Tópicos Avançados em Redes e Sistemas Distribuídos

Projeto Final

Prof. Dr. Fábio Verdi

Bernardo Moreira Zabadal - Especial
Giovanni Alvarenga Silva - 726525
Natham Abdala Merlin Coracini - 7423582

Índice

Introdução

O objetivo deste projeto é fornecer duas slices para dois serviços distintos. Uma para um serviço turístico, na qual deverá conter um vídeo e de forma que este vídeo não fique abaixo de 60fps para nenhum dos usuários que o acesse. E outra para um serviço de coleta de dados de plantações, de modo que o sistema tenha alta disponibilidade.

Máquina utilizada

- Processador intel core i5 7600k
- 16GB de memória RAM
- HD de 1TB
- Sistema Operacional Linux Ubuntu 20.04
- Hypervisor Virtualbox 5.2.43
- Vagrant 2.2.14
- Docker 19.03
- Kubernetes 1.20

Conceitos Utilizados

Slice: Conjunto de elementos computacionais (armazenamento e rede) dedicados (agrupado fim a fim) a um cliente que deve ser controlada, gerenciada e orquestrada individualmente.

Box: Pacote com uma imagem de um S.O pré compilado com apenas os recursos necessários para executar no vagrant.

Contêiner: é uma unidade padrão de software que empacota um código e todas as suas dependências para que uma aplicação seja executada de forma rápida e confiável de um ambiente de computação para outro.

Docker: Plataforma aberta para desenvolvimentos, compartilhamento e execução de aplicativos.

Kubernetes: é uma plataforma aberta (open-source) para o gerenciamento de cargas de trabalho e serviços em ambientes de contêiners, facilitando a automação e configuração de uma infraestrutura (orquestração).

Criação e Execução das Slices

Vagrantfile

O arquivo de configuração base do Vagrant. Nele utilizamos uma imagem de Ubuntu 16.04, criamos três máquinas virtuais através do VirtualBox e as provisionamos com os scripts necessários (incluindo os scripts Ansible para a configuração do Kubernetes).

```
IMAGE_NAME = "ubuntu/xenial64"
N = 2


Vagrant.configure("2") do |config|
    config.vm.provider "virtualbox" do |v|
      v.memory = 2048
      v.cpus = 2

    end

    config.vm.define "k8s-master" do |master|
      master.vm.box = IMAGE_NAME
      master.vm.network "private_network", ip: "192.168.50.10"
      master.vm.network "forwarded_port", guest: 80, host: 8080
      master.vm.hostname = "k8s-master"

      master.vm.provision "ansible" do |ansible|
      ansible.playbook = "./master-playbook.yml"
         ansible.extra_vars = {
            node_ip: "192.168.50.10",
         }
      end
      master.vm.provision "shell", path: "nginx-install.sh"
      master.vm.provision "shell", path: "mongo-install.sh"
      master.vm.provision "shell", path: "setup.sh"
    end

(1..N).each do |i|
    config.vm.define "node-#{i}" do |node|
```

```
       node.vm.box = IMAGE_NAME
       node.vm.network "private_network", ip: "192.168.50.#{i + 10}"
       node.vm.hostname = "node-#{i}"

       node.vm.provision "ansible" do |ansible|
      ansible.playbook = "./node-playbook.yml"
          ansible.extra_vars = {
             node_ip: "192.168.50.#{i + 10}",
          }
       end
    end
  end
end
```

mongo-install.sh

Script para instalação do MongoDB.

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key
add -

sudo apt-get install -y gnupg
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key
add -
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/4.4 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-4.4.list
sudo apt-get update
sudo apt-get install -y mongodb-org
```

nginx-install

Script para instalação do Nginx.

```
sudo apt install -y libpcre3-dev libssl-dev zlib1g-dev

mkdir buildnginx
cd buildnginx
git clone https://github.com/arut/nginx-rtmp-module.git
git clone https://github.com/nginx/nginx.git
cd nginx
```

```
./auto/configure --add-module=../nginx-rtmp-module --with-http_slice_module
--with-http_mp4_module
make
sudo make install
```

Master-playbook.yml

Playbook para criação da máquina master do Kubernetes.

```
---
- hosts: all
  become: true
  tasks:

  - name: Update apt-get repo and cache
    apt: update_cache=yes force_apt_get=yes cache_valid_time=3600

  - name: Install packages that allow apt to be used over HTTPS
    apt:
      name: "{{ packages }}"
      state: present
      update_cache: yes
    vars:
      packages:
      - apt-transport-https
      - ca-certificates
      - curl
      - gnupg-agent
      - software-properties-common

  - name: Add an apt signing key for Docker
    apt_key:
      url: https://download.docker.com/linux/ubuntu/gpg
      state: present

  - name: Add apt repository for stable version
    apt_repository:
      repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu
xenial stable
      state: present
```

```yaml
- name: Install docker and its dependecies
  apt:
    name: "{{ packages }}"
    state: present
    update_cache: yes
  vars:
    packages:
    - docker-ce
    - docker-ce-cli
    - containerd.io
  notify:
    - docker status

- name: Reinicia ssh para funcionamento do docker
  service:
    name: ssh
    state: restarted

- name: Add vagrant user to docker group
  user:
    name: vagrant
    group: docker

- name: Remove swapfile from /etc/fstab
  mount:
    name: "{{ item }}"
    fstype: swap
    state: absent
  with_items:
    - swap
    - none
  become: yes

- name: Disable swap
  command: swapoff -a
  when: ansible_swaptotal_mb > 0
  become: yes

- name: Add an apt signing key for Kubernetes
  apt_key:
    url: https://packages.cloud.google.com/apt/doc/apt-key.gpg
    state: present
  become: yes
```

```
  - name: Adding apt repository for Kubernetes
    apt_repository:
      repo: deb https://apt.kubernetes.io/ kubernetes-xenial main
      state: present
      filename: kubernetes.list
    become: yes

  - name: Atualiza a lista de repositorios e instala os pacotes do K8s
    apt: name={{ item }} update_cache=yes
    loop: ['kubelet', 'kubeadm', 'kubectl']
    become: yes

  # - name: Configure node ip
  #   lineinfile:
  #     path: /etc/default/kubelet
  #     line: KUBELET_EXTRA_ARGS=--node-ip={{ node_ip }}

  - name: Configure node ip
    lineinfile:
      create: yes
      path: /etc/default/kubelet
      line: KUBELET_EXTRA_ARGS=--node-ip={{ node_ip }}

  - name: Restart kubelet
    service:
      name: kubelet
      daemon_reload: yes
      state: restarted

  - name: Inicia o cluster Kubernetes
    command: kubeadm init --apiserver-advertise-address 192.168.50.10
--apiserver-cert-extra-sans="192.168.50.10"  --node-name k8s-master
--pod-network-cidr=192.168.0.0/16
    become: yes

  - name: Cria a estrutura de diret´orios para o funcionamento do
Kubernetes
    file:
      path: /home/vagrant/.kube
      state: directory

  - name: Copia as configura¸c~oes do kubernetes para o diret´orio do
```

```
usu´ario vagrant
    command: cp -i /etc/kubernetes/admin.conf /home/vagrant/.kube/config
    become: yes

  - name: Ajusta as permissoes no arquivo de configuracao do Kubernetes
    file:
      path: /home/vagrant/.kube/config
      owner: vagrant
      group: vagrant
      state: file
    become: yes

  - name: Acertar local da configuracao
    shell: export KUBECONFIG=$HOME/.kube/conf

  - name: Install calico pod network
    become: false
    command: kubectl apply -f
https://docs.projectcalico.org/v3.9/manifests/calico.yaml

  - name: Salva o token para entrada de novos nos no cluster
    command: kubeadm token create --print-join-command
    register: join_command

  - name: Copia o token para entrada de novos nos no cluster em um arquivo
    become: false
    local_action: copy content="{{ join_command.stdout_lines[0] }}"
dest="./join-command.sh"

  handlers:
    - name: docker status
      service: name=docker state=started
```

Nginx.conf

Configuração a ser substituída do Nginx.

```
worker_processes  1;
error_log  logs/error.log;
pid        logs/nginx.pid;
worker_rlimit_nofile 8192;
```

```
events {
  worker_connections  1024;
}

http {
    include       mime.types;
    default_type  application/octet-stream;

    #log_format  main  '$remote_addr - $remote_user [$time_local]
"$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    access_log off;
    sendfile_max_chunk 512k;
    sendfile        on;
    tcp_nopush     on;

    keepalive_timeout  65;

    gzip  on;

    server {
        listen 80;
        location ~ \.mp4$ {
            root /vagrant/videos/;
            mp4;
            mp4_buffer_size       4m;
            mp4_max_buffer_size   10m;
        }
    }
}
```

setup.sh

```
cp -f /vagrant/nginx.conf /usr/local/nginx/conf/nginx.conf
cd /usr/local/nginx/sbin
sudo ./nginx
cp -f /vagrant/mongod.conf /etc/mongod.conf
sudo systemctl unmask mongod
sudo service mongod start
```

Node-playbook.yml

Playbook dos nodes do Kubernetes a serem criados.

```
---
- hosts: all
  become: true
  tasks:

  - name: Update apt-get repo and cache
    apt: update_cache=yes force_apt_get=yes cache_valid_time=3600

  - name: Install packages that allow apt to be used over HTTPS
    apt:
      name: "{{ packages }}"
      state: present
      update_cache: yes
    vars:
      packages:
      - apt-transport-https
      - ca-certificates
      - curl
      - gnupg-agent
      - software-properties-common

  - name: Add an apt signing key for Docker
    apt_key:
      url: https://download.docker.com/linux/ubuntu/gpg
      state: present

  - name: Add apt repository for stable version
    apt_repository:
      repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu
xenial stable
      state: present

  - name: Install docker and its dependecies
    apt:
      name: "{{ packages }}"
      state: present
```

```
    update_cache: yes
  vars:
    packages:
    - docker-ce
    - docker-ce-cli
    - containerd.io
  notify:
    - docker status

- name: Add vagrant user to docker group
  user:
    name: vagrant
    group: docker

- name: Remove swapfile from /etc/fstab
  mount:
    name: "{{ item }}"
    fstype: swap
    state: absent
  with_items:
    - swap
    - none
  become: yes

- name: Disable swap
  command: swapoff -a
  when: ansible_swaptotal_mb > 0
  become: yes

- name: Add an apt signing key for Kubernetes
  apt_key:
    url: https://packages.cloud.google.com/apt/doc/apt-key.gpg
    state: present
  become: yes

- name: Adding apt repository for Kubernetes
  apt_repository:
    repo: deb https://apt.kubernetes.io/ kubernetes-xenial main
    state: present
    filename: kubernetes.list
  become: yes

- name: Atualiza a lista de repositorios e instala os pacotes do K8s
```

```
    apt: name={{ item }} update_cache=yes
    loop: ['kubelet', 'kubeadm', 'kubectl']
    become: yes

  - name: Configure node ip
    lineinfile:
      create: yes
      path: /etc/default/kubelet
      line: KUBELET_EXTRA_ARGS=--node-ip={{ node_ip }}

  - name: Restart kubelet
    service:
      name: kubelet
      daemon_reload: yes
      state: restarted

  - name: Acessa o cluster utilizando o token gerado pelo master
    command: sh /vagrant/join-command.sh
    become: yes

  handlers:
    - name: docker status
      service: name=docker state=started
```

Slice 1 - Serviço para Turismo

Para criar o serviço de turismo, utilizamos um vídeo em 1080p 60fps armazenado na máquina k8s-master, na qual uma pessoa com acesso à slice pode reproduzi-lo entrando no endereço localhost:8080//60fpstest.mp4. O streaming será feito a partir do protocolo http e a taxa de quadros será garantida através do autoscaling do Kubernetes (explicado posteriormente).

Slice 2 - Armazenamento de dados KVS

Para armazenar os dados dos sensores de plantação de soja, foi utilizado o sistema de gerenciamento de banco de dados ***mongodb***. O banco de dados foi arquitetado de forma simples, com somente uma *collection* “plantação” com os campos: tag, temperatura, umidade, teor de clorofila e reflectância.

Para inserir os dados e checar quais campos estão inseridos criamos um pequeno aplicativo de CRUD em node.js, que através das requisições GET e PUT, nos endereços

localhost:3000/plantacao e localhost:3000/cadastro, mostra quais dados estão na tabela e insere dados, respectivamente.

O endereço localhost:3000/faker popula o banco de dados com 5000 entradas com dados aleatórios, para facilitar o processo de teste e desenvolvimento.

O banco de dados foi armazenado na slice 1 da máquina virtual mestre do Kubernetes. Utilizamos o script *mongod.sh* para instalar o *mongodb* e realizamos alguns comandos para iniciar o processo e deixar ele rodando. Primeiramente, substituímos o arquivo de configuração padrão pelo *mongod.conf* do projeto. Posteriormente rodamos os comandos abaixo para garantir que o processo mongod esteja rodando no plano de fundo:

```
sudo systemctl unmask mongod
sudo service mongod start
```

Depois disso, para testar, rodamos na máquina host o app do node criado através do comando *npm start* start no terminal.

Inserimos alguns dados na rota 192.168.50/cadastro através do client REST Insomnia utilizando o PUT request e testamos com uma requisição GET no endereço 192.168.50/plantacao, na qual o JSON dos dados inseridos foi retornado.

Auto scaling

Para realizar o auto scaling utilizamos alguns comandos do Kubernetes. Primeiramente, criamos nossa deployment a partir da imagem base fornecida nos arquivos da pasta do projeto:

```
kubectl create deployment nginx --image=nginx
```

Posteriormente, escalamos nossa deployment para um tamanho inicial ideal para suprir a nossa demanda de uso nos consumo de vídeo e queries no KVS. Isso foi feito através do comando:

```
kubectl scale --replicas=3 deployment.apps/nginx
```

Com esse comando, pegamos nosso deployment e escalamos ele com 3 réplicas.

Por fim, para automatizar esse processo, utilizamos o autoscale do Kubernetes, através do comando abaixo:

```
kubectl autoscale --min=3 --max=10 --cpu-percent=80 deployment.apps/nginx
```

A máquina será escalada automaticamente quando o uso de CPU passar de 80%, de no mínimo 3 pods para no máximo 10. Isso irá garantir que ambas as queries do KVS e a taxa de quadros do vídeo se mantenham estáveis.

Dificuldades

Tivemos dificuldades em algumas etapas do trabalho. Inicialmente, era muito complicado diagnosticar erros na criação das máquinas virtuais através do Vagrant pois a depuração de erros era extremamente demorada, visto que toda vez era necessário construir novamente as máquinas virtuais, o que causa uma demora de 15-20 min a cada erro que cometíamos.

Além disso, tivemos dificuldade para entender como implementar a parte de slicing, visto que com base na documentação do Kubernetes, isso era feito automaticamente através do End-point Slicing. Tivemos também dificuldades nas configurações do Nginx, durante os trabalhos intermediários e o final.

Por fim, tivemos diversos bugs com o Vagrant que precisavam de um tempo grande para serem corrigidos. Talvez se utilizássemos outra versão, a quantidade de bugs seria menor.