Aluno: Pedro Leuchs

1. Atualização dos Pacotes:

Antes de instalar o Docker e o Docker Compose, é importante atualizar os pacotes do seu sistema para garantir que você tenha as últimas versões e dependências disponíveis. Utilize o comando sudo apt update para atualizar os pacotes:

sudo apt update

```
pedro@pedro-Nitro-AN517-54:~$ sudo apt update
sudo apt install docker.io
```

2. Instalação do Docker e Docker Compose:

Com os pacotes atualizados, você pode instalar o Docker e o Docker Compose utilizando o comando sudo apt install -y docker.io docker-compose. A flag -y indica que o processo de instalação será realizado sem a necessidade de confirmação manual.

3. Inicialização do Serviço Docker:

Após a instalação, o serviço Docker precisa ser iniciado para que você possa utilizar os containers. Utilize o comando sudo systemctl start docker para iniciar o serviço:

sudo systemctl start docker

```
pedro@pedro-Nitro-AN517-54:~/meu-wordpress-docker$ sudo systemctl start docker
[sudo] senha para pedro:
```

4. Habilitando o Início Automático do Docker:

Para evitar ter que iniciar o serviço Docker manualmente a cada reinicialização do sistema, você pode habilitá-lo para iniciar automaticamente. Utilize o comando sudo systemctl enable docker para habilitar o início automático:

sudo systemctl enable docker

```
pedro@pedro-Nitro-AN517-54:~/meu-wordpress-docker$ sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
```

5. Verificação da Instalação:

Para verificar se o Docker foi instalado e está funcionando corretamente, você pode utilizar o comando docker info. Este comando exibirá informações sobre o ambiente Docker, como a versão instalada, o status do serviço e as configurações.

```
docker info
```

após isso:

6. Inicialização do Docker Swarm:

O Docker Swarm permite gerenciar containers em clusters, facilitando a escalabilidade e alta disponibilidade das aplicações. Para iniciar o Swarm, utilize o comando sudo docker swarm init:

```
sudo docker swarm init
```

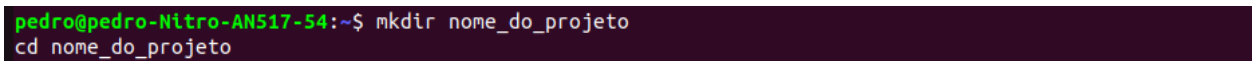
após isso:

Este comando irá inicializar o node gerenciador do Swarm na sua máquina local.

7. Criação da Pasta do Projeto:

Crie uma pasta para armazenar os arquivos de configuração e scripts do projeto:

```
mkdir nome_do_projeto
cd nome_do_projeto
```



8. Criação do Arquivo docker-compose.yml:

Utilize o editor de texto nano para criar o arquivo docker-compose.yml na pasta do projeto. Este arquivo define os serviços que serão implantados no Swarm:

```
pedro@pedro-Nitro-AN517-54:~/nome_do_projeto$ nano docker-compose.yml
```

digite:

```
version: '3.8'

services:
  wordpress:
    image: wordpress:latest
    ports:
      - '8080:80'
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress_password
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wordpress_data:/var/www/html
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure

  db:
    image: mysql:5.7
    environment:
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress_password
      MYSQL_ROOT_PASSWORD: root_password
    volumes:
      - db_data:/var/lib/mysql
    deploy:
      replicas: 1
      restart_policy:
```

```
      condition: on-failure

  redis:
    image: redis:latest
    ports:
      - '6379:6379'
    volumes:
      - redis_data:/data
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure

  prometheus:
    image: prom/prometheus:latest
    ports:
      - '9090:9090'
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure

  grafana:
    image: grafana/grafana:latest
    ports:
      - '3000:3000'
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana_data:/var/lib/grafana
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure

volumes:
  wordpress_data:
  db_data:
  redis_data:
  grafana_data:
```

```yaml
version: '3.8'
services:
  wordpress:
    image: wordpress:latest
    ports:
      - '8080:80'
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress_password
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wordpress_data:/var/www/html
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure

  db:
    image: mysql:5.7
    environment:
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress_password
      MYSQL_ROOT_PASSWORD: root_password
    volumes:
      - db_data:/var/lib/mysql
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure

  redis:
    image: redis:latest
    ports:
      - '6379:6379'
    volumes:
      - redis_data:/data
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure

  prometheus:
    image: prom/prometheus:latest
    ports:
      - '9090:9090'
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    deploy:
      replicas: 1
      restart_policy:
```

```
        condition: on-failure

  grafana:
    image: grafana/grafana:latest
    ports:
      - '3000:3000'
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana_data:/var/lib/grafana
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure

volumes:
  wordpress_data:
  db_data:
  redis_data:
  grafana_data:
```

Observações:

- A indentação do código YAML é crucial para o seu funcionamento. Certifique-se de seguir a indentação conforme o exemplo acima.
- As variáveis de ambiente WORDPRESS_DB_HOST, WORDPRESS_DB_USER, WORDPRESS_DB_PASSWORD, MYSQL_ROOT_PASSWORD e GF_SECURITY_ADMIN_PASSWORD devem ser configuradas de acordo com suas necessidades.
- Os volumes wordpress_data, db_data, redis_data e grafana_data armazenam os dados persistentes dos serviços.

9. Salvando o arquivo docker-compose.yml:

No editor nano, utilize Ctrl+O para salvar o arquivo e Ctrl+X para sair.

10. Criação do Arquivo prometheus.yml:

Crie o arquivo prometheus.yml na pasta do projeto para definir as configurações de coleta de métricas do Prometheus:

nano prometheus.yml

```
global:
  scrape_interval: 15s


scrape_configs:
  - job_name: 'wordpress'
    static_configs:
      - targets: ['wordpress:9100']
  - job_name: 'mysql'
    static_configs:
      - targets: ['db:3306']
  - job_name: 'redis'
    static_configs:
      - targets: ['redis:6379']
  - job_name: 'node'
    static_configs:
      - targets: ['node-exporter:9100']
```

```yaml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'wordpress'
    static_configs:
      - targets: ['wordpress:9100']
  - job_name: 'mysql'
    static_configs:
      - targets: ['db:3306']
  - job_name: 'redis'
    static_configs:
      - targets: ['redis:6379']
  - job_name: 'node'
    static_configs:
      - targets: ['node-exporter:9100']
```

11. Salvando o arquivo prometheus.yml:

No editor nano, utilize Ctrl+O para salvar o arquivo e Ctrl+X para sair.

12. Implantando a Stack no Docker Swarm:

Utilize o comando sudo docker stack deploy para implantar a stack definida no arquivo docker-compose.yml. Substitua nome_da_stack pelo nome que você deseja dar à sua stack:

sudo docker stack deploy -c docker-compose.yml nome_da_stack

```
pedro@pedro-Nitro-AN517-54:~/nome_do_projeto$ sudo docker stack deploy -c docker-compose.yml nome_da_stack
[sudo] senha para pedro:
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network nome_da_stack_default
Creating service nome_da_stack_grafana
Creating service nome_da_stack_wordpress
Creating service nome_da_stack_db
Creating service nome_da_stack_redis
Creating service nome_da_stack_prometheus
```

Este comando irá criar os serviços, containers e redes necessários para executar sua aplicação em modo clusterizado no Docker Swarm.

13. Verificando o Status dos Serviços:

Utilize o comando docker service ls para verificar o status dos serviços da sua stack:

docker service ls

```
pedro@pedro-Nitro-AN517-54:~/nome_do_projeto$ docker service ls
ID             NAME                      MODE         REPLICAS   IMAGE                    PORTS
g5esw7jeximu   nome_da_stack_db          replicated   0/1        mysql:5.7
6akvsok5ei6e   nome_da_stack_grafana     replicated   1/1        grafana/grafana:latest   *:3000->3000/tcp
tcoo4uoytn4k   nome_da_stack_prometheus  replicated   1/1        prom/prometheus:latest   *:9090->9090/tcp
i8muw8us40u5   nome_da_stack_redis       replicated   0/1        redis:latest             *:6379->6379/tcp
rngz81xrwoe7   nome_da_stack_wordpress   replicated   1/1        wordpress:latest         *:8080->80/tcp
```

A saída do comando deve listar todos os serviços definidos no arquivo docker-compose.yml, com seu estado atual (running, pending, etc.).

14. Verificando Imagens:

Caso algum serviço esteja com o estado 0/1 nas replicas, verifique se você possui todas as imagens necessárias no seu sistema. Utilize o comando docker images para listar as imagens disponíveis:

```
pedro@pedro-Nitro-AN517-54:~/nome_do_projeto$ docker images
REPOSITORY          TAG                   IMAGE ID       CREATED        SIZE
grafana/grafana     latest                97bf66938f26   19 hours ago   445MB
wordpress           latest                73ca9d5ac9d3   20 hours ago   685MB
prom/prometheus     latest                b74abbcc4eac   6 days ago     271MB
mysql               8.0                   8279f68ee77d   6 days ago     567MB
redis               <none>                aceb1262c1ea   4 weeks ago    117MB
redis               6.2                   0bff933a3fa7   4 weeks ago    106MB
prom/node-exporter  latest                0c6f6c1bdd47   4 weeks ago    23.3MB
mysql               latest                e9387c13ed83   7 weeks ago    578MB
trydockerleve       latest                e35116d61755   8 weeks ago    77.4MB
<none>              <none>                32756256b862   8 weeks ago    113MB
ubuntu              latest                de52d803b224   2 months ago   76.2MB
mysql               <none>                5107333e08a8   6 months ago   501MB
maven               3.6.3-openjdk-17-slim 850a4d5b96e4   3 years ago    415MB
```

15. Puxando Imagens Faltantes:

Se alguma imagem necessária estiver faltando, utilize o comando docker pull para puxá-la do Docker Hub:

no meu caso faltava redis e mysql

```
pedro@pedro-Nitro-AN517-54:~/nome_do_projeto$ docker pull redis
Using default tag: latest
latest: Pulling from library/redis
Digest: sha256:e422889e156ebea83856b6ff973bfe0c86bce867d80def228044eeecf925592b
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
pedro@pedro-Nitro-AN517-54:~/nome_do_projeto$ docker pull mysql:5.7
5.7: Pulling from library/mysql
Digest: sha256:4bc6bc963e6d8443453676cae56536f4b8156d78bae03c0145cbe47c2aad73bb
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
```

Repita o comando docker pull para cada imagem faltante.

16. Acessando o WordPress:

Após a implantação da stack, você pode acessar o site WordPress em:

- http://localhost:8080 (se você não alterou a porta no arquivo docker-compose.yml)
- http://localhost:80 (se você alterou a porta para 80 no arquivo docker-compose.yml)

17. Instalando e Configurando o Redis Object Cache:

Acesse o painel de administração do WordPress (geralmente em http://localhost:8080/wp-admin) e siga estes passos:

- **Acesse a aba de plugins.**
- **Clique em "Adicionar novo plugin".**
- **Procure por "Redis Object Cache".**
- **Clique em "Instalar agora" e depois em "Ativar".**
- **Acesse a página de configurações do Redis Object Cache.**
- **Clique em "ATIVAR O CACHE DE OBJETO".**

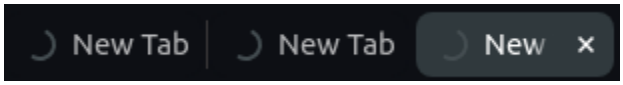
18. Acessando o Prometheus e Grafana:

- **Prometheus:** http://localhost:9090
- **Grafana:** http://localhost:3000 (usuário: admin, senha: admin)

19. Links Úteis:

- **WordPress:** http://localhost:8080 ou http://localhost:80 (conforme sua configuração)
- **Prometheus:** http://localhost:9090
- **Grafana:** http://localhost:3000 (usuário: admin, senha: admin)

conclusão:
infelizmente no meu caso não foi possivel concluir pois as abas para acessar ficaram em looping infinito e não consegui dar continuidade no projeto:




```
pedro@pedro-Nitro-AN517-54:~/nome_do_projeto$ docker service ls
ID             NAME                      MODE         REPLICAS   IMAGE                    PORTS
g5esw7jeximu   nome_da_stack_db          replicated   1/1        mysql:5.7
6akvsok5ei6e   nome_da_stack_grafana     replicated   1/1        grafana/grafana:latest   *:3000->3000/tcp
tcoo4uoytn4k   nome_da_stack_prometheus  replicated   1/1        prom/prometheus:latest   *:9090->9090/tcp
i8muw8us40u5   nome_da_stack_redis       replicated   1/1        redis:latest             *:6379->6379/tcp
rngz81xrwoe7   nome_da_stack_wordpress   replicated   1/1        wordpress:latest         *:8080->80/tcp
```