Faculdade das Ciências Exatas e da Engenharia

Mestrado em Engenharia Informática

Sistemas distribuídos

Projeto Prático - Fluxo de trabalho de CI/CD para aplicações na nuvem com AKS e K3s

Docentes:

Docente: Karolina Baras

Trabalho Realizado por:

Gonçalo de Castro - 2084515

José Sousa - 2027617

Rúben Silva - 2037517

Funchal, 23 de janeiro de 2022

Índice

1 Resumo

No âmbito da unidade curricular de Sistemas distribuídos foi-nos proposto a realização de um projeto usando a tecnologia Docker para publicar uma aplicação previamente criada e construir um fluxo de trabalho de integração e entrega contínua (CI/CD, Continuous Integration/Continuous Delivery). Numa primeira fase criou-se o fluxo de trabalho de integração e entrega contínua, e numa segunda fase criou-se um cluster local utilizando 2 raspberry pi's.

Para a criação deste fluxo de trabalho usou-se varias tecnologias e ferramentas. Para a alteração de código, relativo ao fluxo de trabalho, utilizou-se o Visual Studio Code, para a criação de contentores utilizou-se o Docker, para o repositório de código foi utilizado o Github, tendo sido usado o Github Actions para automatizar a criação da imagem a partir do Dockerfile e posterior publicação no Azure Container Registry, tendo este sido usado em conjunto com o Azure Kubernetes Service para criar os Clusters para correr a aplicação.

Referente à segunda parte do projeto, criou-se um cluster local a partir de Raspberry Pi (RPI) e K3s, sendo que relativamente as aplicações usadas para o fluxo de trabalho e entrega continua utilizou-se o Visual Studio Code, o Github Actions e o DockerHub.

2 Introdução

Atualmente, existem vários modelos e técnicas para desenvolver aplicações, existindo varias dificuldades como por exemplo o controlo das dependências de uma aplicação, ou a estabilidade da aplicação independentemente da maquina em que se encontra a ser executada.

Existe no entanto uma forma que permite atualizações das aplicações sem ser necessário cancelar a sua execução na totalidade (devido à modularidade), e a sua execução sem depender do sistema operativo da maquina em que se encontram a ser executadas, sendo isto possível a partir da conteinerização.

Neste projeto utiliza-se a plataforma Docker, de forma a criar a imagem da aplicação para posteriormente criar os contentores, sendo usado em conjunção com o Kubernetes, permitindo a criação e gestão de clusters.

Na fase 1 o fluxo de trabalho foi criado tendo em conta a utilização do Visual Studio Code, do Github Actions, do Azure Container Registry e do Azure Kubernetes Service.

No entanto, na fase 2 utilizou-se apenas o Visual Studio Code e o Github Actions, tendo sido adicionado o DockerHub, e os Raspberry Pi's que foram usados para a criação do cluster local.

3 Descrição das atividades

3.1 Fase 1

Nesta fase foi realizado todo o processo base do fluxo de trabalho, tendo sido escolhida a aplicação de teste (aplicação Todo App, utilizada durante a aula).

No workflow (ver Source Code 1) definiu-se que quando for realizado um push para o branch *main*, o fluxo de trabalho fica ativo, sendo depois criada a imagem da app e posteriormente é feito o push da imagem para o Azure Container Registry (ACR), para alem disso, as credenciais de acesso ao ACR foram colocadas como segredos do repositório.

```
1  name: Project Workflow Fase 1
2  on:
3    push:
4      branches:
5        - main
6  jobs:
7    build:
8      runs-on: ubuntu-latest
9      steps:
10       - uses: actions/checkout@v2
11       - name: Build the Docker image
12         working-directory: ./todo_app/app
13         run: docker build . --file Dockerfile --tag
           ↪ projetosdgrupo3.azurecr.io/projetogrupo3sd:latest
14       - name: Login ACR
```

```
15          uses: azure/docker-login@v1
16          with:
17            login-server: ${{ secrets.ACR_SERVER }}
18            username: ${{ secrets.ACR_USERNAME }}
19            password: ${{ secrets.ACR_PASSWORD }}
20        - name: Push Image to ACR
21          run: docker push
            ↪ projetosdgrupo3.azurecr.io/projetogrupo3sd:latest
```

Source Code 1: Workflow utilizado na Fase 1

Relativamente ao Dockerfile, o codigo foi utilizado de um dockerfile fornecido numa das aulas da unidade curricular, tendo sido adicionada a linha 5 e tendo sido feito algumas alterações (ver Source Code 2).

```
1 FROM node:12-alpine
2 RUN apk add --no-cache python2 g++ make
3 WORKDIR /app
4 COPY . .
5 RUN yarn config set "strict-ssl" false -g
6 RUN yarn install --production
7 CMD ["node", "src/index.js"]
```

Source Code 2: DockerFile da app

Relativamente ao fluxo de trabalho foram realizados os seguintes passos:

1. Criou-se um grupo de recursos:

   ```
   az group create --name ProjetoSDGrupo3 --location eastus
   ```

2. Criou-se um registo de contentores (Azure Container Registry) no Azure:

```
az acr create --resource-group ProjetoSDGrupo3 --name
↪ projetosdgrupo3 --sku Standard
```

3. Ativou-se *admin* no ACR:

```
az acr update -n projetosdgrupo3 --admin-enabled true
```

4. Criou-se um Cluster do AKS:

```
az aks create --resource-group ProjetoSDGrupo3 --name
↪ projetosdgrupo3AKSCluster --node-count 2
↪ --generate-ssh-keys
```

5. Obteve-se as credenciais do Cluster AKS:

```
az aks get-credentials --resource-group ProjetoSDGrupo3
↪ --name projetosdgrupo3AKSCluster
```

6. Conectou-se o Cluster AKS ao ACR:

```
az aks update --attach-acr projetosdgrupo3 --name
↪ projetosdgrupo3AKSCluster --resource-group
↪ ProjetoSDGrupo3
```

7. Obteve-se as credenciais de acesso ao ACR:

```
az acr show -n projetosdgrupo3 --query loginServer -o tsv
az acr credential show -n projetosdgrupo3 --query username
↪ -o tsv
az acr credential show -n projetosdgrupo3 --query
↪ passwords[0].value -o tsv
```

8. Colocou-se as credencias do ACR como segredos no repositório.

9. Fez-se um push para o *main* de forma a ativar o workflow.

10. No Azure Cli, criamos um ficheiro *app.yaml*, e copiamos o código presente no ficheiro *app.yaml* (ver Source Code 4).

    ```
    nano app.yaml
    ```

11. Criou-se o cluster a partir do ficheiro *app.yaml*.

    ```
    kubectl apply -f app.yaml
    ```

12. Obteve-se o endereço IP externo para abrir a app no browser.

    ```
    kubectl get service todo-app --watch
    ```

3.2 Fase 2

Nesta fase o procedimento realizado foi diferente do realizado na fase 1, não tendo sido usado o Azure Container Registry nem o Azure Kubernetes Service. Nesta fase utilizou-se o Github Actions e o DockerHub para o fluxo de trabalho e de integração continua.

Para a criação do cluster local foram utilizados 2 raspberry pi's, sendo que um é o master e o outro é o worker/slave. Foram atribuidos 2 endereços IP, 1 para cada RPI (ver Tabela 1)

Role	IP Address	Password	Hostname
Master	10.2.15.156	*Node3_2*	node32
Worker/Slave	10.2.15.155	*Node3_1*	node31

Tabela 1: Endereços IP, Password e Hostname dos Raspberry Pi's.

Para a configuração dos RPI, utilizou-se 2 cartões micro sd preparados anteriormente para possibilitar o acesso aos RPI's a partir de ssh, e para

poder iniciar os RPI's a partir de uma pen USB.

Para a configuração das pens USB, a serem usadas para iniciar cada um dos RPI's, foram realizados os seguintes passos:

1. Gravou-se o sistema operativo (Raspbian OS Lite 32-bits).
2. Criou-se um ficheiro ssh no diretório boot do sistema.
3. Editou-se o ficheiro 'cmdline.txt' adicionando o seguinte separado por espaços:
    - No ficheiro do worker/slave:
        - cgroup_enable $=$ cpuset
        - cgroup_memory $=$ 1
        - cgroup_enable $=$ memory
        - ip $=$ 10.2.15.155
    - No ficheiro do master:
        - cgroup_enable $=$ cpuset
        - cgroup_memory $=$ 1
        - cgroup_enable $=$ memory
        - ip $=$ 10.2.15.156
4. Ligou-se cada pen USB ao respetivo RPI em substituição do cartão SD.
5. Alterou-se as configurações do RPI (Ativar SSH; mudar a password, alterar o hostname. Ver Tabela 1).

6. Editou-se os ficheiros '/etc/dhcpcd.conf' e 'cmdline.txt', retirando comentarios e adicionando o IP, e apagando o IP de cada ficheiro respetivamente.
7. Trocou-se a framework **nftables** pela **iptables**. Realizando depois um reboot dos RPI's.
8. De seguida instalou-se e configurou-se o K3s com os seguintes comandos:
   (a) No Master:
      i. `K3S_KUBECONFIG_MODE="644"`
      ii. `curl -sfL https://get.k3s.io | sh -`
      iii. Testou-se se a instalação ocorreu sem erros usando:
      `sudo kubectl get nodes`
      iv. Copiou-se o Token do Master:
      `sudo cat /var/lib/rancher/k3s/server/node-token`
   (b) No Worker/Slave:
      i. Conectou-se o Worker/Slave com o Master, usando o IP do Master e o Token copiado:

```
curl -sfL http://get.k3s.io |
↪ K3S_URL=https://IP_DO_MASTER:6443
↪ K3S_TOKEN=TOKEN_OBTIDO_PREVIAMENTE sh -
```

9. Testou-se no Master se a conexão com o Worker/Slave funciona:
   `sudo kubectl get nodes`

Relativamente à criação da imagem, criou-se um workflow do github

relativo à esta fase, para que quando seja realizado um push para o branch *main*, seja criada a imagem e colocada no DockerHub (ver Source Code 3).

```
19        - name: build the image
20          working-directory: ./video-streaming1.0
21          run: |
22            docker buildx build --push \
23            --tag 2027617/projetogrupo3videostreamsd:latest \
24            --platform linux/amd64,linux/arm/v7,linux/arm64 .
```

Source Code 3: Image build

Após a imagem ter sido colocada no DockerHub, e os RPI's terem a configuração do K3s concluída e a funcionar, utilizou-se o ficheiro *app.yaml* (ver Source Code 4) para criar o cluster.

4 Análise de resultados

Análise dos resultados 2 a 4 Paginas

5 Conclusão e trabalho futuro

Com este trabalho, colocou-se em prática e aprofundou-se os conceitos lecionados na unidade curricular, nomeadamente os conceitos relativos à conteinerização, assim como os relativos à utilização do Docker, do Kubernetes, do Azure DevOps e do K3s e Raspberry Pi.

Obteve-se também o conhecimento básico relativo à utilização do Github Actions que nos permitiu a automatização do fluxo de trabalho.

Relativamente ao trabalho futuro, considera-se importante a automatização total do fluxo de trabalho a partir do momento em que é efetuada uma alteração no repositório (assim que for realizado um push ou um pull request), de forma que o Workflow criado, realize todos os passos necessários tendo em conta se é necessário parar a execução da aplicação ou não para ser realizada a correção ou atualização de um ou mais modulos dessa aplicação.

6 Anexos

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-app-demo
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      todo-app: web
  template:
    metadata:
      labels:
        todo-app: web
    spec:
      containers:
      - name: todo-app-site
        image: projetosdgrupo3.azurecr.io/projetogrupo3sd
        ports:
        - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: todo-app
  namespace: default
```

```
27 spec:
28   type: LoadBalancer
29   selector:
30     todo-app: web
31   ports:
32     - port: 80
33       targetPort: 3000
```

Source Code 4: Ficheiro descrição do cluster.