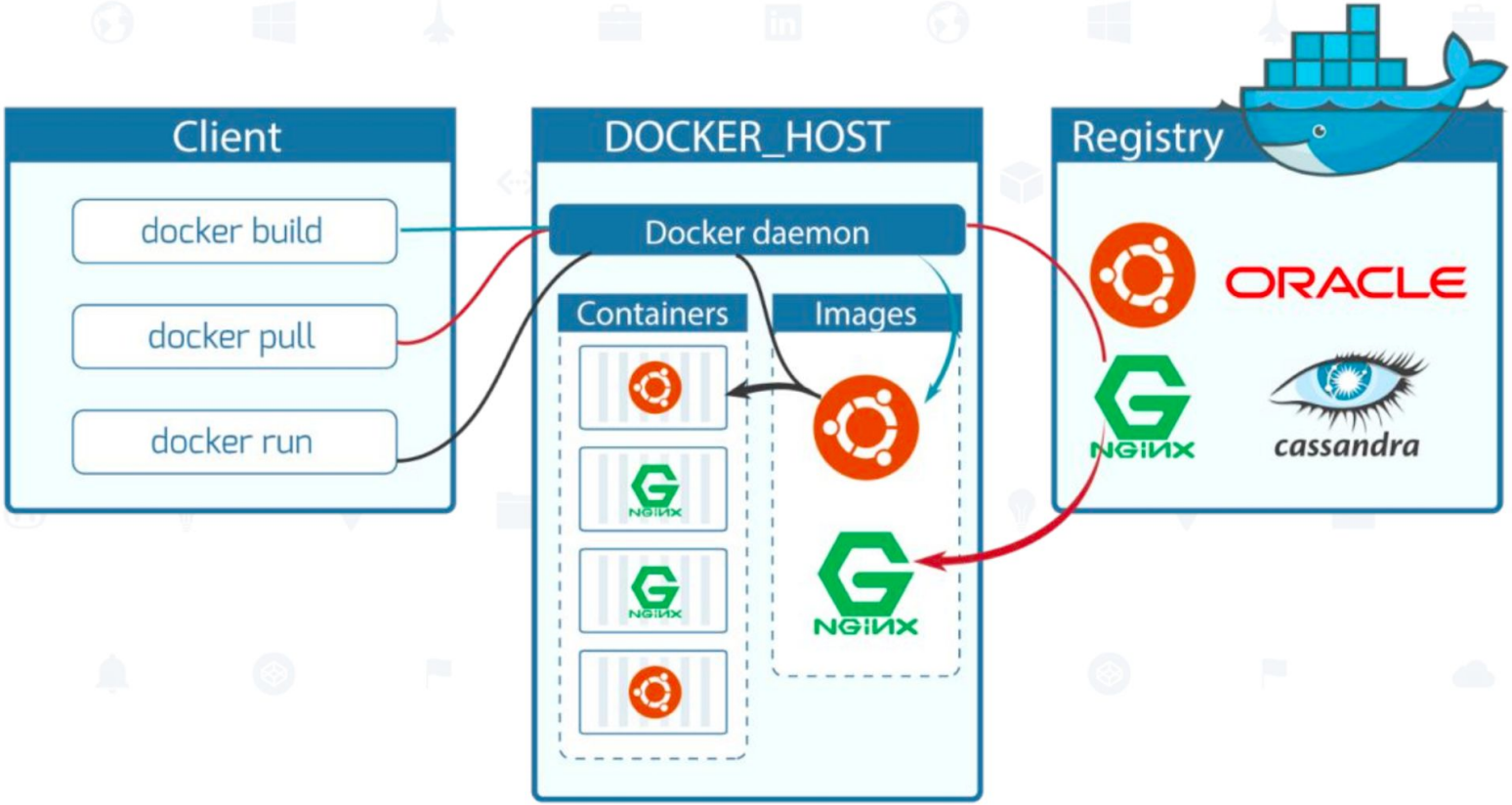
Docker — програмне забезпечення для автоматизації розгортання та керування програмами в середовищах з підтримкою контейнеризації, контейнеризатор додатків.

У своєму ядрі docker дозволяє запускати практично будь-яку програму, безпечно ізольовану в контейнері. Безпечна ізоляція дозволяє запускати на одному хості багато контейнерів одночасно. Легковажна природа контейнера, який запускається без додаткового навантаження гіпервізора, дозволяє вам домагатися більше від заліза.

Платформа та засоби контейнерної віртуалізації можуть бути корисними у таких випадках:

- упаковка вашої програми (і так само використовуваних компонентів) в контейнери docker;
- роздача та доставка цих контейнерів вашим командам для розробки та тестування;
- викладання цих контейнерів на ваші продакшени, як у дата центри так і в хмари.

DOCKER COMPONENTS



Docker-демон

Як показано на діаграмі, демон запускається на хост-машині. Користувач не взаємодіє із сервером на пряму, а використовує для цього клієнт.

Docker-клієнт

Docker-клієнт, програма docker – головний інтерфейс до Docker. Вона отримує команди від користувача та взаємодіє з docker демоном.

Усередині docker-а

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

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

Образи

Docker-образ - це read-only шаблон. Наприклад, образ може містити операційну систему Ubuntu с Apache та додатком на ній. Образи використовуються для створення контейнерів. Docker дозволяє легко створювати нові образи, оновлювати існуючі або ви можете скачати образи, створені іншими людьми. Образи - це компонент складання docker-a.

Реєстр

Docker-реєстр зберігає образи. Існують громадські і приватні реєстри, у тому числі можна завантажити чи завантажити образи. Публічна Docker-реєстр - це Docker Hub. Там зберігається величезна колекція образів. Як ви знаєте, образи можуть бути створені вами чи ви Ви можете використовувати образи, створені іншими. Реєстри – це компонента поширення.

Контейнери

Контейнери схожі на директорію. У контейнерах міститься все, що потрібне для роботи програми. Кожен контейнер створюється з образу. Контейнери можуть бути створені, запущені, зупинені, перенесені або вилучені. Кожен контейнер ізольований і є безпечною платформою для застосування. Контейнери – це компонент роботи

Файл Dockerfile

Файл Dockerfile містить набір інструкцій, дотримуючись яких Docker збиратиме образ контейнера. Цей файл містить опис базового образу, який буде вихідним шаром образу. Серед популярних офіційних базових образів можна відзначити python, ubuntu, alpine.
У образ контейнера поверх базового образу можна додавати додаткові шари. Робиться це відповідно до інструкцій з Dockerfile. Наприклад, якщо Dockerfile описує образ, який планується використовувати для вирішення задач машинного навчання, то в ньому можуть бути інструкції для включення до проміжного шару такого образу бібліотек NumPy, Pandas та Scikit-learn.

І, нарешті, в образі може міститися, поверх решти, ще один тонкий шар, дані, що зберігаються в якому, піддаються зміни. Це невеликий за обсягом шар, що містить програму, яку планується запускати в контейнері.

Docker Compose

Docker Compose - це інструмент, який спрощує розгортання додатків, для роботи яких потрібно кілька контейнерів Docker. Docker Compose дозволяє виконувати команди, які описуються у файлі docker-compose.yml. Ці команди можна виконувати стільки разів, скільки потрібно. Інтерфейс командного рядка Docker Compose спрощує взаємодію з багатоконтейнерними програмами. Цей інструмент встановлюється під час встановлення Docker

Дюжина інструкцій Dockerfile

1. FROM - задає базовий (батьківський) образ.
2. LABEL - описує метадані. Наприклад, відомості про те, хто створив і підтримує образ.
3. ENV - встановлює постійні змінні середовища.
4. RUN – виконує команду та створює шар образу. Використовується для встановлення контейнера пакетів.
5. COPY — копіює у контейнер файли та папки.
6. ADD - копіює файли і папки в контейнер, може розпаковувати локальні .tar-файли.
7. CMD - описує команду з аргументами, яку потрібно виконати, коли контейнер буде запущений. Аргументи можуть бути перевизначено під час запуску контейнера. У файлі може бути лише одна інструкція CMD.
8. WORKDIR – задає робочу директорію для наступної інструкції.
9. ARG — задає змінні передачі Docker під час складання образу.
10. ENTRYPOINT — надає команду з аргументами виклику під час виконання контейнера. Аргументи не перевизначаються.
11. EXPOSE – вказує на необхідність відкрити порт.
12. VOLUME – створює точку монтування для роботи з постійним сховищем

Dockerfile_app

```
FROM openjdk:17
MAINTAINER oStepurko
EXPOSE 8080
COPY target/ExhangerMarket-spring-boot.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]

```

Dockerfile_mysql

```
FROM mysql:latest
MAINTAINER oStepurko
EXPOSE 3306
ENV MYSQL_DATABASE=exhangeMarket \
    MYSQL_ROOT_PASSWORD=rootroot

ADD init.sql /docker-entrypoint-initdb.d
```

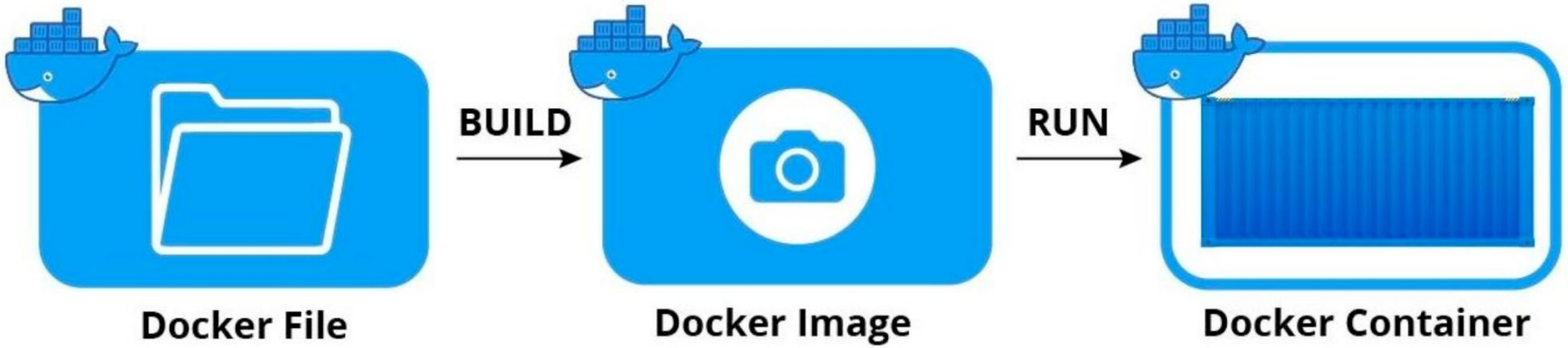
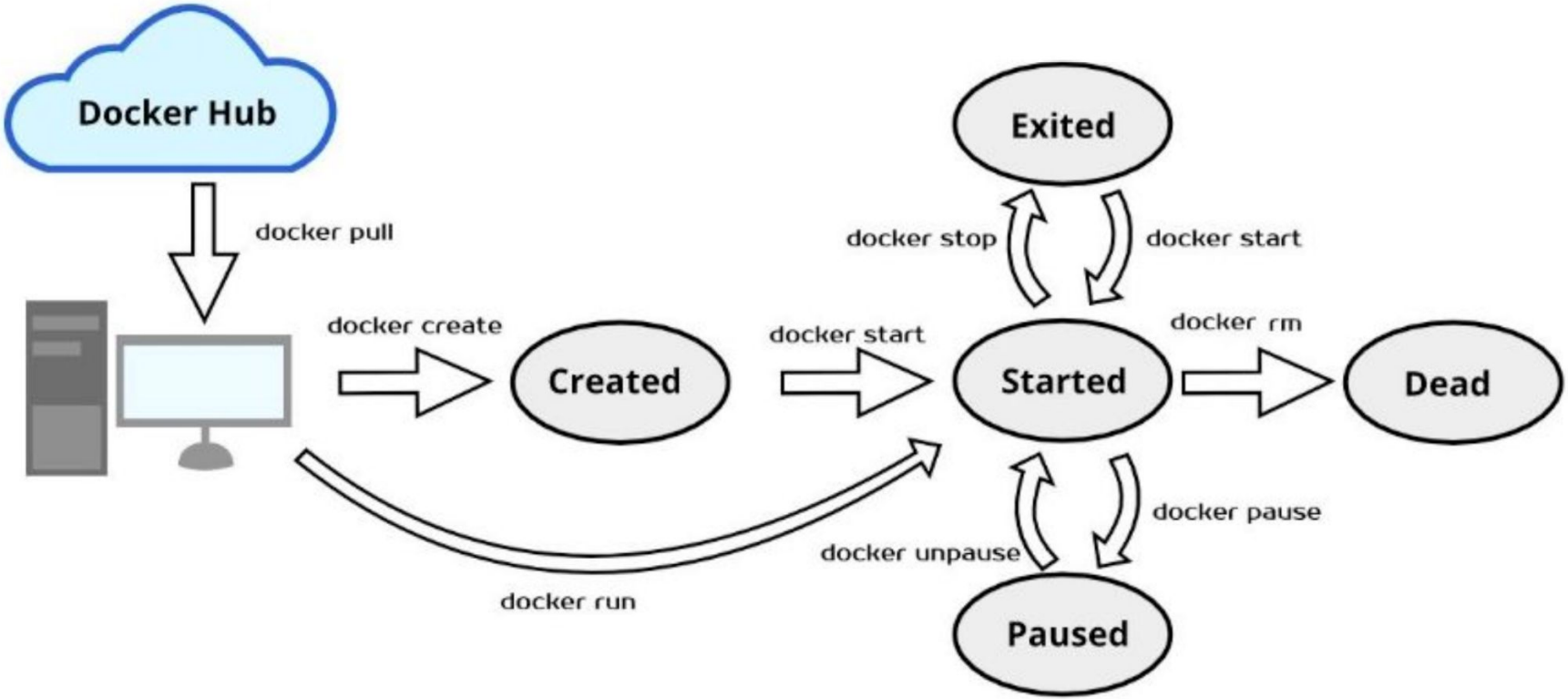

BUILD
RUN
Docker File
Docker Image
Docker Container

Схема Lifecycle of Docker Container



Create container
Створіть контейнер, щоб потім запустити його з потрібним чином.
docker create --name <container-name> <image-name>

Run docker container
Запустіть контейнер докеру з необхідним чином та вказаною командою/процесом. Прапор -d використовується для запуску контейнера фоновий режим.
docker run -it -d --name <container-name> <image-name> bash

Pause container
Використовується для зупинення процесів, запущених усередині контейнера.
docker pause <container-id/name>

Unpause container
Використовується для відновлення процесів усередині контейнера.
docker unpause <container-id/name>

Start container
Запустіть контейнер, якщо він перебуває у зупиненому стані.
docker start <container-id/name>

Stop container
Зупинити контейнер та процеси, запущені всередині контейнера:
docker stop <container-id/name>
Щоб зупинити всі запущені контейнери докерів
docker stop $(docker ps -a -q)

Restart container

Використовується для перезапуску контейнера та процесів, що працюють усередині контейнера.

docker restart <container-id/name>

Kill container

Ми можемо вбити працюючий контейнер.

docker kill <container-id/name>

Destroy container

Краще знищувати контейнер, тільки якщо він перебуває у зупиненому стані, замість примусово знищувати контейнер. працюючий контейнер.

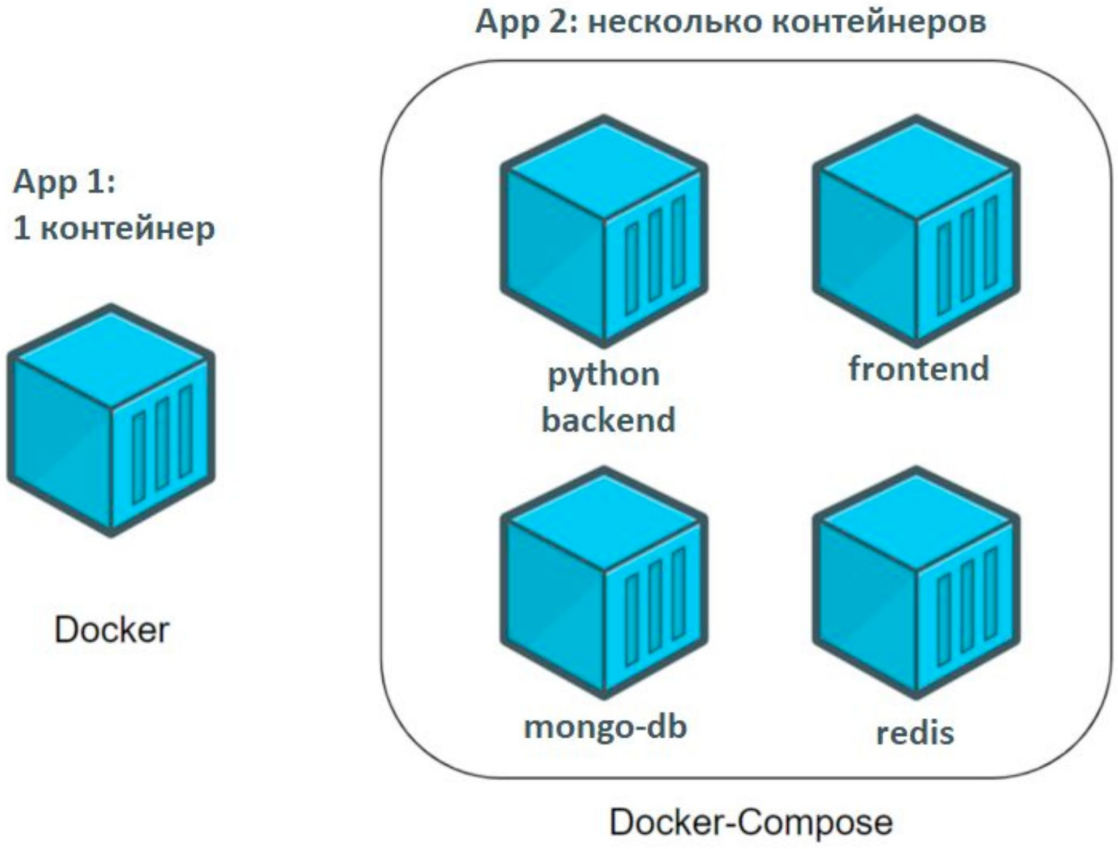
docker rm <container-id/name>

Щоб видалити всі зупинені контейнери докерів

docker rm $ (docker ps -q -f status = exited)

Що таке Docker Compose?

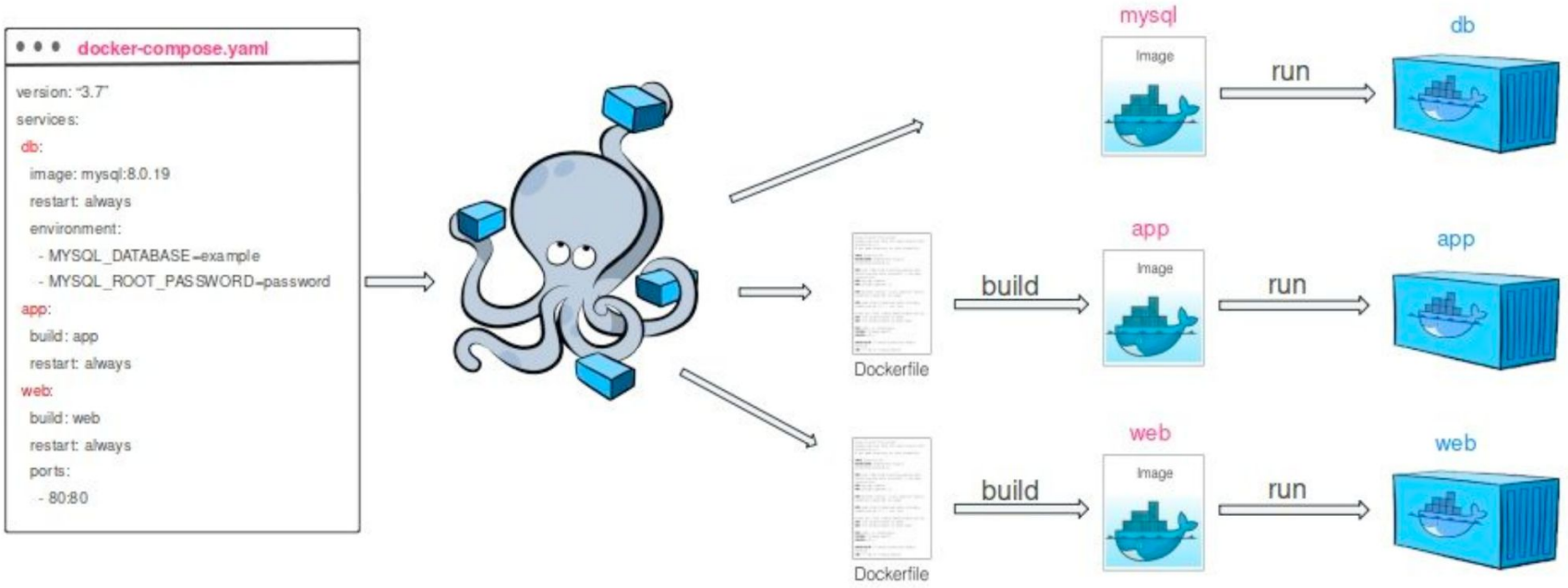
Docker Compose — це інструмент, який спрощує запуск програм, що складаються з кількох контейнерів.



Docker Compose дозволяє записувати команди в файл docker-compose.yml для повторного використання. Інтерфейс командної рядки Docker Compose (cli) полегшує взаємодію з вашим багатоконтейнерним додатком.

docker-compose.yml

```yaml
version: "3.7"
services:
  ex-market:
    depends_on:
      - mysqldb
    command: sh -c './wait-for mysqldb:3306 -- npm start'
    build:
      context: .
      dockerfile: Dockerfile_app
    ports:
      - 8080:8080
    links:
      - "mysqldb:app_db"
  mysqldb:
    build:
      context: .
      dockerfile: Dockerfile_mysql
    ports:
      - 3306:3306
    env_file: .env
    volumes:
      - myapp:/home/node/app
volumes:
  myapp:
```


docker-compose.yaml
version: "3.7"
services:
db:
image: mysql:8.0.19
restart: always
environment:
- MYSQL_DATABASE=example
- MYSQL_ROOT_PASSWORD=password
app:
build: app
restart: always
web:
build: web
restart: always
ports:
- 80:80
mysql
Image
run
db
Dockerfile
build
app
Image
run
app
Dockerfile
build
web
Image
run
web

Системи управління базами даних

СУБД - це загальний термін, що стосується всіх видів абсолютно різних інструментів, від комп'ютерних програм до вбудованих бібліотек. Ці програми керують чи допомагають керувати наборами даних. Так як ці дані можуть бути різного формату та розміру, було створено різні види СУБД.

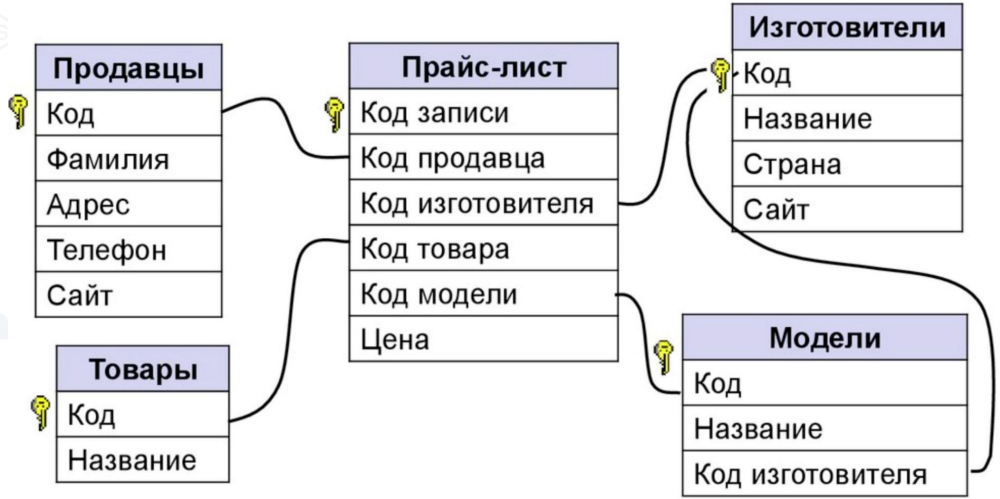
СУБД засновані на моделях баз даних - певних структурах обробки даних. Кожна СУБД створена для роботи з однією з них з урахуванням особливостей операцій над інформацією.

Хоча рішень, що реалізують різні моделі баз даних, дуже багато, періодично деякі з них стають дуже популярними та використовуються протягом багатьох років. Нині найпопулярнішою моделлю є реляційна система управління базами даних (РСУБД).

Реляційна модель

Представлена у 70-х, реляційна модель пропонує математичний спосіб структуризації, зберігання та використання даних. Відносини (англ. relations) дають можливість групування даних як пов'язаних наборів, представлених у вигляді таблиць, що містять упорядковану інформацію (наприклад, ім'я та адресу людини) та відповідні значення та атрибути (його номер паспорти).

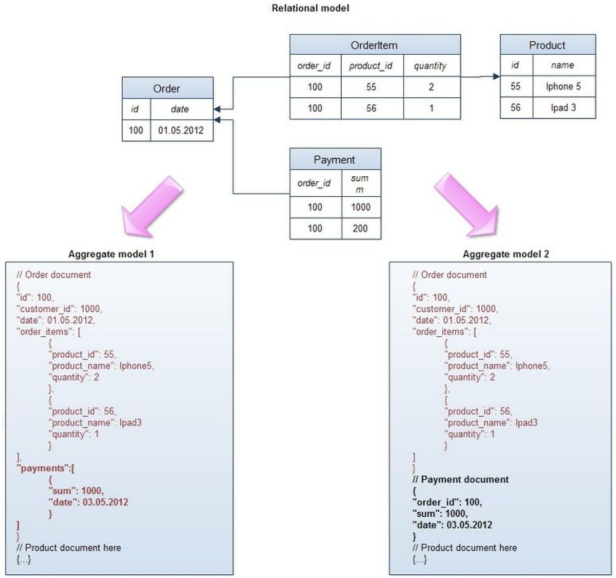
Завдяки десятиліттям досліджень та розробки РСУБД працюють продуктивно та надійно. У поєднанні з великим досвідом використання адміністраторами реляційних баз даних стали вибором, що гарантує захист інформації від втрат.



Безмодельний (NoSQL) підхід

NoSQL-спосіб структуризації даних полягає у позбавленні обмежень при зберіганні та використанні інформації. Бази даних NoSQL, використовуючи неструктуризований підхід, пропонують багато ефективних способів обробки даних в окремих випадках (наприклад, під час роботи зі сховищем текстових документів).

На відміну від реляційної моделі, яка зберігає логічну бізнес-сутність програми у різні фізичні таблиці в 3 метою нормалізації, NoSQL сховища оперують із цими сутностями як із цілісними об'єктами:

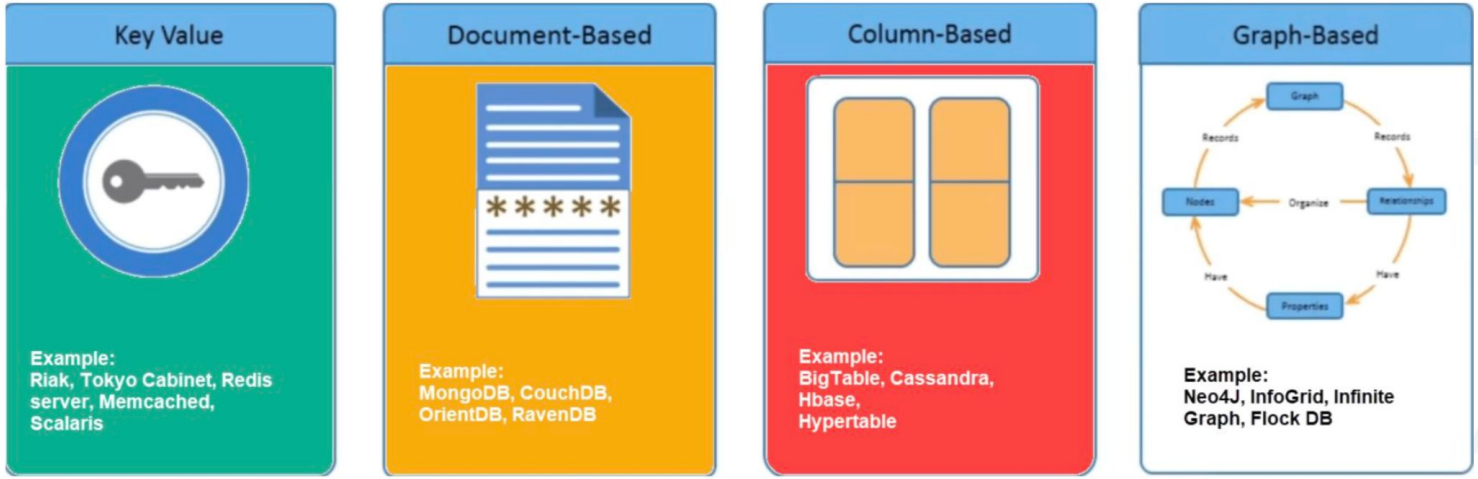


NoSQL-СУБД

NoSQL-СУБД не використовують реляційну модель структуризації даних. Існує багато реалізацій, що вирішують це питання по-своєму, часто дуже специфічно. Ці безсхемні рішення допускають необмежене формування записів та зберігання даних у вигляді ключ-значення.

На відміну від традиційних РСУБД, деякі бази даних NoSQL, наприклад MongoDB, дозволяють групувати колекції даних з іншими базами даних. Такі СУБД зберігають дані як одне ціле. Ці дані можуть бути одиночним об'єктом на зразок JSON і водночас коректно відповідати на запити до полів.

NoSQL бази даних не використовують загальний формат запиту (як SQL у реляційних базах даних). Кожне рішення використовує власну систему запитів.



Порівняння SQL та NoSQL

● Структура і тип даних, що зберігаються: SQL/реляційні бази даних вимагають наявності однозначно певної структури зберігання даних, а бази даних NoSQL таких обмежень не ставлять.

● Запити: незалежно від ліцензії, РСУБД реалізують SQL-стандарти, тому з них можна отримувати дані за допомогою мови SQL. Кожна база даних NoSQL реалізує свій спосіб роботи з даними.

● Масштабованість: обидва рішення легко розтягуються вертикально (наприклад, шляхом збільшення системних ресурсів). Тим не менш, через свою сучасність, рішення NoSQL зазвичай надають простіші способи горизонтального масштабування (наприклад, створення кластера з кількох машин).

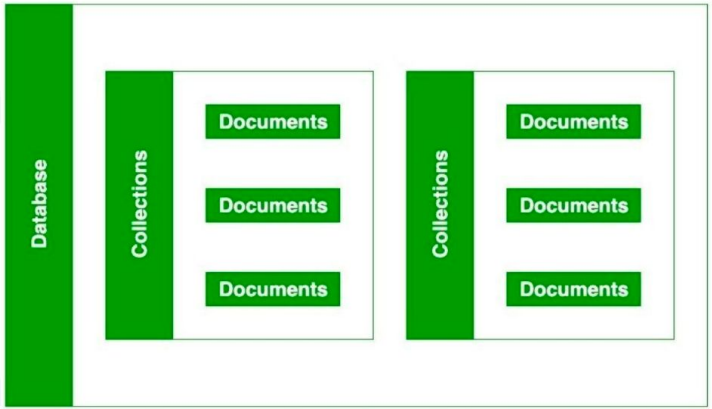
● Надійність: коли мова йде про надійність, SQL бази даних однозначно попереду.

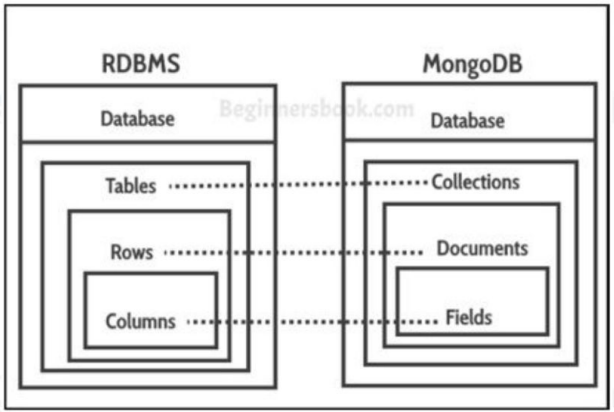
● Підтримка: РСУБД мають дуже довгу історію. Вони дуже популярні, і тому отримати підтримку, платну чи ні, дуже легко. Тому, при необхідності, вирішити проблеми з ними набагато простіше, ніж з NoSQL, особливо якщо проблема складна своєї природи (наприклад, під час роботи з MongoDB).

● Зберігання та доступ до складних структур даних: за своєю природою реляційні бази даних передбачають роботу зі складними ситуаціями, тому і тут вони перевершують NoSQL-рішення.

MongoDB

MongoDB – це база даних NoSQL. У чому на відміну від традиційних баз даних на основі SQL? По-перше, у неї немає зумовленої структури, такі як SQL таблиці. Не потрібно заздалегідь визначати, як виглядатимуть ваші таблиці. По-друге, вона не використовує термін таблиці взагалі, натомість використовується термін документи, які по суті є подібні JSON структури, і вам вирішувати, чи потрібна вам схема чи ні.





(a) Embedded Data Model

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
             phone: "123-456-7890",
             email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document

(b) Normalized Data Model