Universidade Federal de São Carlos
Programa de Pós-graduação em Ciência da Computação - Sorocaba
Disciplina de Tópicos Avançados em Arquiteturas Distribuídas de Software

Jaime Freire de Souza
Marcelo Rodrigo de Castro
Marco Antonio da Silva Santos

RELATÓRIO DO PROJETO FINAL

Sorocaba - SP
Dezembro/2020

1. Introdução

O seguinte projeto se aplica no contexto de um provedor de *Cloud Network Slicing* (CNS). O objetivo é prover e gerenciar uma infraestrutura escalável para a execução de duas *slices* distintas. Na execução do projeto, foram utilizadas duas tecnologias habilitadoras: docker (para a criação de contêineres) e kubernetes (para a orquestração dos contêineres criados). Os códigos utilizados estão disponíveis no github[1].

2. Configuração do ambiente

O ambiente é composto por nós virtualizados. Essas máquinas foram configuradas para compor um *cluster* kubernetes. Cada *slice* é definida como um *pod* em execução no *cluster*. Para fins de aprendizado e experimentação, foram utilizadas três alternativas para a criação da infraestrutura virtualizada do *cluster:* ambiente local com virtualbox e vagrant, nuvem da Oracle e nuvem da Google.

A seguir serão descritos os procedimentos realizados para a criação e configuração do ambiente em cada plataforma.

2.1 Local (VirtualBox)

Um ambiente virtualizado localmente (no notebook) com o auxílio do virtualbox e vagrant, foi criado como parâmetro para ambientes que não estejam em nuvem, bem como para demonstrar o uso e adequação conforme a infraestrutura existente. Para o ambiente foi configurado um servidor *master* e dois *workers*, com 2 VCPUs, memória de 2 GB, interface de rede padrão e com interface virtual para acesso aos *pods*. Os *scripts* utilizados, foram adaptados conforme o ambiente em nuvem da Oracle (a ser descrito a seguir).

[1] https://github.com/jaimesouza/k8s-cluster

2.2 Nuvem da Oracle

No serviço de computação em nuvem oferecido pela Oracle[2], foram instanciadas três máquinas virtuais (um *master* e dois *workers*) homogêneas, do tipo VM.Standard.E2.2, com 2 núcleos (4 *threads*), 16 GB de memória RAM, 1.4 Gbps de largura de banda da rede, e sistema operacional Ubuntu 20.04. As três VMs estão em um mesmo domínio de disponibilidade e mesma rede virtual interna. Além disso, cada máquina possui um IP público associado:

- **master:** 150.136.80.103
- **worker 1:** 129.213.59.35
- **worker 2:** 150.136.164.119

A instalação e configuração do kubernetes foi feita com o auxílio dos seguintes *scripts* em *shell*: *config-master.sh*[3] para a configuração do *master* e *config-worker.sh*[4] para a configuração dos *workers*. Ambos os *scripts* executam o arquivo *install-docker.sh*[5] para a instalação e configuração do docker. O comando para adição de cada *worker* ao *cluster* (*kubeadm join*), foi executado manualmente. A figura abaixo, apresenta a lista de nós do *cluster*, após a inserção dos *workers*.

```
ubuntu@master:~$ kubectl get nodes
NAME      STATUS   ROLES                  AGE    VERSION
master    Ready    control-plane,master   3d5h   v1.20.0
worker1   Ready    <none>                 3d5h   v1.20.0
worker2   Ready    <none>                 3d5h   v1.20.0
```

2.3 Nuvem da Google

No serviço de computação em nuvem oferecido pela Google[6], foram instanciados quatro nós de forma homogênea, do tipo n1-standard-1, com 1 núcleo, 3.75 GB de memória RAM, 1.4 Gbps de largura de banda da rede, e sistema operacional Container-Optimized OS do Google. As VMs estão em um mesmo domínio de disponibilidade e mesma rede virtual interna, no caso *us-central*.

[2] https://www.oracle.com/cloud/
[3] https://github.com/jaimesouza/k8s-cluster/blob/main/cluster-config/config-master.sh
[4] https://github.com/jaimesouza/k8s-cluster/blob/main/cluster-config/config-worker.sh
[5] https://github.com/jaimesouza/k8s-cluster/blob/main/cluster-config/install-docker.sh
[6] https://cloud.google.com/

Neste tipo de ambiente, o monitoramento é feito pela própria nuvem e não é preciso criar um nó *master*, tendo apenas os quatro nós *workers*. Além disso, cada máquina possui um IP público associado. No entanto, como o recurso é gratuito, a cada três horas o serviço é deletado. Assim, a cada execução dos testes tivemos um IP associado diferente.

A instalação e configuração do kubernetes foi feita com o auxílio do seguinte *script* em *shell*: *ScriptToCreateK8sClusterInGCP.sh*[7], onde todo o processo de criação do *cluster*, disponibilização do contêiner, exposição da porta e balanceamento de carga, foram feitos de forma automatizada. A figura abaixo, apresenta a lista de nós do *cluster*. Lembrando, que neste caso, o papel de monitoramento é feito pelo próprio serviço da nuvem.

```
student_03_363a0330e2e2@cloudshell:~ (qwiklabs-gcp-03-3ef297a232ee)$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
gke-seminario-default-pool-d6686697-0v4r   Ready    <none>   3m21s   v1.16.15-gke.4300
gke-seminario-default-pool-d6686697-432v   Ready    <none>   3m13s   v1.16.15-gke.4300
gke-seminario-default-pool-d6686697-8bgb   Ready    <none>   3m11s   v1.16.15-gke.4300
gke-seminario-default-pool-d6686697-wvql   Ready    <none>   3m11s   v1.16.15-gke.4300
student_03_363a0330e2e2@cloudshell:~ (qwiklabs-gcp-03-3ef297a232ee)$
```

Na GCP o provisionamento, criação do *cluster* e disponibilização das slices se deu conforme os scripts disponibilizados no github.

3. Slices

O ambiente executa duas *slices* distintas. A primeira *slice* consiste em um serviço de transmissão de vídeo, que apresenta variação da demanda de acordo com o período. Desta forma a nuvem deve se adaptar (elasticidade) para atender os picos de acesso ao serviço, de modo a manter a taxa de FPS (quadros por segundo) igual ou superior a 60. A segunda *slice* possui um serviço de armazenamento de dados do tipo *Key Value Store* (KVS). Ambas as slices estão em execução, de forma isolada, como *pods* diferentes do *cluster*, como mostrado nas figuras a seguir.

---

7 https://github.com/jaimesouza/k8s-cluster/blob/main/nginx-seminario/ScriptToCreateK8sClusterInGCP.sh

```
ubuntu@master:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
kvs-5799cf557-7zdft                 1/1     Running   0          138m
nginx-seminario-7c9bd8db98-cl858    1/1     Running   0          42h
```

Pods na Oracle

```
student_03_363a0330e2e2@cloudshell:~ (qwiklabs-gcp-03-3ef297a232ee)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
kvs-64d476bc64-bt9cm                1/1     Running   0          57s
nginx-seminario-6f5cfbd9bc-s5qnx    1/1     Running   0          84s
student_03_363a0330e2e2@cloudshell:~ (qwiklabs-gcp-03-3ef297a232ee)$
```

Pods na Google

Os *pods* listados nas imagens acima (em cada um dos provedores de nuvem) representam respectivamente as *slices* 2 e 1. A seguir serão detalhados os passos para a criação de cada *slice*.

3.1 Slice 1 - Serviço de vídeo

A primeira slice consiste na execução de um contêiner docker para uma aplicação web que disponibiliza um serviço de vídeo. O serviço pode ser acessado via uma URL em algum navegador. Como é um serviço web simples, foi instalado o servidor nginx e alterado a página inicial para “consumir” o vídeo.

O contêiner foi criado utilizando uma imagem base do nginx (*nginx:latest*), disponível no dockerhub. Como a imagem possui o serviço web pré-configurado, não foi necessário fazer ajustes e nem instalar nenhum pacote adicional. Para criar a imagem docker da aplicação do serviço de vídeo foi utilizado um *dockerfile*. Após a criação da aplicação na forma de uma imagem de contêiner, foi realizada a publicação no dockerhub.

Na sequência foi executado o *deployment* da aplicação no cluster kubernetes, seguindo os comandos do arquivo *ScriptToCreateK8sClusterInGCP.sh*. O serviço, exposto para acesso público via LoadBalance, é executado na porta 80 do contêiner e disponibilizado, também, na porta 80 para acesso externo no IP que é atribuído pela própria Google ou no IP 150.136.80.103 da Oracle com a porta 30084, atribuída pelo kubernetes[8]. A elasticidade do serviço é determinada (e

[8] http://150.136.80.103:30084

disparada) de acordo com o uso de CPU. Quando o uso de CPU chega a 50%, o kubernetes cria mais réplicas (escalabilidade horizontal) para o *pod* da *slice* 1. Da mesma forma, o sistema reduz a quantidade de réplicas quando o uso de CPU chega a um valor inferior ao limite determinado.

3.2. Slice 2 - Key Value Store (KVS)

A segunda *slice* consiste em um *pod* que executa um contêiner docker para uma aplicação de armazenamento de chaves e valores (*key value store* (KVS), em inglês). O serviço foi desenvolvido para ser uma aplicação *web*, podendo ser acessado via uma URL[9] no navegador. Para o desenvolvimento da aplicação foram utilizados o mongoDB[10] (banco de dados) e python (pacote flask[11]). O flask provê um serviço web que acessa e atualiza a base de dados armazenada no mongoDB. A aplicação permite as operações de inserção (*put*), busca (*get*) e remoção de registros. O código fonte (desenvolvido pela equipe deste projeto) está disponível no github[12].

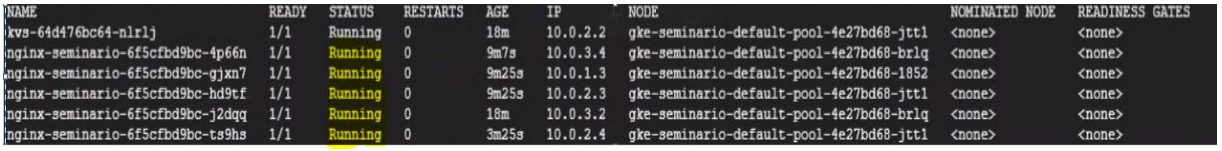
Para a criação do contêiner, foi usada como base a imagem do mongoDB (*mongo:latest*), disponível no dockerhub[13]. No contêiner base instanciado, foi instalado o python3, com os pacotes flask (servidor e aplicação *web*) e pymongo (pacote para acesso e atualização do mongoDB via python). Para criar a imagem docker da aplicação do KVS foi utilizado um dockerfile[14] e executados os comandos do arquivo *create-image.sh*[15]. Uma vez feito o empacotamento da aplicação na forma de uma imagem de contêiner, foi realizada a publicação no dockerhub. E na sequência foi executado o *deployment* da aplicação no *cluster* kubernetes, seguindo os comandos do arquivo *deploy-on-k8s.sh*[16]. O serviço, exposto para acesso público via NodePort, é executado na porta 5000 do contêiner e 31351 (porta atribuída pelo kubernetes) para acesso externo no IP 150.136.80.103.

[9] http://150.136.80.103:31351
[10] https://www.mongodb.com
[11] https://flask.palletsprojects.com
[12] https://github.com/jaimesouza/k8s-cluster/tree/main/kvs-app
[13] https://hub.docker.com/_/mongo
[14] https://github.com/jaimesouza/k8s-cluster/blob/main/kvs-app/Dockerfile
[15] https://github.com/jaimesouza/k8s-cluster/blob/main/kvs-app/create-image.sh
[16] https://github.com/jaimesouza/k8s-cluster/blob/main/kvs-app/deploy-on-k8s.sh

4. Teste de escalabilidade

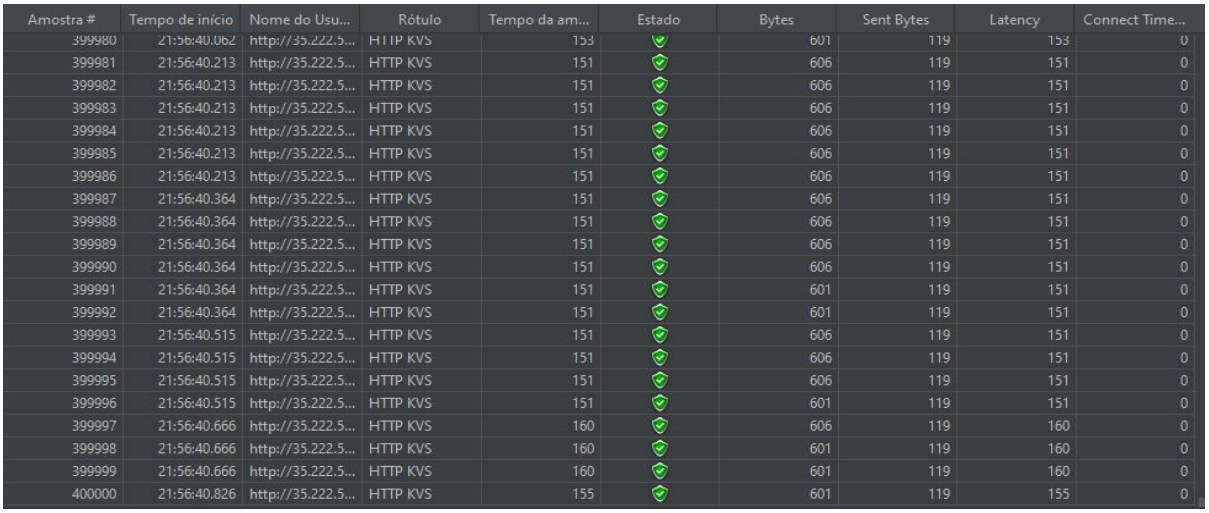
Para simular a carga de demanda e verificar a escalabilidade da *slice* 1, utilizamos a ferramenta Apache JMETER, configurando acessos simultâneos por usuários, tempo de requisição na página e quantidade de repetições. Ao executar o teste de acesso, monitoramos os *pods* em execução na nuvem da Google. Os parâmetros de testes foram os seguintes:

- **Primeiro teste:** 4 usuários simultâneos com inicialização de 20 segundos e 2 repetições. Essa carga de acesso não apresentou impacto na escalabilidade.
- **Segundo teste:** 500 usuários simultâneos com inicialização de 20 segundos e 20 repetições. Neste cenário, o kubernetes escalou a *slice* 1 para 5 réplicas do *pod*, como mostra a figura abaixo.



NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
kvs-64d476bc64-nlrlj	1/1	Running	0	18m	10.0.2.2	gke-seminario-default-pool-4e27bd68-jtt1	<none>	<none>
nginx-seminario-6f5cfbd9bc-4p66n	1/1	Running	0	9m7s	10.0.3.4	gke-seminario-default-pool-4e27bd68-brlq	<none>	<none>
nginx-seminario-6f5cfbd9bc-gjxn7	1/1	Running	0	9m25s	10.0.1.3	gke-seminario-default-pool-4e27bd68-1852	<none>	<none>
nginx-seminario-6f5cfbd9bc-hd9tf	1/1	Running	0	9m25s	10.0.2.3	gke-seminario-default-pool-4e27bd68-jtt1	<none>	<none>
nginx-seminario-6f5cfbd9bc-j2dqq	1/1	Running	0	18m	10.0.3.2	gke-seminario-default-pool-4e27bd68-brlq	<none>	<none>
nginx-seminario-6f5cfbd9bc-ts9hs	1/1	Running	0	3m25s	10.0.2.4	gke-seminario-default-pool-4e27bd68-jtt1	<none>	<none>

Neste experimento, o JMETER não gerou indisponibilidade do serviço, como mostra a figura a seguir.



Amostra #	Tempo de início	Nome do Usu...	Rótulo	Tempo da am...	Estado	Bytes	Sent Bytes	Latency	Connect Time...
399980	21:56:40.062	http://35.222.5...	HTTP KVS	153		601	119	153	0
399981	21:56:40.213	http://35.222.5...	HTTP KVS	151		606	119	151	0
399982	21:56:40.213	http://35.222.5...	HTTP KVS	151		606	119	151	0
399983	21:56:40.213	http://35.222.5...	HTTP KVS	151		606	119	151	0
399984	21:56:40.213	http://35.222.5...	HTTP KVS	151		606	119	151	0
399985	21:56:40.213	http://35.222.5...	HTTP KVS	151		606	119	151	0
399986	21:56:40.213	http://35.222.5...	HTTP KVS	151		606	119	151	0
399987	21:56:40.364	http://35.222.5...	HTTP KVS	151		606	119	151	0
399988	21:56:40.364	http://35.222.5...	HTTP KVS	151		606	119	151	0
399989	21:56:40.364	http://35.222.5...	HTTP KVS	151		606	119	151	0
399990	21:56:40.364	http://35.222.5...	HTTP KVS	151		606	119	151	0
399991	21:56:40.364	http://35.222.5...	HTTP KVS	151		601	119	151	0
399992	21:56:40.364	http://35.222.5...	HTTP KVS	151		601	119	151	0
399993	21:56:40.515	http://35.222.5...	HTTP KVS	151		606	119	151	0
399994	21:56:40.515	http://35.222.5...	HTTP KVS	151		606	119	151	0
399995	21:56:40.515	http://35.222.5...	HTTP KVS	151		606	119	151	0
399996	21:56:40.515	http://35.222.5...	HTTP KVS	151		601	119	151	0
399997	21:56:40.666	http://35.222.5...	HTTP KVS	160		606	119	160	0
399998	21:56:40.666	http://35.222.5...	HTTP KVS	160		601	119	160	0
399999	21:56:40.666	http://35.222.5...	HTTP KVS	160		601	119	160	0
400000	21:56:40.826	http://35.222.5...	HTTP KVS	155		601	119	155	0

Ao final do pico de carga gerado pela ferramenta, e passados 5 minutos (configuração do kubernetes), as réplicas do *pod* começaram a ser encerradas, retornando ao estado original, com apenas 1 réplica.

5. Conclusão e dificuldades encontradas

Para este projeto, a equipe criou e configurou um cluster kubernetes para a execução de duas *slices* diferentes. Além disso, foi feito o desenvolvimento das aplicações a serem executadas em cada *slice*. A primeira dificuldade encontrada foi a obtenção de recursos físicos com os requisitos mínimos necessários para a criação do ambiente virtualizado, pois era necessário um computador (local) que pudesse suportar a instalação de três máquinas virtuais ou o acesso aos recursos virtuais em um ambiente de nuvem. Seguindo as dicas dos colegas em sala de aula, conseguimos o acesso à nuvem da Oracle e da Google.

Uma vez dispondo de recursos computacionais, foi necessário um esforço para entender a plataforma de nuvem (Oracle e Google), instalar e configurar o kubernetes nas máquinas virtuais. Após algumas tentativas, a instalação e configuração do *cluster* kubernetes foi concluída com sucesso. Com o cluster funcionando corretamente, foram necessárias algumas pesquisas para entender a ideia e o funcionamento do kubernetes, bem como os comandos necessários para a implantação das aplicações que foram desenvolvidas. Ao final do processo, os dois serviços (*slices*) foram implantados com sucesso no *cluster*.