Aplicação com Docker + Kubernetes

Alunos:

- Felipe Drumm
- Fernando Cipriano
- Luís Felipe Borsoi

A implementação da aplicação pode ser encontrada no repositório:
https://github.com/luisfelipe998/docker-kubernetes-arq-com-2

Implementação

Neste arquivo será apresentado um resumo da pesquisa e implementação (parcial) de uma aplicação rodando em um ambiente kubernetes de maneira containerizada. O sistema foi desenvolvido utilizando uma máquina rodando MacOS.

Docker e Kubernetes

Foi realizada uma pesquisa sobre o docker e o kubernetes a fim de entender melhor o que eles são, como funcionam e como utilizá-los. Para a apresentação, pretende-se comentar:

- O que é um container;
- Diferenças entre container e máquina virtual
- o que são container images;
- o que são container engines;
- o que são container hosts;
- o que é um registry;
- o que são as camadas de uma imagem;
- Detalhes de implementação do docker:
  - Docker desktop (interface de usuário);
  - Docker CLI;
  - Docker daemon;
  - Dockerfile;
  - Docker compose;
  - Rede e Volumes;
- O que são orquestradores de containers;
- Principais características deles.

Aplicação prática

Após comentar-se sobre a teoria, realizou-se a implementação prática de um projeto rodando o projeto de maneira containerizada. Foi desenvolvida uma aplicação em Node.JS que provê 2 endpoints, um `/register` (para cadastrar um usuário (para simular um login na aplicação). Ambos os endpoints são do tipo POST que recebem o seguinte payload em json:

```
{
    "username": "foo",
    "password": "bar"
}
```

A aplicação é integrada a um banco de dados não relacional: `MongoDB`. A senha é armazenada com encoding, simulando um armazenamento seguro. Em aplicações reais, utilizaria-se outro algoritmo de encoding, mas para fins de simplificação, neste projeto foi utilizado um shift left com uma adição: `hash = ((hash << 5) - hash) + char`.

Containerização

Para rodar a aplicação de maneira containerizada, é necessário instalar o docker:

```
brew install docker
```

Com o docker rodando, pode-se criar um arquivo `Dockerfile` para definir como a imagem deve ser gerada:

```
FROM mhart/alpine-node:16.4 # imagem base
WORKDIR / # define diretório de trabalho onde os arquivos serão copiados no
container
COPY ./package.json ./ # copia o package.json que contém a definição do projeto
e dependências
RUN npm install # instala as dependências em Node.JS
COPY ./src ./src # copia o código fonte do projeto
EXPOSE 8080 # expõe a porta HTTP onde a aplicação estará rodando
CMD ["npm", "start"] # define o comando que o container executará quando
inicializar
```

Como será executado um container com a aplicação e um segundo container com o banco de dados (mongoDB), criou-se um segundo arquivo descritor `docker-compose.yml`, que permite

realizar o gerenciamento de multiplos com maior facilidade. A image para rodar o mongodb foi pega do registry oficial disponível no dockerhub.

```
version: "3.8"

services: # define os serviços que serão criados (mongo + aplicação)
  mongodb:
    image: mongo:5.0.8 # imagem base do dockerhub
    restart: unless-stopped
    env_file: ./.env
    environment:
      - MONGO_INITDB_ROOT_USERNAME=$MONGODB_USER
      - MONGO_INITDB_ROOT_PASSWORD=$MONGODB_PASSWORD
    ports:
      - $MONGODB_LOCAL_PORT:$MONGODB_CONTAINER_PORT # declaração das portas a
serem utilizas e expostas pelo container
    volumes:
      - db:/data/db # ponto de montagem do volume para armazenar dados de forma
persistente
    networks:
      - network # rede de comunicação interna do docker para que os 2 containers
tenham acesso um ao outro
  app:
    image: luisfelipe998/docker-k8s:0.0.1 # localização no registry com a imagem
da aplicação
    # build: . # alternativamente pode-se realizar build da imagem sempre que
subir o container
    restart: unless-stopped
    env_file: ./.env
    ports:
      - $APP_LOCAL_PORT:$APP_CONTAINER_PORT # declaração das portas a serem
utilizas e expostas pelo container
    environment:
      - DB_HOST=mongodb
      - DB_USER=$MONGODB_USER
      - DB_PASSWORD=$MONGODB_PASSWORD
      - DB_NAME=$MONGODB_DATABASE
      - DB_PORT=$MONGODB_CONTAINER_PORT
      - PORT=$APP_CONTAINER_PORT
    networks:
      - network
    labels:
      kompose.service.type: LoadBalancer

volumes: # criação do volume utilizado pelo mongodb
  db:

networks: # criação da rede utilizada entre os containers
```

```
  network:
    driver: bridge
```

Fazendo upload para um registry

Para realizar o upload da imagem para um registry, pode-se realizar o build e o push da imagem como mostrado a seguir. Dessa forma, a imagem fica disponível para download e utilização por outras pessoas. Foi criado um script `push_to_registry.sh` para facilitar o processo:

```
APP_NAME="docker-k8s" # nome da imagem
APP_VERSION="0.0.1" # versão
APP_TAG=${APP_NAME}:${APP_VERSION}
NAMESPACE="luisfelipe998" # namespace (usuário no registry)

docker build -t ${NAMESPACE}/${APP_TAG} . # build da imagem
docker push ${NAMESPACE}/${APP_TAG} # push para registry dockerhub

if [ $? -eq 0 ]; then
    echo "Application image uploaded on:
https://hub.docker.com/r/luisfelipe998/docker-k8s"
else
    echo "Failed to upload application image. Check logs above."
fi
```

Rodando localmente

Para rodar a aplicação localmente, basta executar o comando do docker compose apontando para o `docker-compose.yml` comentado anteriormente. Para facilitar, criou-se um script `run.sh`

```
CONTAINER_NAME=docker-k8s # prefixo para os containers
MONGODB_VOLUME_NAME=${CONTAINER_NAME}_db # prefixo para o volume do banco de
dados

docker compose -p ${CONTAINER_NAME} down # se houver containers rodando, exclui
eles
docker volume rm ${MONGODB_VOLUME_NAME} # se tiver um volume criado, exclui,
para uma instalação "limpa"
docker compose -p ${CONTAINER_NAME} up --build -d # sobe todos os containers
declarados como serviços no docker compose, bem como cria o volume e a rede
```

Para testar a aplicação, pode-se chamar o endpoint `http://localhost:8080/register`, da seguinte forma:

```
brew install curl
```

E após:

```
curl --request POST \
  --url http://localhost:8080/register \
  --header 'Content-Type: application/json' \
  --header 'User-Agent: insomnia/2023.5.8' \
  --data '{
    "username": "foo",
    "password": "bar"
}'
```

Com a seguinte resposta:

```
{
    "hash": "17c13"
}
```

Após isso, pode-se chamar o endpoint `http://localhost:8080/login`, da seguinte forma:

```
curl --request POST \
  --url http://localhost:8080/login \
  --header 'Content-Type: application/json' \
  --header 'User-Agent: insomnia/2023.5.8' \
  --data '{
    "username": "foo",
    "password": "bar"
}'
```

Com a seguinte resposta:

```
{
    "ok": true,
    "message": "user 'foo' logged in."
}
```

Se for informada uma senha incorreta no payload, a resposta do endpoint será:

```
{
    "error": "password doesn't match"
}
```

Instalando um ambiente kubernetes local (minikube)

Agora que se tem a aplicação rodando diretamente no docker, o próximo passo é realizar a instalação de um runtime kubernetes para criamos nosso cluster. Para este projeto, foi escolhido o `minikube`. Para instalar (é imporante ter o docker rodando pois o minikube é instalado como um container):

```
brew install minikube
```

Com ele instalado, rodar `minikube start`. Um cluster kubernetes será instalado localmente. Output:

```
😄  minikube v1.32.0 on Darwin 14.1
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🤹  Updating the running docker "minikube" container ...
🐳  Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🔎  Verifying Kubernetes components...
🌟  Enabled addons: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```

Para verificar o status da instalação:

```
minikube status
```

Output:

```
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

Convertendo os arquivos docker em descritores para kubernetes

Para realizar a instalação dos containers e recursos no cluster kubernetes recém instalado, serão utilizados arquivos descritores. Estes podem ser gerados automaticamente a partir do `docker-compose.yml` utilizando a ferramenta `kompose`. Para instalar:

```
brew install kompose
```

Após a instalação, foi criado o script `k8s_convert.sh` com código a seguir para realizar a conversão:

```
docker-compose config > docker-compose-resolved.yml # gera um arquivo temporário "resolvido" com os valores das variáveis de ambientes definidos
rm -r k8s # remove a pasta k8s onde os arquivos serão gerados
mkdir k8s # cria a pasta k8s onde os arquivos serão gerados
kompose convert -f docker-compose-resolved.yml -o ./k8s # realiza a conversão com o compose
rm docker-compose-resolved.yml # remove o arquivo temporário criado
```

São criados 5 arquivos `yaml` de descrição de recursos necessários para instalação no kubernetes. São eles: `app-deployment.yaml`, `app-tcp-service.yaml`, `mongodb-deployment.yaml`, `mongodb-service.yaml`, `db-persistentvolumeclaim.yaml`.

- Os arquivos de `deployment` são utilizados para gerenciar todos os recursos necessários para o lifecycle das aplicações (aplicação e banco de dados).

- Os arquivos `service` é criado para permitir a exposição das portas de rede dentro do cluster. Por padrão, os containers são totalmente isolados e esse recurso permite a comunicação entre eles.
- O arquivo `persistentvolumeclaim` é utilizado para criar um `persistent volume`, que será utilizado para armazenar os dados do mongodb de maneira não transiente, ou seja, se a aplicação for derrubada, ou se o deployment for deletado, os dados permanecem persistidos.

Rodando no kubernetes

Com os arquivos descritores gerados pelo `kompose`, pode-se realizar o `apply` deles no cluster. Para realizar essa tarefa, foi criado um script `k8s_install.sh`, com o seguinte conteúdo:

```
kubectl apply -f ./k8s/mongodb-deployment.yaml
kubectl apply -f ./k8s/mongodb-service.yaml
kubectl apply -f ./k8s/db-persistentvolumeclaim.yaml
kubectl apply -f ./k8s/app-deployment.yaml
kubectl apply -f ./k8s/app-tcp-service.yaml
```

Basicamente o script realiza o `apply` dos 5 arquivos descritores gerados anteriormente. Pela natureza descritiva (ao invés de imperativa), basta indicar quais recursos devem ser criados pelo kubernetes, e ele se encarrega de criar todo o necessário automaticamente por debaixo dos panos. Após rodar o script, tem-se a seguinte saída:

```
deployment.apps/mongodb created
service/mongodb created
persistentvolumeclaim/db created
deployment.apps/app created
service/app-tcp created
```

Os recursos por padrão são instalados no namespace `default`. Para verificar a criação correta dos recursos pode-se utilizar o CLI do kubernetes:

```
brew install kubectl
```

Para os deployments `kubectl get deployments`:

```
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
app       2/2     2            2           5m10s
```

```
mongodb   1/1     1            1           5m12s
```

Para os services `kubectl get services`:

```
app-tcp      LoadBalancer   10.110.55.126   <pending>     8080:30991/TCP   6m33s
kubernetes   ClusterIP      10.96.0.1       <none>        443/TCP          178m
mongodb      ClusterIP      10.103.50.239   <none>        27017/TCP        6m35s
```

Para os pods `kubectl get pods` (o pod é a recurso do kubernetes onde o container é efetivamente executado):

```
app-5b4f6c46fb-jwlml       1/1     Running   0          5m38s
app-5b4f6c46fb-pbpn9       1/1     Running   0          5m38s
mongodb-7fbc5ccdb7-vddgp   1/1     Running   0          5m40s
```

Para o persistent volume claim do mongo db `kubectl get pvc`:

```
NAME   STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
db     Bound    pvc-b9eb3846-88b2-4435-95ae-7e0521fa62fd   100Mi      RWO
standard       6m5s
```

Expondo a porta da aplicação e testando

Por padrão, os services expõem a porta das aplicações apenas internamente no cluster, ou seja, pela máquina local não é possível acessar a aplicação. Isso pode ser verificado através do `EXTERNAL_IP` dos serviços criados anteriormente, com status `<pendind>`, ou `<none>`. Para solucionar isso, podemos utilizar o tunelamento provido pelo minikube para anexar a porta do serviço dentro do kubernetes à uma porta da máquina local (hosteando o kubernetes). Para isso:

```
  minikube tunnel
```

Rodando novamente o comando `kubectl get services`, temos:

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
app-tcp      LoadBalancer   10.110.55.126   127.0.0.1     8080:30991/TCP   9m51s
kubernetes   ClusterIP      10.96.0.1       <none>        443/TCP          3h1m
mongodb      ClusterIP      10.103.50.239   <none>        27017/TCP        9m53s
```

Nota-se o que o IP do serviço da aplicação foi anexada a interface de rede loopback da máquina, ou seja, ela deve estar disponível localmente agora. Caso a máquina estivesse exposta em um IP público, a aplicação se tornaria disponível em toda a internet. Para testar, pode-se rerodar os `curl` chamando os endpoints HTTP da aplicação:

```
curl --request POST \
  --url http://127.0.0.1:8080/register \
  --header 'Content-Type: application/json' \
  --header 'User-Agent: insomnia/2023.5.8' \
  --data '{
    "username": "foo",
    "password": "bar"
}'
```

```
curl --request POST \
  --url http://127.0.0.1:8080/login \
  --header 'Content-Type: application/json' \
  --header 'User-Agent: insomnia/2023.5.8' \
  --data '{
    "username": "foo",
    "password": "bar"
}'
```

Ambos os comandos retornam a mesma reposta que foi retornada quando se chamou rodando os containers localmente no docker.

Próximos passos

Com a aplicação e o banco de dados rodando no cluster local (minikube), pretende-se modificar o descritor de deployment para adicionar mais replicas (pods) da aplicação e realizar o load balancing entre os diversos pods criados. Ainda se está pesquisando a melhor forma de realizar isso. Estamos procurando se há como fazer isso diretamente pelo kubernetes ou se será necessário algum recurso/ferramenta extra no cluster.

Referências

ADEKANYE, F. Understanding Docker concepts. Disponível em https://www.section.io/engineering-education/docker-concepts/ Acesso em 21 Nov 20222

AGARWAL, N. Docker Container's Filesystem Desmystified. Disponível em https://medium.com/@BeNitinAgarwal/docker-containers-filesystem-demystified-b6ed8112a04a

Acesso em 21 Nov 2023 BEZKODER, Docker Compose Nodejs and MongoDB example Disponível em: https://github.com/bezkoder/docker-compose-nodejs-mongodb Acesso em 21 Nov 2023 CALIZO, M. 6 container concepts you need to understand. Disponível em https://opensource.com/article/20/12/containers-101. Acesso em 21 Nov 2023 CONTAINERD. Containerd. Disponível em containerd.io. Acesso em 21 Nov 2023 CROSBY, M. What is containerd? Disponível em https://www.docker.com/blog/what-is-containerd-runtime/ Acesso em 21 Nov 2023 DOCKER. Container Network. Disponível em: https://docs.docker.com/config/containers/container-networking/ Acesso em 21 Nov 2023 DOCKER. Docker Desktop. Disponível em: https://www.docker.com/products/docker-desktop/ Acesso em 21 Nov 2023 DOCKER. Docker Overview. Disponível em: https://docs.docker.com/get-started/overview/ Acesso em 21 Nov 2023 DOCKER. Manage Data in Docker. Disponível em: https://docs.docker.com/storage/ Acesso em 21 Nov 2023 DOCKER. Persist the DB. Disponível em: https://docs.docker.com/get-started/05_persisting_data/ Acesso em 21 Nov 2023 DOCKER. Use Bind Mounts. Disponível em: https://docs.docker.com/storage/bind-mounts/ Acesso em 21 Nov 2023 DOCKER. Use Bridges. Disponível em: https://docs.docker.com/network/bridge/ Acesso em 21 Nov 2023 DOCKER. Use Volumes. Disponível em: https://docs.docker.com/storage/volumes/ Acesso em 21 Nov 2023 DOCKER. Use containers to Build, Share and Run your applications. Disponível em https://www.docker.com/resources/what - container/#::text=A%20Docker%20container%20image%20is,tools%2C%20system%20libraries %20and%20settings. Acesso em 21 Nov 2023 KOMPOSE, Getting Started. Disponível em https://kompose.io/getting-started Acceso em 21 Nov 2023 KOMPOSE, User Guide. Disponível em https://kompose.io/user-guide Acceso em 21 Nov 2023 KUBERNETES. Kubernetes. Disponível em kubernetes.io Acesso em 21 Nov 2023 MCCARTY, S. A Practical Introduction to Container Terminology. Disponível em https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction# Acesso em 21 Nov 2023 NETAPP. What are containers. Disponível em https://www.netapp.com/devops-solutions/what-are-containers/#::text=Benefits%20of%20containers,-Containers%20are%20a&text=Containers%20require%20less%20system%20resources,t%20include%20operating%20system%20images.&text=Applications%20running%20in%20containers%20can,operating%20systems%20and%20hardware%20platforms. Acesso em 21 Nov 2023 MONGODB, MongoDB Node Driver. Disponível em https://www.mongodb.com/docs/drivers/node/current/ Acesso em 21 Nov 2023 SANTOS, L. Entendendo Runtimes de Containers. Disponível em: https://blog.lsantos.dev/entendendo-runtimes-de-containers/ Acesso em 21 Nov 2023 SANTOS, L. Kubernetes sem Docker? – Entendendo OCI, CRI, e o ecossistema de containers. Disponível em https://blog.lsantos.dev/oci-cri-docker-ecossistema-de-containers/ Acesso em 21 Nov 2023 SIMANUPANG, I. Daemonless Container Engine. Disponível em: https://medium.com/easyread/daemonless-container-engine-5364394b80ec. Acesso em 21 Nov 2023 SIMPLILEARN. What is Docker: Advantages and Components Disponível em https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker Acesso em 21 Nov 2023 SITE24X7. How do containers work. Disponível em https://www.site24x7.com/learn/containers/how-containers-work.html#:~:text=will%20utilize%20it.-,Containers%20are%20an%20abstraction%20in%20the

%20application%20layer%2C%20whereby%20code,running%20as%20an%20isolated%20process. Acesso em 21 Nov 2023 SUPALOV, V. What are Docker image layers? Disponível em https://vsupalov.com/docker-image-layers/ Acesso em 21 Nov 2023 VELICHKO, I. Why and How to Use Containerd From Command Line. Disponível em https://iximiuz.com/en/posts/containerd-command-line-clients/ Acesso em 21 Nov 2023 VMWARE. What is container orchestration? Disponível em https://www.vmware.com/topics/glossary/content/container-orchestration.html#:~:text=Container%20orchestration%20is%20the%20automation,networking%2C%20load%20balancing%20and%20more. Acesso em 21 Nov 2023 WALKER, J. What is Podman and How does it differ from Docker? Disponível em https://www.howtogeek.com/devops/what-is-podman-and-how-does-it-differ-from-docker/#:~:text=In%20Podman%2C%20containers%20can%20form,to%20the%20Kubernetes%20Pod%20concept.&text=The%20Pod%20concept%20is%20powerful,and%20manage%20them%20in%20unison. Acesso em 21 Nov 2023 WAVEWORKS. Comparing Container Orchestrators: 6 choices analyzed. Disponível em https://www.weave.works/blog/comparing-container-orchestration/ Acesso em 21 Nov 2023