Utilização de contêineres para encapsulamento de aplicações

José Matheus Mendes Trindade
21/2006334
Turma 02

I. Resumo

Este relatório apresenta o desenvolvimento de um arquivo `docker-compose.yml` que define quatro serviços interdependentes, simulando uma aplicação comercial real. Cada serviço foi cuidadosamente escolhido e configurado para interconectar com os demais, criando um sistema coeso e funcional. O relatório detalha a razão da escolha de cada serviço, explicando como eles foram configurados para trabalhar juntos e como essa configuração pode ser aplicada em um cenário de negócios real.

***Index Terms*—Streaming, Python, Sockets, Redes**

II. Introdução

A tecnologia de contêineres revolucionou o mundo do desenvolvimento de *software* e da implantação de aplicações. Contêineres, uma forma de virtualização a nível de sistema operacional, permitem que os desenvolvedores encapsulem suas aplicações juntamente com todas as suas dependências em um pacote autônomo. Este pacote, ou contêiner, pode então ser executado de forma consistente em qualquer ambiente que suporte a tecnologia de contêineres, independentemente das especificidades do sistema operacional ou do *hardware* subjacente [1].

Os contêineres oferecem várias vantagens significativas em relação aos métodos tradicionais de implantação de aplicações. Eles são leves, pois compartilham o kernel do sistema operacional host e não requerem um sistema operacional completo para cada aplicação. Além disso, eles são portáteis entre diferentes plataformas e provedores de nuvem, facilitam a orquestração e o escalonamento de aplicações e aumentam a eficiência e a utilização de recursos [3].

No entanto, apesar de suas muitas vantagens, a adoção de contêineres também apresenta desafios. Estes incluem a necessidade de novas ferramentas e habilidades, preocupações com a segurança e a complexidade do gerenciamento de contêineres em grande escala [2]. Este trabalho pretende explorar tanto as oportunidades quanto os desafios associados ao uso de contêineres para encapsulamento de aplicações.

III. Fundamentação Teórica

A tecnologia de contêineres é baseada no princípio da virtualização a nível de sistema operacional. Ao contrário da virtualização tradicional, que emula um sistema operacional completo para cada instância, a virtualização a nível de sistema operacional permite que várias instâncias (ou contêineres) compartilhem o mesmo kernel do sistema operacional. Isso resulta em contêineres que são significativamente mais leves e mais eficientes do que as máquinas virtuais tradicionais [1].

Os contêineres são isolados uns dos outros e do sistema operacional host, cada um com seu próprio espaço de usuário. Isso significa que os processos dentro de um contêiner não podem interferir nos processos em outros contêineres ou no host. Isso proporciona um alto grau de segurança e flexibilidade, permitindo que os desenvolvedores executem suas aplicações em ambientes isolados e controlados [2].

A portabilidade é outra característica chave dos contêineres. Uma vez que um contêiner é criado, ele pode ser executado em qualquer máquina que suporte a tecnologia de contêineres, independentemente do sistema operacional ou do hardware subjacente. Isso facilita a migração de aplicações entre diferentes ambientes e plataformas, desde um laptop de desenvolvedor até um cluster de produção na nuvem [3].

A orquestração de contêineres é o processo de gerenciamento de contêineres, que inclui a implantação de contêineres, o escalonamento de contêineres para atender às demandas de carga de trabalho, a garantia de que os contêineres estão funcionando corretamente e a substituição de contêineres que falham. Ferramentas de orquestração de contêineres, como o Kubernetes, fornecem um framework para gerenciar contêineres em grande escala [3].

Arquivo docker-compose.yml

A seguir está o conteúdo do arquivo `docker-compose.yml` utilizado para configurar os serviços:

```
version: '3.4'
services:
  web:
    image: httpd:latest
    ports:
      - 8080:80
  nextcloud:
    image: nextcloud:latest
```

```
    ports:
      - 8081:80
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  phpmyadmin:
    depends_on:
      - db
    image: phpmyadmin/phpmyadmin
    restart: always
    ports:
      - 8082:80
    environment:
      PMA_HOST: db
      MYSQL_ROOT_PASSWORD: somewordpress
  rocketchat:
    image: rocket.chat:latest
    restart: always
    labels:
      traefik.enable: "true"
      traefik.http.routers.rocketchat.
          ↪ rule: Host(`${DOMAIN:-}`)
      traefik.http.routers.rocketchat.
          ↪ tls: "true"
      traefik.http.routers.rocketchat.
          ↪ entrypoints: https
      traefik.http.routers.rocketchat.
          ↪ tls.certresolver: le
    environment:
      MONGO_URL: "${MONGO_URL:-\
        mongodb://${
            ↪ MONGODB_ADVERTISED_HOSTNAME
            ↪ :-mongodb}:${
            ↪ MONGODB_INITIAL_PRIMARY_PORT_
            ↪ :-27017}/\
        ${MONGODB_DATABASE:-rocketchat}?
            ↪ replicaSet=${
            ↪ MONGODB_REPLICA_SET_NAME:-
            ↪ rs0}}"
      MONGO_OPLOG_URL: "${
          ↪ MONGO_OPLOG_URL:\
        -mongodb://${
            ↪ MONGODB_ADVERTISED_HOSTNAME
            ↪ :-mongodb}:${
            ↪ MONGODB_INITIAL_PRIMARY_PORT_NUMBER
            ↪ :-27017}/\
        local?replicaSet=${
            ↪ MONGODB_REPLICA_SET_NAME:-
            ↪ rs0}}"
      ROOT_URL: ${ROOT_URL:-http://
          ↪ localhost:${HOST_PORT
          ↪ :-3000}}
      PORT: ${PORT:-3000}
      DEPLOY_METHOD: docker
      DEPLOY_PLATFORM: ${DEPLOY_PLATFORM
          ↪ :-}
      REG_TOKEN: ${REG_TOKEN:-}
    depends_on:
      - mongodb
    expose:
      - ${PORT:-3000}
    ports:
      - "${BIND_IP:-0.0.0.0}:${HOST_PORT
          ↪ :-3000}:${PORT:-3000}"
  mongodb:
    image: docker.io/bitnami/mongodb:${
        ↪ MONGODB_VERSION:-5.0}
    restart: always
    volumes:
      - mongodb_data:/bitnami/mongodb
    environment:
      MONGODB_REPLICA_SET_MODE: primary
      MONGODB_REPLICA_SET_NAME: ${
          ↪ MONGODB_REPLICA_SET_NAME:-
          ↪ rs0}
      MONGODB_PORT_NUMBER: ${
          ↪ MONGODB_PORT_NUMBER:-27017}
      MONGODB_INITIAL_PRIMARY_HOST: ${
          ↪ MONGODB_INITIAL_PRIMARY_HOST
          ↪ :-mongodb}
      MONGODB_INITIAL_PRIMARY_PORT_NUMBER
          ↪ : ${
          ↪ MONGODB_INITIAL_PRIMARY_PORT_NUMBER
          ↪ :-27017}
      MONGODB_ADVERTISED_HOSTNAME: ${
          ↪ MONGODB_ADVERTISED_HOSTNAME
          ↪ :-mongodb}
      MONGODB_ENABLE_JOURNAL: ${
          ↪ MONGODB_ENABLE_JOURNAL:-true
          ↪ }
      ALLOW_EMPTY_PASSWORD: ${
          ↪ ALLOW_EMPTY_PASSWORD:-yes}
  volumes:
    db_data: {}
    mongodb_data: { driver: local }
```

Listing 1. docker-compose.yml

Explicações Detalhadas

A seguir, forneço explicações detalhadas para cada parte do arquivo ‘`docker-compose.yml`‘:

1) **web:** Este serviço utiliza a imagem `httpd:latest`, que representa o servidor web Apache. O Apache é uma escolha comum e confiável para servir conteúdo web. O serviço é mapeado para a porta 8080 do host, permitindo acesso externo ao servidor *web*.
2) **nextcloud:** Utiliza a imagem `nextcloud:latest`, uma aplicação de código aberto para colaboração e compartilhamento de arquivos em nuvem. A escolha do Nextcloud se deve à sua robustez e capacidade de fornecer uma solução completa para armazenamento e colaboração em nuvem. O serviço é mapeado para a porta 8081 do *host*.
3) **db:** Este serviço utiliza a imagem `mysql:5.7` e cria um volume para armazenar os dados do MySQL. A escolha do MySQL como banco de dados se baseia em sua ampla adoção, confiabilidade e recursos robustos. São configuradas variáveis de ambiente necessárias para inicialização do MySQL, como senha do *root* e detalhes do banco de dados.
4) **phpmyadmin:** Este serviço depende do serviço `db` e utiliza a imagem `phpmyadmin/phpmyadmin`. O PhpMyAdmin é uma ferramenta web popular para administração de bancos de dados MySQL. A escolha visa facilitar a administração do MySQL via interface web. O serviço é mapeado para a porta 8082 do *host*.
5) **rocketchat:** Utiliza a imagem `rocket.chat:latest` e depende do serviço `mongodb`. O Rocket.Chat é uma plataforma de comunicação em equipe. A escolha do Rocket.Chat se deve à sua flexibilidade e recursos de chat. Rótulos são configurados para integração com o Traefik, um *proxy* reverso. São definidas variáveis de ambiente relacionadas ao MongoDB e à aplicação Rocket.Chat. O serviço é mapeado para a porta 3000 do *host*.
6) **mongodb:** Utiliza a imagem `docker.io/bitnami/mongodb` na versão especificada. O MongoDB é escolhido como banco de dados NoSQL para armazenar os dados do Rocket.Chat. É configurado para operar em um modo de conjunto de réplicas, garantindo alta disponibilidade e escalabilidade. O serviço é mapeado para a porta 27017 do *host*.

Razão da Escolha dos Serviços

A escolha dos serviços reflete a consideração cuidadosa das necessidades de uma aplicação comercial real. O Apache HTTP Server (web) é fundamental para hospedar conteúdo web. O Nextcloud fornece uma solução completa para armazenamento em nuvem e colaboração. O MySQL é um banco de dados confiável, enquanto o PhpMyAdmin simplifica sua administração.

O Rocket.Chat é adotado para comunicação em equipe, e o MongoDB é escolhido como banco de dados para o Rocket.Chat devido à sua natureza NoSQL, adequada para aplicações de chat escaláveis.

Configuração dos Serviços

Os serviços são configurados para interconectar-se de maneira eficiente. Por exemplo, o serviço `phpmyadmin` depende do serviço `db`, garantindo que o PhpMyAdmin tenha acesso ao banco de dados MySQL. Além disso, o Rocket.Chat depende do serviço `mongodb`, assegurando a conectividade com o banco de dados NoSQL.

A exposição de portas é estrategicamente escolhida para facilitar o acesso externo aos serviços, permitindo a interação com a aplicação de diferentes pontos.

Utilização em um Cenário de Negócios

Esse conjunto de serviços oferece uma solução abrangente para um ambiente de negócios. O Apache serve como ponto de entrada para a aplicação web, enquanto o Nextcloud proporciona colaboração e armazenamento em nuvem seguro.

A combinação do MySQL e do PhpMyAdmin simplifica a administração do banco de dados, facilitando o gerenciamento de dados críticos para a empresa. O Rocket.Chat, integrado ao MongoDB, oferece uma plataforma de comunicação interna robusta, melhorando a colaboração entre os membros da equipe.

A configuração em contêineres permite fácil implantação e escalabilidade conforme as necessidades do negócio evoluem. A utilização de conjuntos de réplicas no MongoDB garante alta disponibilidade e tolerância a falhas para o sistema de chat.

Em resumo, essa configuração proporciona uma base sólida para um ambiente de negócios moderno, onde a colaboração, o armazenamento seguro e a comunicação eficaz são essenciais.

IV. Conclusão

Este relatório apresentou a criação e configuração de um arquivo `docker-compose.yml` que define a interconexão de quatro serviços distintos, simulando uma aplicação comercial completa. Cada serviço foi escolhido com base em sua relevância para um ambiente de negócios moderno, e a configuração foi elaborada para garantir uma integração eficiente entre eles.

A escolha dos serviços, como Apache para hospedagem web, Nextcloud para colaboração e armazenamento em nuvem, MySQL para o banco de dados, e Rocket.Chat para comunicação em equipe, reflete uma abordagem abrangente para atender às diversas necessidades de uma empresa.

A utilização de contêineres proporciona benefícios significativos em termos de portabilidade, escalabilidade e eficiência de recursos. Cada serviço foi cuidadosamente configurado para se integrar harmoniosamente, proporcionando uma base sólida para um ambiente de negócios moderno e dinâmico.

Além disso, a explicação detalhada do arquivo d`docker-compose.yml` e das escolhas feitas fornece uma compreensão clara da configuração e das razões por trás de cada decisão.

Em um cenário de negócios real, essa configuração oferece uma solução completa, desde a hospedagem web até a comunicação interna, simplificando a administração do banco de dados e garantindo a segurança e a eficácia das operações.

REFERENCES

[1] Docker. What is a container? https://www.docker.com/resources/what-container, 2021. Acessado em 13 de novembro de 2023.

[2] Red Hat. Containers. https://www.redhat.com/en/topics/containers/whats-a-linux-container, 2021. Acessado em 13 de novembro de 2023.

[3] Kubernetes. What is kubernetes? https://kubernetes.io/pt/docs/concepts/overview/what-is-kubernetes/, 2021. Acessado em 13 de novembro de 2023.