

# MENTORED

## Guia para novos usuários do *testbed* MENTORED

Davi Daniel Gemmer, Bruno Henrique Meyer  
MENTORED-WP4

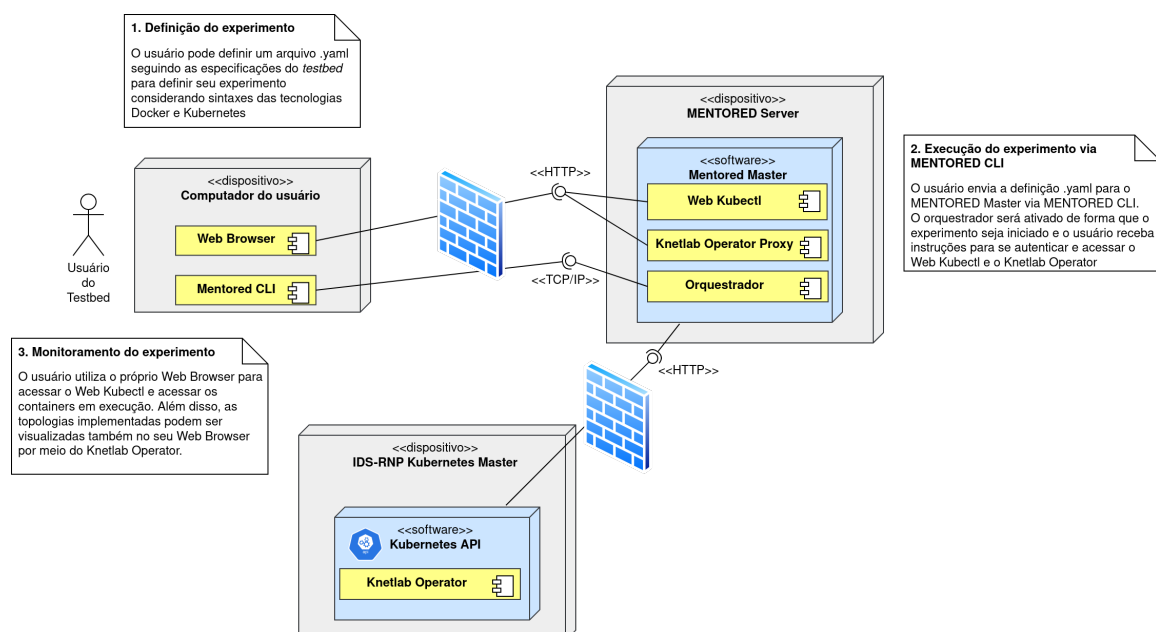
Fevereiro de 2022

Este guia tem como objetivo auxiliar novos usuários do *testbed* MENTORED em seus experimentos. A Sessão 1 apresenta uma contextualização do *testbed* MENTORED. A Sessão 2 é referente a criação de uma imagem Docker a partir de um arquivo Dockerfile. Na Sessão 3 é apresentado a construção e execução da topologia através de um arquivo *YML*. A Sessão 4 mostra a topologia criada através do KNetLab. A Sessão 5 é referente a manipulação dos *pods*. Na Sessão 6 é mostrado um exemplo de comunicação entre os *pods*. Por fim a Sessão 7 apresenta comandos que podem ser uteis durante a execução do experimento.

### 1. Diagrama do processo de utilização do MENTORED

O processo de utilização do ambiente MENTORED é dividido em três principais pontos; 1. Definição do experimento, 2. Execução do experimento via MENTORED CLI e 3. Monitoramento do experimento. A Figura 1 mostra o diagrama com a descrição das etapas para a utilização do *testbed* MENTORED.

Figura 1: Diagrama MENTORED



## 2. Elaboração da imagem de *container*

Cada pesquisador possui objetivos diferentes com a execução dos experimentos, como por exemplo sistemas operacionais e ferramentas específicas. A elaboração de imagens de *container* evita a execução repentina de comandos e acelera o processo de configuração do ambiente. Esse processo é feito através de um documento de texto denominado Dockerfile que cria automaticamente imagens lendo as instruções do arquivo. O código 1 é um exemplo básico para a composição de uma imagem.

Código 1: Exemplo de um arquivo Dockerfile

```
FROM debian

ENV LANG C.UTF-8

RUN apt update --fix-missing && \
    apt -y install \
    python3 python3-pip procps iputils-ping net-tools \
    ↪ wondershaper nano vim curl iproute2

COPY ./entry.sh /
COPY ./create_env_from_mentored_ip_list.py /
```

- **FROM:** A instrução FROM é obrigatória e é utilizada para definir o ponto de partida. Como pode ser observado nesse exemplo foi utilizado a versão do sistema operacional Debian;
- **LANG:** A instrução LANG é utilizada para definir a localidade;
- **RUN:** A instrução RUN é utilizada para definir quais serão os comandos executados na criação da imagem. Essa instrução também pode ser utilizada mais que uma vez. Nesse exemplo foi executado o comando *apt update* responsável pela atualização da imagem e na sequência foi utilizado o comando *apt install* com os *softwares* que serão instalados na imagem;
- **COPY:** A instrução COPY permite a passagem de arquivos ou diretórios. É utilizado para copiar arquivos para dentro da imagem. Nesse exemplo foi copiado o arquivo *entry.sh* e o arquivo *create-env-from-mentored-ip-list.py*. Ambos arquivos são disponibilizados pelo projeto MENTORED e obrigatórios para o funcionamento do experimento.

Após o arquivo do Dockerfile estar configurado é necessária sua execução através do comando *docker build*. O *docker build* é responsável pela construção da imagem a partir do Dockerfile. O *build* não trabalha com o caminho do arquivo, tornando necessário informar o caminho do diretório. Nesse exemplo o *build* será executado dentro do diretório em que foi salvo o arquivo Dockerfile com o objetivo de facilitar esse processo, os arquivos *entry.sh* e *create-env-from-mentored-ip-list.py*

também estão presentes no mesmo diretório do Dockerfile lembrando que devem ser copiados para dentro da imagem. O comando é dividido em quatro partes;

1. **docker build** : Comando responsável pela construção da imagem;
2. **-t** : Parâmetro utilizado para informar que a imagem pertence ao usuário;
3. **user/generic-client:v1** : Nome do usuário seguido pelo nome da imagem e tag com a versão.
4. **.** : Significa que o Dockerfile está no diretório em que o build será executado

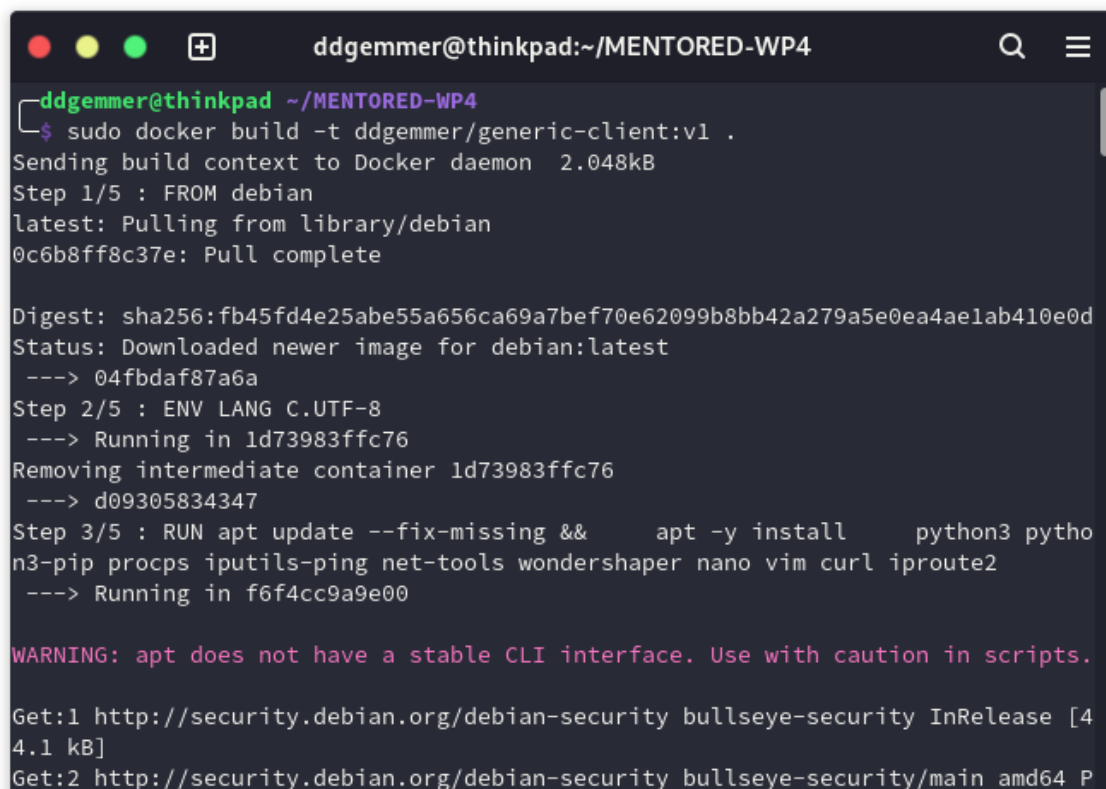
Para esse processo é necessário abrir uma janela de terminal e ter permissão de super usuário para a utilização do Docker. A partir disso é iniciado o processo de criação da imagem através da execução do comandos no terminal:

```
$ sudo docker build -t user/generic-client:v1 .
```

A figura 2 mostra a execução do Dockerfile. Como pode ser observado na Figura 3, após a conclusão é possível observar a imagem através do comando:

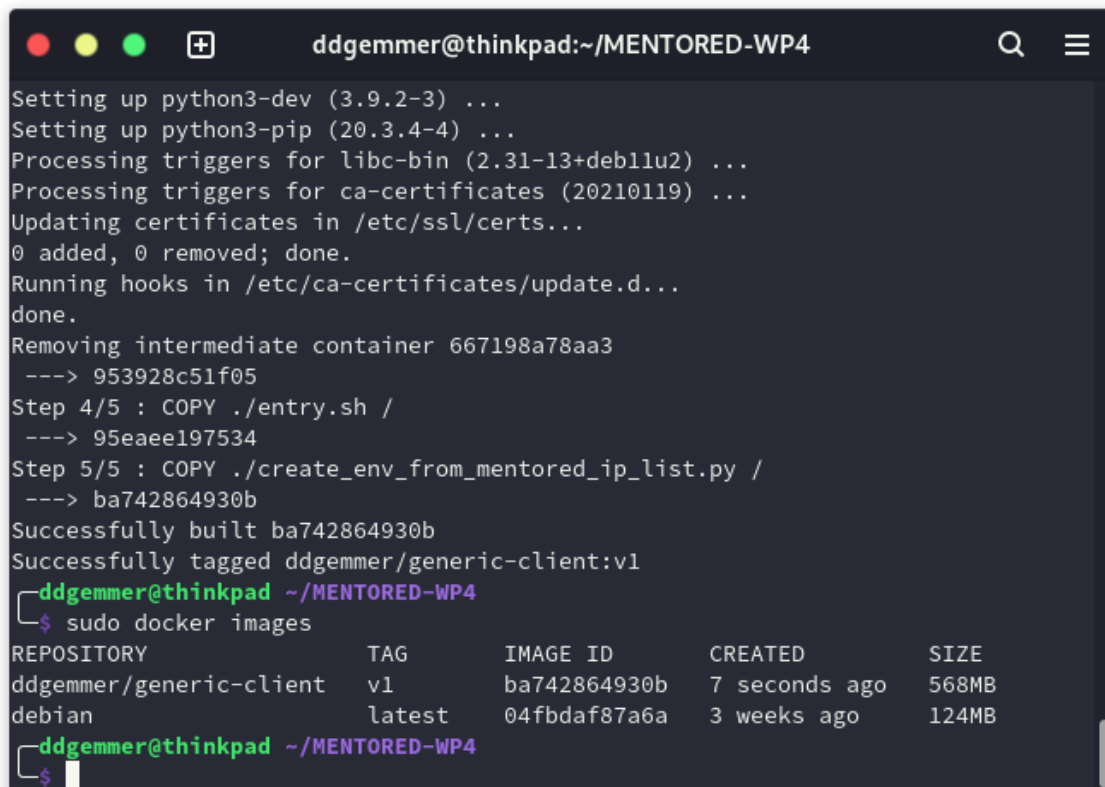
```
$ sudo docker images
```

Figura 2: Execução do arquivo Dockerfile



```
ddgemmer@thinkpad ~/MENTORED-WP4
$ sudo docker build -t ddgemmer/generic-client:v1 .
Sending build context to Docker daemon 2.048kB
Step 1/5 : FROM debian
latest: Pulling from library/debian
0c6b8ff8c37e: Pull complete
Digest: sha256:fb45fd4e25abe55a656ca69a7bef70e62099b8bb42a279a5e0ea4ae1ab410e0d
Status: Downloaded newer image for debian:latest
--> 04fbdaf87a6a
Step 2/5 : ENV LANG C.UTF-8
--> Running in 1d73983ffc76
Removing intermediate container 1d73983ffc76
--> d09305834347
Step 3/5 : RUN apt update --fix-missing && apt -y install python3 python3-pip procps iputils-ping net-tools wondershaper nano vim curl iproute2
--> Running in f6f4cc9a9e00
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
Get:1 http://security.debian.org/debian-security bullseye-security InRelease [44.1 kB]
Get:2 http://security.debian.org/debian-security bullseye-security/main amd64 P
```

Figura 3: Listagem das imagens Docker



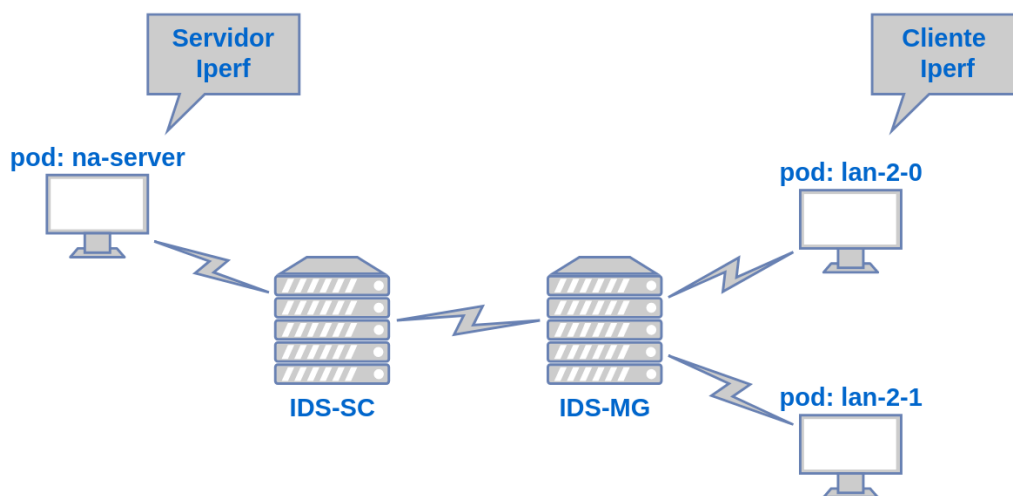
```
ddgemmer@thinkpad:~/MENTORED-WP4
Setting up python3-dev (3.9.2-3) ...
Setting up python3-pip (20.3.4-4) ...
Processing triggers for libc-bin (2.31-13+deb11u2) ...
Processing triggers for ca-certificates (20210119) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Removing intermediate container 667198a78aa3
--> 953928c51f05
Step 4/5 : COPY ./entry.sh /
--> 95eae197534
Step 5/5 : COPY ./create_env_from_mentored_ip_list.py /
--> ba742864930b
Successfully built ba742864930b
Successfully tagged ddgemmer/generic-client:v1
ddgemmer@thinkpad ~/MENTORED-WP4
$ sudo docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
ddgemmer/generic-client v1         ba742864930b  7 seconds ago  568MB
debian               latest      04fbdaf87a6a  3 weeks ago   124MB
ddgemmer@thinkpad ~/MENTORED-WP4
$
```

Como pode ser observado na Figura 3 existem duas imagens, onde a primeira é a do Debian, lembrando que a imagem construída nesse exemplo utilizou o Debian com o comando FROM. E a segunda imagem é a versão que foi construída através do Dockerfile. Isso encerra a etapa da elaboração de uma imagem de *container* a partir de um arquivo Dockerfile. Agora a imagem com os *softwares* do pesquisador está pronta para ser implementada em um arquivo YML.

### 3. Elaboração e execução do experimento

Primeiramente o pesquisador deve estipular qual será a topologia necessária para seu experimento. Nesse exemplo será construída uma topologia simples para fins demonstrativos do processo de construção da mesma. Como pode ser observado na Figura 4 o objetivo será elaborar uma topologia com um *pod* denominado *na-server* conectado no IDS-SC que posteriormente será utilizado como servidor para a comunicação entre *Pods* através do *software iperf*. E um segundo *pod* denominado *lan-2-0* com uma replica denominada *lan-2-1* conectada no IDS-MG que posteriormente será utilizado como cliente do *software iperf*.

Figura 4: Topologia



Código 2: Exemplo do arquivo YML com a topologia

```
Experiment:
  name: exemplo_mentored # @

nodeactors:
  - name: 'na-server'
    replicas: 1
    containers:
      - name: na-server
        image: ddgemmer/generic-client:v1
        imagePullPolicy: "Always"
        command: ["tail", "-f", "/dev/null"]
        securityContext:
          privileged: true
        region: 'ids-sc'
  - name: 'lan-2'
    replicas: 2
    containers:
      - name: lan-2
        image: ddgemmer/generic-client:v1
        imagePullPolicy: "Always"
        command: ["tail", "-f", "/dev/null"]
        securityContext:
          privileged: true
        region: 'ids-mg'
  topology: 'ovs_fully_connected'
```

A partir da Figura 4 é configurado o arquivo *YML* do Código 2. Como pode ser observado as informações contidas no código *YML* são responsáveis pela construção da topologia. O Código 2 composto pelos seguintes itens:

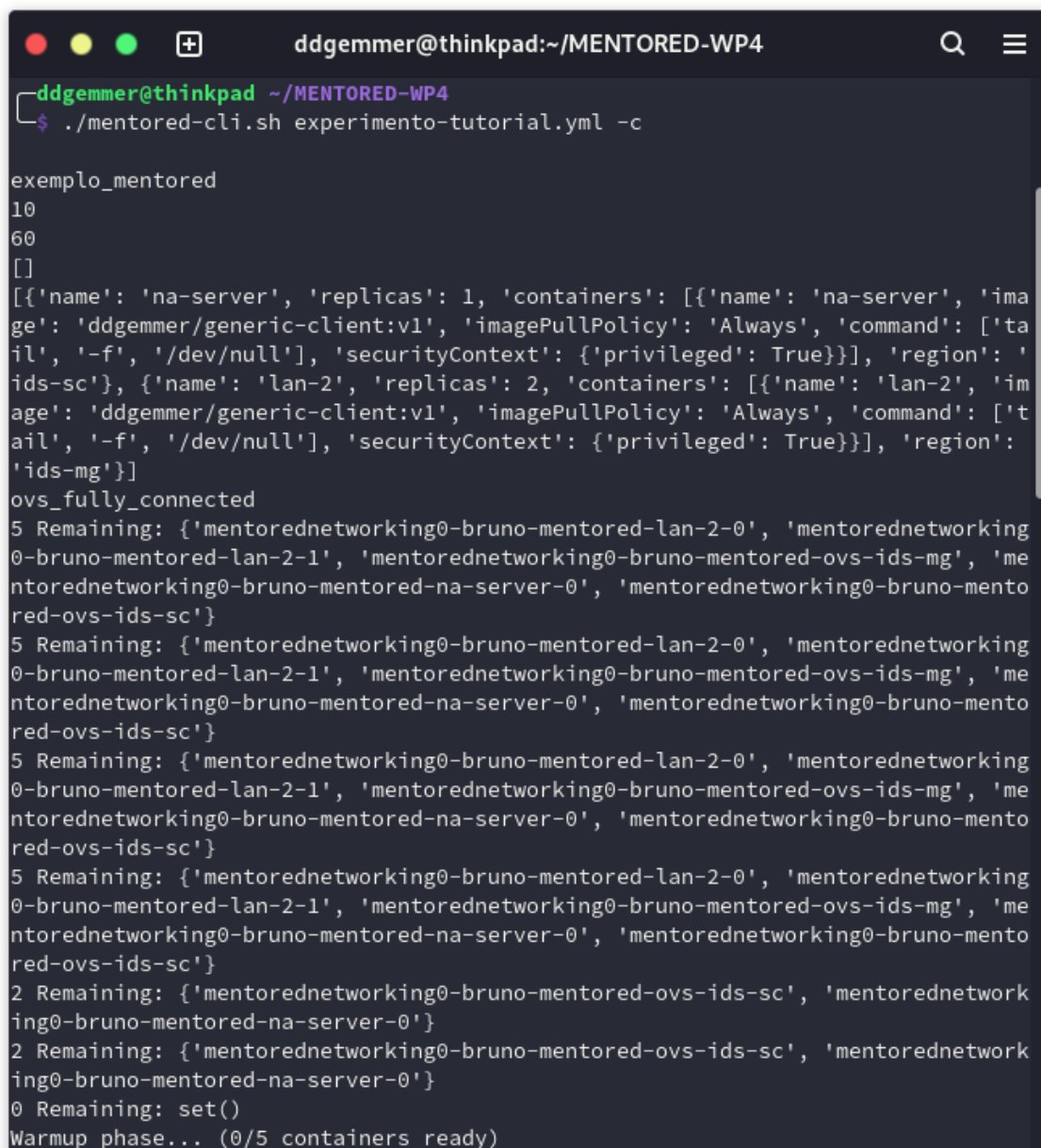
- **name:** Nome do experimento;
- **nodeactors:** Nós da topologia. Usuário deve inserir ao menos um nó;
- **- name:** Nome que identifica o nodeactor;
- **replicas:** Número de replicas;
- **image:** Imagem utilizada pelos *pods*;
- **imagePullPolicy:** Informa ao Kubernetes para extrair a imagem do registro;
- **command:** Comando utilizado para manter um contêiner ativo;
- **privileged:** Permissão atribuída ao *pod*;
- **region:** Nome de um worker no ids. Caso seja 'auto', o serviço irá identificar automaticamente um worker para instanciar os containers;
- **topology:** Cria um OVS (*Open Virtual Switch*) para cada região utilizada. Todos os nós de uma região terão um *link* com seu OVS na interface de rede chamada '*ovs-link*'.

A execução do experimento é através da utilização do cliente MENTORED seguido código *YML* com o parâmetro "-c". Nesse exemplo o arquivo *YML* foi denominado *experimento-tutorial.yml* e o comando completo para execução é:

```
$ ./mentored-cli.sh experimento-tutorial.yml -c
```

Como pode ser observado na Figura 5 durante a execução do experimento é retornado um *log* com informações da criação da topologia que foi configurada no Código 2.

Figura 5: Execução do Experimento



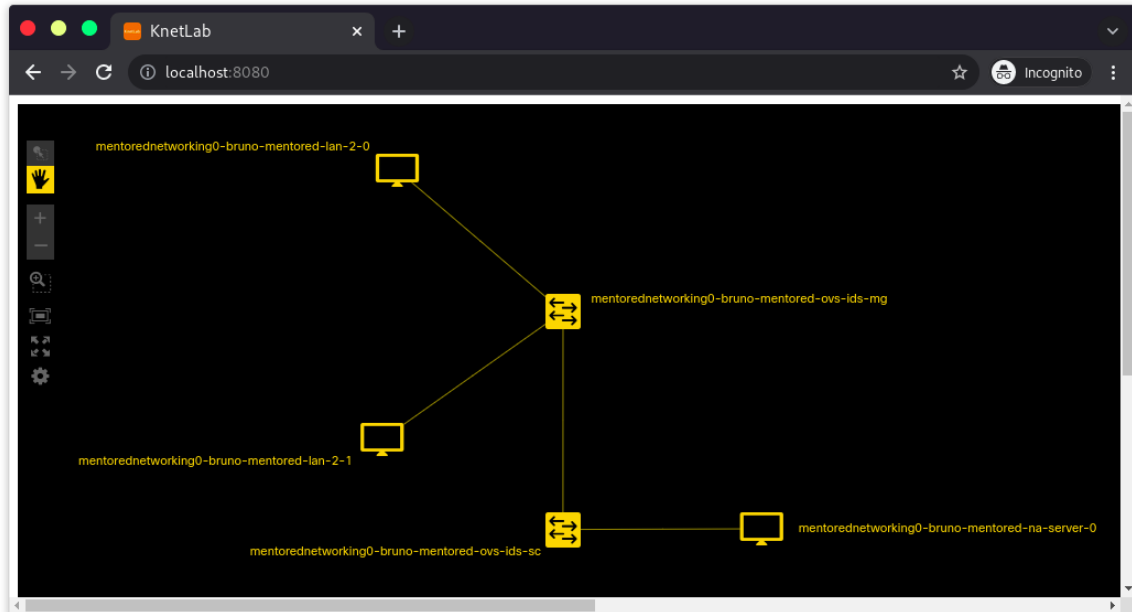
```
ddgemmer@thinkpad:~/MENTORED-WP4
$ ./mentored-cli.sh experimento-tutorial.yml -c

exemplo_mentored
10
60
[]
[{'name': 'na-server', 'replicas': 1, 'containers': [{'name': 'na-server', 'image': 'ddgemmer/generic-client:v1', 'imagePullPolicy': 'Always', 'command': ['tail', '-f', '/dev/null'], 'securityContext': {'privileged': True}}], 'region': 'ids-sc'}, {'name': 'lan-2', 'replicas': 2, 'containers': [{'name': 'lan-2', 'image': 'ddgemmer/generic-client:v1', 'imagePullPolicy': 'Always', 'command': ['tail', '-f', '/dev/null'], 'securityContext': {'privileged': True}}], 'region': 'ids-mg'}]
ovs_fully_connected
5 Remaining: {'mentorednetworking0-bruno-mentored-lan-2-0', 'mentorednetworking0-bruno-mentored-lan-2-1', 'mentorednetworking0-bruno-mentored-ovs-ids-mg', 'mentorednetworking0-bruno-mentored-na-server-0', 'mentorednetworking0-bruno-mentored-ovs-ids-sc'}
5 Remaining: {'mentorednetworking0-bruno-mentored-lan-2-0', 'mentorednetworking0-bruno-mentored-lan-2-1', 'mentorednetworking0-bruno-mentored-ovs-ids-mg', 'mentorednetworking0-bruno-mentored-na-server-0', 'mentorednetworking0-bruno-mentored-ovs-ids-sc'}
5 Remaining: {'mentorednetworking0-bruno-mentored-lan-2-0', 'mentorednetworking0-bruno-mentored-lan-2-1', 'mentorednetworking0-bruno-mentored-ovs-ids-mg', 'mentorednetworking0-bruno-mentored-na-server-0', 'mentorednetworking0-bruno-mentored-ovs-ids-sc'}
5 Remaining: {'mentorednetworking0-bruno-mentored-lan-2-0', 'mentorednetworking0-bruno-mentored-lan-2-1', 'mentorednetworking0-bruno-mentored-ovs-ids-mg', 'mentorednetworking0-bruno-mentored-na-server-0', 'mentorednetworking0-bruno-mentored-ovs-ids-sc'}
2 Remaining: {'mentorednetworking0-bruno-mentored-ovs-ids-sc', 'mentorednetworking0-bruno-mentored-na-server-0'}
2 Remaining: {'mentorednetworking0-bruno-mentored-ovs-ids-sc', 'mentorednetworking0-bruno-mentored-na-server-0'}
0 Remaining: set()
Warmup phase... (0/5 containers ready)
```

#### 4. Topologia KNetLab

O KNetLab é uma iniciativa da GCI e consiste em uma ferramenta que permite a criação de redes em *containers* utilizando princípios *Cloud Native*. Como pode ser observado na Figura 6, a partir de um navegador *web* é possível analisar a topologia que foi criada com o Código 2 baseado na Figura 4.

Figura 6: Topologia do Knetlab



## 5. Acesso aos *pods*

Por meio de um navegador *Web* é possível efetuar o acesso aos *pods* criados na Sessão 3. Através desse acesso é possível a execução e ajustes do experimento. A Figura 7 mostra o retorno após a execução do comando utilizado para listar os *pods* presente no *namespace*.

```
$ kubectl get pods -n mentored-lab11
```

Figura 7: Listagem dos *pods*

A captura de tela mostra uma interface de terminal web. No topo, há uma barra de endereço com o URL 'mentored-testbed.cafeexpresso.rnp.br:8080/terminal/?token=7t4l7tjkhzb1y53mpao4'. O terminal exibe a seguinte saída:

```
Welcome to Web Kubectl, try kubectl --help.
> kubectl get pod -n mentored-lab11
NAME                                READY   STATUS    RESTARTS   AGE
knetlab-operator-5d78f648df-mzvqg   1/1     Running   0           38m
mentorednetworking0-bruno-mentored-lan-2-0-67bcd748f7-rhlhv  1/1     Running   0           36m
mentorednetworking0-bruno-mentored-lan-2-1-96f8789d9-5pb5d   1/1     Running   0           36m
mentorednetworking0-bruno-mentored-na-server-0-b7865dc84-h6hxd 1/1     Running   0           36m
mentorednetworking0-bruno-mentored-ovs-ids-mg-d7fb8c8d8-rhhsz  1/1     Running   0           36m
mentorednetworking0-bruno-mentored-ovs-ids-sc-857dc754b6-rvjn2 1/1     Running   0           36m
```



Como é possível observar na Figura 7 é exibido o nome do *pod*s abaixo de "NAME". Com a utilização do nome e do comando *kubectl exec* é possível iniciar uma sessão de *shell* em um *pod* que está em execução. O comando utilizado para a execução é;

```
$ kubectl exec -it NAME -- /bin/sh
```

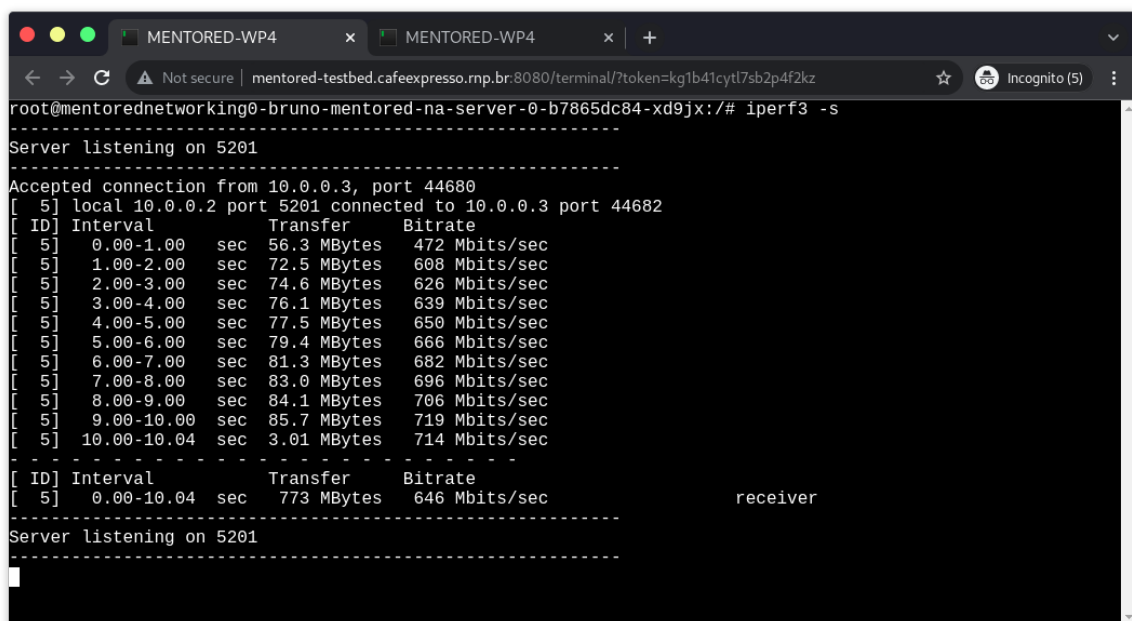
1. **kubectl exec**: Permite a conexão com o *container*;
2. **-it**: "i" modo interativo, "t" Aloca uma pseudo TTY;
3. **NAME**: Nome do *pod*
4. **- /bin/sh**: É um executavel que representa o *shell* do sistema

## 6. Comunicação entre os *pod*s

Após ser iniciada a sessão de *shell* é possível a interação com o *pod*. Nesse exemplo é iniciado duas sessões *shell* em dois *pod*s com o objetivo de mostrar a comunicação entre eles. O primeiro *pod* denominado na-server no Código 2 e localizado no IDS-SC pode ser observado na Figura 8 é utilizado como servidor.

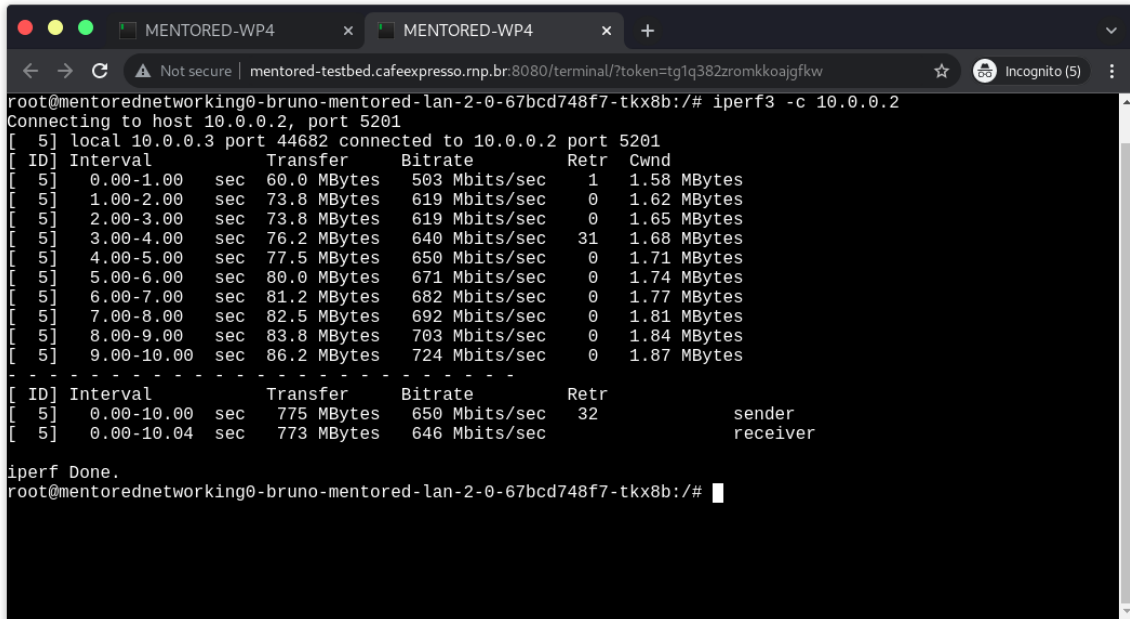
O segundo *pod* denominado lan-2 no mesmo arquivo está localizado no IDS-MG é utilizado como cliente e é possível ser observado na Figura 9. Como é possível perceber através do resultado das Figuras 8 e 9 existe a comunicação entre os *pod*s.

Figura 8: *Pod* Servidor



```
root@mentorednetworking0-bruno-mentored-na-server-0-b7865dc84-xd9jx:/# iperf3 -s
Server listening on 5201
Accepted connection from 10.0.0.3, port 44680
[ 5] local 10.0.0.2 port 5201 connected to 10.0.0.3 port 44682
[ ID] Interval      Transfer    Bitrate
[ 5] 0.00-1.00    sec 56.3 MBytes 472 Mbits/sec
[ 5] 1.00-2.00    sec 72.5 MBytes 608 Mbits/sec
[ 5] 2.00-3.00    sec 74.6 MBytes 626 Mbits/sec
[ 5] 3.00-4.00    sec 76.1 MBytes 639 Mbits/sec
[ 5] 4.00-5.00    sec 77.5 MBytes 650 Mbits/sec
[ 5] 5.00-6.00    sec 79.4 MBytes 666 Mbits/sec
[ 5] 6.00-7.00    sec 81.3 MBytes 682 Mbits/sec
[ 5] 7.00-8.00    sec 83.0 MBytes 696 Mbits/sec
[ 5] 8.00-9.00    sec 84.1 MBytes 706 Mbits/sec
[ 5] 9.00-10.00   sec 85.7 MBytes 719 Mbits/sec
[ 5] 10.00-10.04  sec 3.01 MBytes 714 Mbits/sec
[ ID] Interval      Transfer    Bitrate
[ 5] 0.00-10.04   sec 773 MBytes 646 Mbits/sec
Server listening on 5201
```

Figura 9: Pod Cliente



```
root@mentorednetworking0-bruno-mentored-lan-2-0-67bcd748f7-tkx8b:/# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 5] local 10.0.0.3 port 44682 connected to 10.0.0.2 port 5201
[ ID] Interval      Transfer    Bitrate    Retr  Cwnd
[ 5]  0.00-1.00    sec   60.0 MBytes  503 Mbits/sec    1   1.58 MBytes
[ 5]  1.00-2.00    sec   73.8 MBytes  619 Mbits/sec    0   1.62 MBytes
[ 5]  2.00-3.00    sec   73.8 MBytes  619 Mbits/sec    0   1.65 MBytes
[ 5]  3.00-4.00    sec   76.2 MBytes  640 Mbits/sec   31   1.68 MBytes
[ 5]  4.00-5.00    sec   77.5 MBytes  650 Mbits/sec    0   1.71 MBytes
[ 5]  5.00-6.00    sec   80.0 MBytes  671 Mbits/sec    0   1.74 MBytes
[ 5]  6.00-7.00    sec   81.2 MBytes  682 Mbits/sec    0   1.77 MBytes
[ 5]  7.00-8.00    sec   82.5 MBytes  692 Mbits/sec    0   1.81 MBytes
[ 5]  8.00-9.00    sec   83.8 MBytes  703 Mbits/sec    0   1.84 MBytes
[ 5]  9.00-10.00   sec   86.2 MBytes  724 Mbits/sec    0   1.87 MBytes
-----
[ ID] Interval      Transfer    Bitrate    Retr
[ 5]  0.00-10.00   sec   775 MBytes  650 Mbits/sec   32
[ 5]  0.00-10.04   sec   773 MBytes  646 Mbits/sec
      sender
      receiver

iperf Done.
root@mentorednetworking0-bruno-mentored-lan-2-0-67bcd748f7-tkx8b:/#
```

Ao concluir o experimento é possível a exclusão do mesmo por meio da utilização do cliente MENTORED através do comando:

```
$ ./mentored-cli.sh experimento-tutorial.yml -d
```

## 7. Comandos

Lista com os principais comandos utilizados para a manipulação das ferramentas utilizadas no *testbed* MENTORED para os experimentos.

### 7.1. Comandos básicos do Docker

```
$ docker ps # Lista todos os containe.
$ docker rm # Remove um ou mais containe.
$ docker images # Lista as imagens disponíveis no
→ \textit{host}.
$ docker rmi # Remove uma ou mais imagens.
$ docker run # Executa um comando em um novo containe.
$ docker exec # Executa uma instrução dentro de um
→ \textit{container}.

$ docker start # Inicia um containe parado.
$ docker stop # Para um containe que esteja em execução.
$ docker restart # Reinicia o containe.
$ docker stats # Exibe informações de uso de CPU, memória e
→ rede.
```

```
$ docker pull # Faz o pull de uma imagem.
$ docker push # Faz o push de uma imagem.
$ docker build # Cria uma imagem a partir de um Dockerfile.
$ docker commit # Cria uma imagem a partir de um contêiner.

$ docker cp # Copia arquivos ou diretórios do
→ \textit{container} para o \textit{host}.
$ docker rename # Renomeia o containe.
$ docker tag # Aplica uma tag em uma imagem.
```

## 7.2. Parâmetros do Docker

```
$ -d # Execução do \textit{container} em
→ \textit{background}.
$ -i # Modo interativo.
$ -t # Aloca uma pseudo TTY.
$ --rm # Remove o \textit{container} após finalização.
$ --name # Nomear o \textit{container}.
$ -v # Mapeamento de volume.
$ -p # Mapeamento de porta.
$ -m # Limitar o uso de memória RAM.
$ -c # Balancear o uso de CPU.
```

## 7.3. Exemplo básico de utilização do Docker

```
$ docker run ubuntu --name ubuntu. # Cria um
→ \textit{container} com a imagem do ubuntu com o nome
→ ubuntu.
$ docker exec -it ubuntu /bin/bash. # Executa a TTY em modo
→ interativo no \textit{container} ubuntu.
```

## 7.4. Comandos básicos do Kubernetes

```
$ kubectl cluster-info # Estado de um cluster.
$ kubectl get nodes -o wide # Lista os nodes.
$ kubectl get pods -o wide # Lista os pods.
$ kubectl get services # Informações sobre os serviços.

$ kubectl create deployment <NOME_DO_DEPLOY>
→ --image=<NOME_DA_IMAGEM>. # Cria o deploy
$ kubectl get deployment. # Lista o deploy
```

```
$ kubectl logs <NOME_DO_POD>. Lista os logs do pod.  
$ kubectl describe nodes <NOME_DO_NODE>. # Descreve  
→ camandos com saída detalhada do node.  
$ kubectl describe pods <NOME_DO_POD>. # Descreve camandos  
→ com saída detalhada do pod.  
  
$ kubectl delete pod <NOME_DO_POD>. # Deleta um pod.  
$ kubectl delete service <NOME_DO_SERVIÇO>. # Delata um  
→ serviço.  
$ kubectl delete deploy <NOME_DO_DEPLOY>. # Deleta um  
→ deploy.  
  
$ kubectl apply -f arquivo.yml. # Implementação a partir de  
→ um arquivo *.yml
```