TÌM HIỂU DOCKER

Tổng quan Docker

Docker đưa ra một giải pháp mới cho vấn đề ảo hóa, thay vì tạo ra các máy ảo con chạy độc lập kiểu hypervisors, các ứng dụng sẽ đóng gói lại thành các Container
Các thành phần chính

- Docker Engine: là thành phần chính của Docker, như một công cụ để đóng gói app
- Docker Hub: là dịch vụ Cloud để chia sẻ ứng dụng và tự động hóa chuỗi các công việc liên tục có thể pull/push
- Docker Image: là một “read-only template”
- Docker registries: là kho chứa images
- Docker container: Hoạt động giống như một thư mục chứa tất cả những gì cần thiết để một ứng dụng có thể chạy được
- Dockerfile: là một file chứa tập hợp các lệnh để Docker có thể đọc và thực hiện đóng gói một image
- Orchestration: là công cụ để điều phối và quản lý nhiều containers

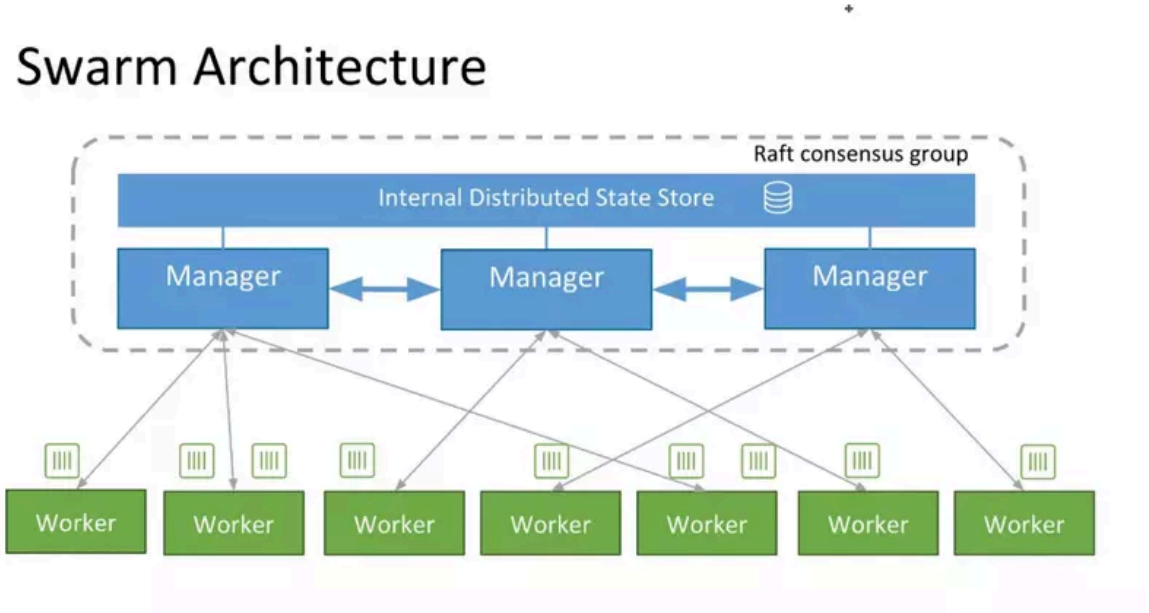
1. Docker Compose và Docker Swarm

Docker Compose là một công cụ dùng để định nghĩa và chạy các ứng dụng Docker đa container. Với Docker Compose, có thể sử dụng một tệp YAML để cấu hình các dịch vụ của ứng dụng, sau đó, với một lệnh đơn giản là có thể khởi động tất cả các dịch vụ này.
Docker Swarm là một chế độ điều phối của Docker cho phép tạo ra và quản lý một cụm (cluster) các máy Docker. Swarm cung cấp các khả năng như cân bằng tải, mở rộng tự động và tự phục hồi.
Tính năng Docker Swarm

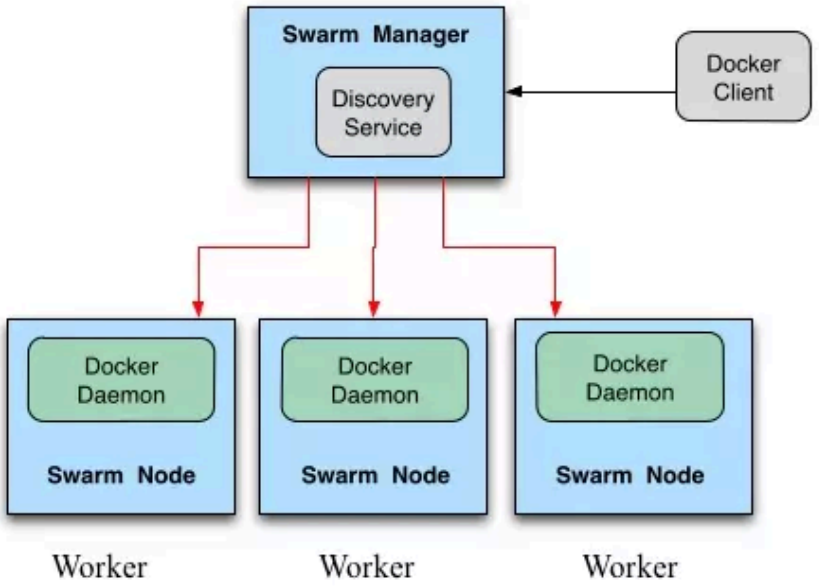
- Cluster management integrated with Docker Engine: Quản lý cluster với Docker Engine bằng việc sử dụng Docker CLI để tạo swarm.
- Decentralized design: Docker Swarm được thiết kế dạng phân cấp. Thay vì xử lý sự khác biệt giữa các roles của node tại thời điểm triển khai, Docker Engine xử lý bất kỳ chuyên môn hoá nào khi runtime. Bạn có thể triển khai cả hai loại node: managers và worker bằng Docker Engine.
- Declarative service model: Docker Engine sử dụng phương thức khai báo để cho phép bạn define trạng thái mong muốn của các dịch vụ khác nhau trong stack ứng dụng của bạn. VD: Bạn có thể mô tả ứng dụng bao gồm: web front-end với service message queueing và database back-end.
- Scaling: Đối với mỗi service bạn có thể khai báo số lượng task mà bạn muốn run. Khi bạn scale up hoặc down thì swarm manager sẽ tự động thêm hoặc xoá task để duy trì trạng thái mong muốn.
- Desired state reconciliation: Hãy hình dung bạn thiết lập một service run 10 replicas của một container và một worker machine (host/vps) đang giữ 2 trong số 10 replicas đó gặp sự cố bị crash, lúc này swarm manager sẽ tiến hành tạo thêm 2 replicas mới để thay để cho 2 replicas đã bị crash đó và tiến hành chuyển 2 replicas mới này cho các worker đang run.
- Multi-host networking: Bạn có thể chỉ định một overlay network cho các services của mình. Swarm manager sẽ tự động gán địa chỉ IP cho các container trên overlay network khi nó khởi tạo và cập nhật application.
- Service discovery: Swarm manager node gán mỗi service trong swarm một DNS duy nhất và bạn có thể truy vấn được thông qua DNS này.

- Load balancing: Có thể expose các port cho các services tới load balance để giao tiếp với bên ngoài.
- Secure by default: Các service giao tiếp với nhau thông qua giao thức bảo mật TLS. Bạn có thể tuỳ chỉnh sử dụng chứng chỉ ký tự root hoặc chứng chỉ từ một custom root CA.
- Rolling updates: Swarm giúp bạn update image của service một cách hoàn toàn tự động. Swarm manager giúp bạn kiểm soát độ trễ giữa service deploy tới các node khác nhau và bạn có thể rolling back bất cứ lúc nào.

Kiến trúc Swarm



Bao gồm các Manager và các Worker. Người dùng có thể khai báo trạng thái mong muốn của nhiều service để chạy trong Swarm sử dụng YAML files.



Swarm: là một cluster của một hoặc nhiều Docker Engine đang run (cụ thể ở đây là các node) trong chế độ Swarm, thay vì phải chạy các container bằng câu lệnh thì ta sẽ thiết lập các services để phân bổ các bản replicas tới các node.
Node: Một node là một máy vật lý hay máy ảo đang run phiên bản Docker Engine trong chế độ Swarm. Node sẽ gồm hai loại: Manager Node và Worker Node.
Manager Node: Là node nhận các define service từ user, nó quản lý và điều phối các task đến các node Worker. Theo mặc định node Manager cũng được coi là node Worker.
Worker Node: là node nhận và thực thi các task từ node Manager.
Service: Một service xác định image của container và số lượng các replicas (bản sao) mong muốn khởi chạy trong swarm.
Task: là một tác vụ mà node worker phải thực hiện. Tác vụ này sẽ do node Manager phân bổ xuống. Một task mang một Docker Container và các lệnh để chạy bên container.

Ứng dụng Docker giải quyết các vấn đề liên quan đến cơ sở dữ liệu

-

A. Đặt vấn đề

khi phát triển một web-application mà cần sử dụng nhiều loại hệ quản trị cơ sở dữ liệu như Redis, MongoDb, Mysql,... thì việc setup từng loại rất là khó. Nếu setup trên local thì có tình trạng “máy tôi chạy được máy ông không chạy được”. Ngoài ra cũng vì sử dụng nhiều loại hệ quản trị cơ sở dữ liệu nên việc đồng bộ dữ liệu rất quan trọng. Mà các tool để đồng bộ như Debezium, Kafka,... cũng rất khó để cài đặt. Đây là các vấn đề em gặp phải trong khi làm bài tập lớn môn hệ quản trị cơ sở dữ liệu. Sau khi học Docker, em đã có thể giải quyết các vấn đề nêu trên.

B. Cài đặt và xử lí các vấn đề liên quan

1. Redis

```yaml
version: '3.7'

services:
  redis:
    container_name: redis
    image: redis:latest
    ports:
      - 6379:6379
    restart: unless-stopped
    command: redis-server
```

2. MongoDb

   Ở trong Mongodb sẽ cài đặt thêm replica set. Replica set trong MongoDB là một group các mongod processes để duy trì cùng một cơ sở dữ liệu. Replica set cung cấp khả năng dự phòng và tính sẵn sàng cao.

```yaml
mongo-0-a:
  container_name: mongo-0-a
  image: mongo
  ports:
    - 127.0.10.1:27017:27017
  volumes:
    - mongo-0-a:/data/db
  restart: unless-stopped
  command: "--wiredTigerCacheSizeGB 0.25 --replSet rs0"
  networks:
    - hotelbooking

mongo-0-b:
  container_name: mongo-0-b
  image: mongo
  ports:
    - 127.0.10.2:27017:27017
  volumes:
    - mongo-0-b:/data/db
  restart: unless-stopped
  command: "--wiredTigerCacheSizeGB 0.25 --replSet rs0"
  networks:
    - hotelbooking

mongo-0-c:
  container_name: mongo-0-c
  image: mongo
  ports:
    - 127.0.10.3:27017:27017
  volumes:
    - mongo-0-c:/data/db
  restart: unless-stopped
  command: "--wiredTigerCacheSizeGB 0.25 --replSet rs0"
  networks:
    - hotelbooking
```

3. Mysql

```yaml
mysql:
  container_name: mysql8
  image: mysql:8.0
  # command: mysqld --default-authentication-plugin=mysql_native_password --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: bookingapp
    # MYSQL_USER: root
    # MYSQL_PASSWORD: root
    # MYSQL_ALLOW_EMPTY_PASSWORD: "yes"
  ports:
    - '5001:3306'
  volumes:
    -  /etc/docker/mysql:/etc/mysql/conf.d
  #   - './docker/db/data:/var/lib/mysql'
  #   - './docker/db/my.cnf:/etc/mysql/conf.d/my.cnf'
  #   - './docker/db/sql:/docker-entrypoint-initdb.d'
  networks:
    - hotelbooking
```

4. Đồng bộ dữ liệu

   Image: Sử dụng image debezium/zookeeper.

   Ports: Map các cổng từ container ra host:

   2181: Cổng dịch vụ của Zookeeper.

   2888 và 3888: Cổng giao tiếp giữa các node trong một cụm Zookeeper.

   depends_on: Xác định rằng Kafka phụ thuộc vào Zookeeper và sẽ khởi động sau khi Zookeeper đã khởi động.

```yaml
zookeeper:
  container_name: zookeeper
  image: debezium/zookeeper
  ports:
    - "2181:2181"
    - "2888:2888"
    - "3888:3888"
  networks:
    - hotelbooking
kafka:
  container_name: kafka
  image: debezium/kafka
  ports:
    - "9092:9092"
    - "29092:29092"
  depends_on:
    - zookeeper
  environment:
    ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_LISTENERS: EXTERNAL_SAME_HOST://:29092,INTERNAL://:9092
    KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka:9092,EXTERNAL_SAME_HOST://localhost:29092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,EXTERNAL_SAME_HOST:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
  networks:
    - hotelbooking
```

   image: Sử dụng image debezium/connect.

   ports: Map cổng 8083 từ container ra host.

   depends_on: Xác định rằng Connect phụ thuộc vào Kafka, Zookeeper và MySQL, sẽ khởi động sau khi các dịch vụ này đã khởi động.

```yaml
connect:
  container_name: connect
  image: debezium/connect
  ports:
    - "8083:8083"
  depends_on:
    - kafka
    - zookeeper
    - mysql
  environment:
    - BOOTSTRAP_SERVERS=kafka:9092
    - GROUP_ID=1
    - CONFIG_STORAGE_TOPIC=my_connect_configs
    - OFFSET_STORAGE_TOPIC=my_connect_offsets
    - STATUS_STORAGE_TOPIC=my_connect_statuses
    - ENABLE_DEBEZIUM_KC_REST_EXTENSION=true
    - ENABLE_DEBEZIUM_SCRIPTING=true
    - CONNECT_REST_EXTENSION_CLASSES=io.debezium.kcrestextension.DebeziumConnectRestExtension
    - ADVERTISED_HOST_NAME=localhost
  networks:
    - hotelbooking
```

container_name: Đặt tên cho container là watcher.
command: Chạy lệnh watch-topic -a -k dbserver1.bookingapp.booker để theo dõi topic Kafka dbserver1.bookingapp.booker.
depends_on: Xác định rằng Watcher phụ thuộc vào Zookeeper và Kafka, sẽ khởi động sau khi các dịch vụ này đã khởi động.

```yaml
watcher:
  image: debezium/kafka
  container_name: watcher
  command: watch-topic -a -k dbserver1.bookingapp.booker
  depends_on:
    - zookeeper
    - kafka
  networks:
    - hotelbooking
```

C. Output