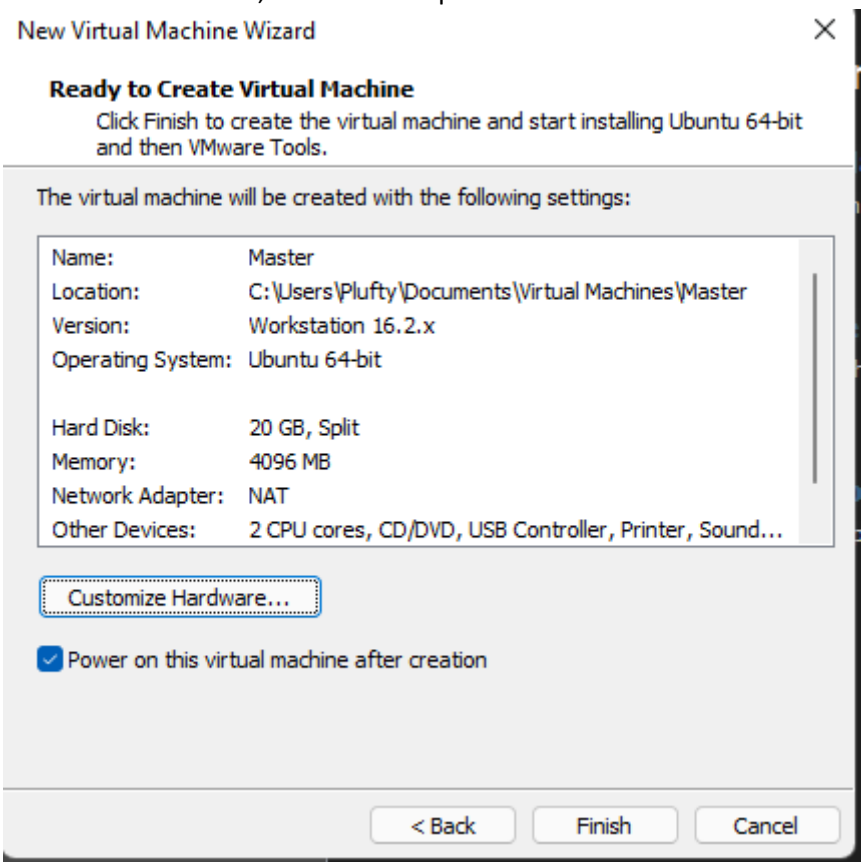
Gleidson Vinícius Gomes Barbosa - 6331

Cluster Kubernetes - Como fazer?

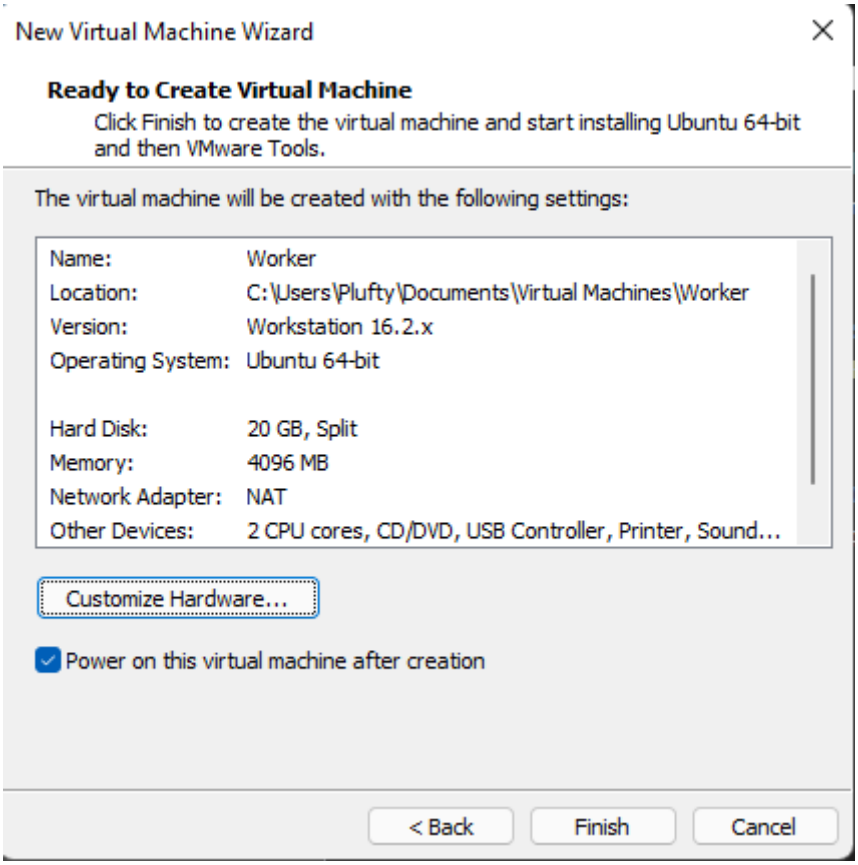
O procedimento será executado por mim em duas máquinas virtuais utilizando VMWare, então o primeiro passo será a criação de ambas as máquinas virtuais e ativar a virtualização das mesmas visto que será utilizado o KVM. Será instalado em ambas as máquinas o Ubuntu Server 20.04. Os procedimentos detalhados nesse relatório exceto pelos relacionados ao VMWare podem ser encontrados em:

- Installing kubeadm | Kubernetes
- Install Docker Engine on Ubuntu | Docker Documentation
- Ports and Protocols | Kubernetes
- Como abrir portas no Ubuntu Server? (vocepergunta.com)
- kubernetes - Kubelet service is not running. It seems like the kubelet isn't running or healthy - Stack Overflow
- Creating a cluster with kubeadm | Kubernetes
- Run a Stateless Application Using a Deployment | Kubernetes

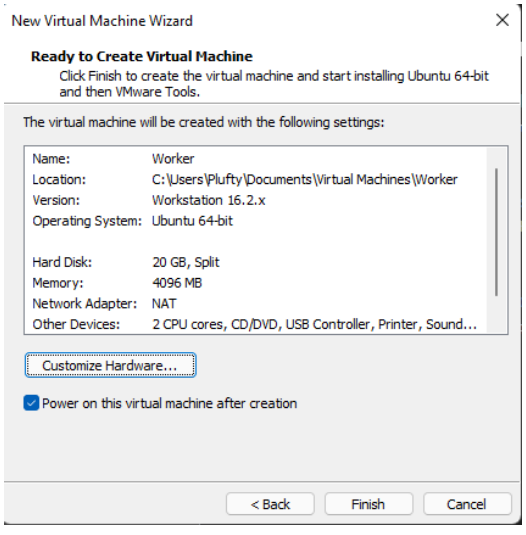
1. Criando a VM Master, ela será a máquina master desse cluster.

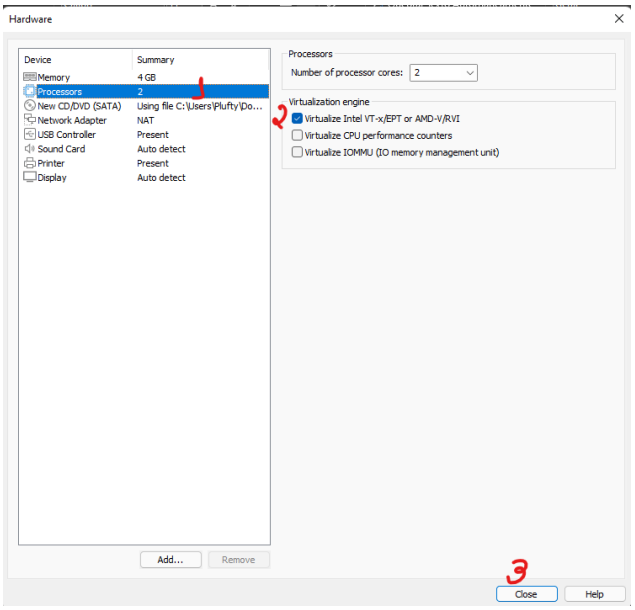


2. Criando a VM Worker que será a máquina cliente desse processo, ela será o único worker desse processo.

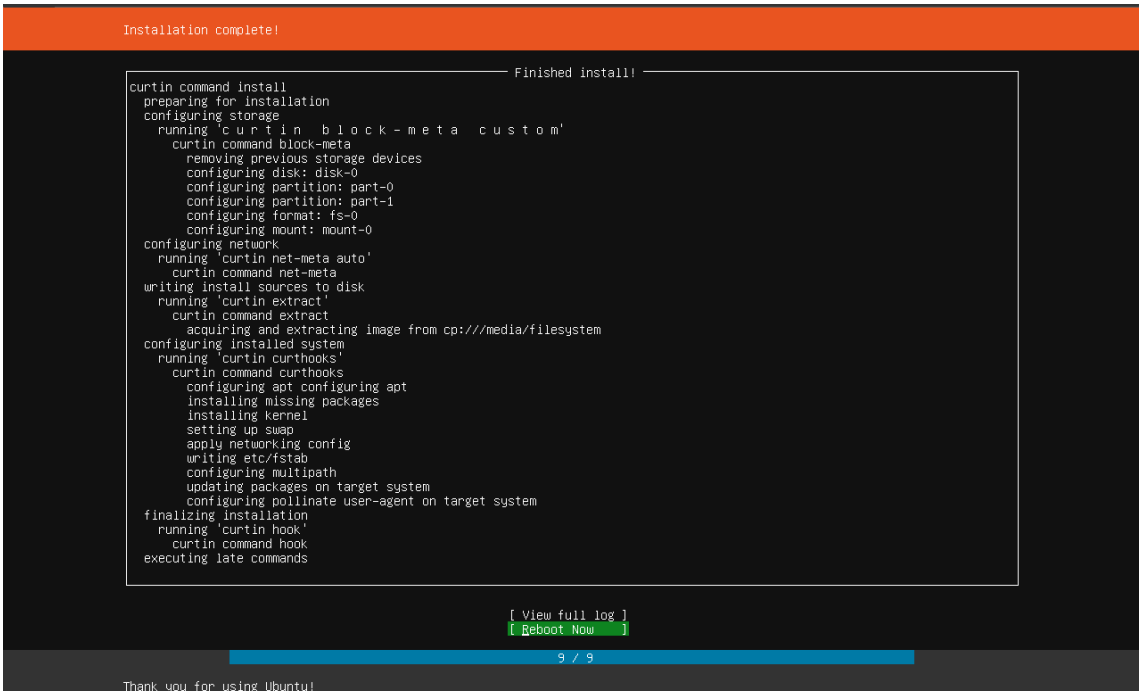


3. Ativando a virtualização no VMWare, ao clicar em Customize Hardware, selecione Processors e ative a virtualização.





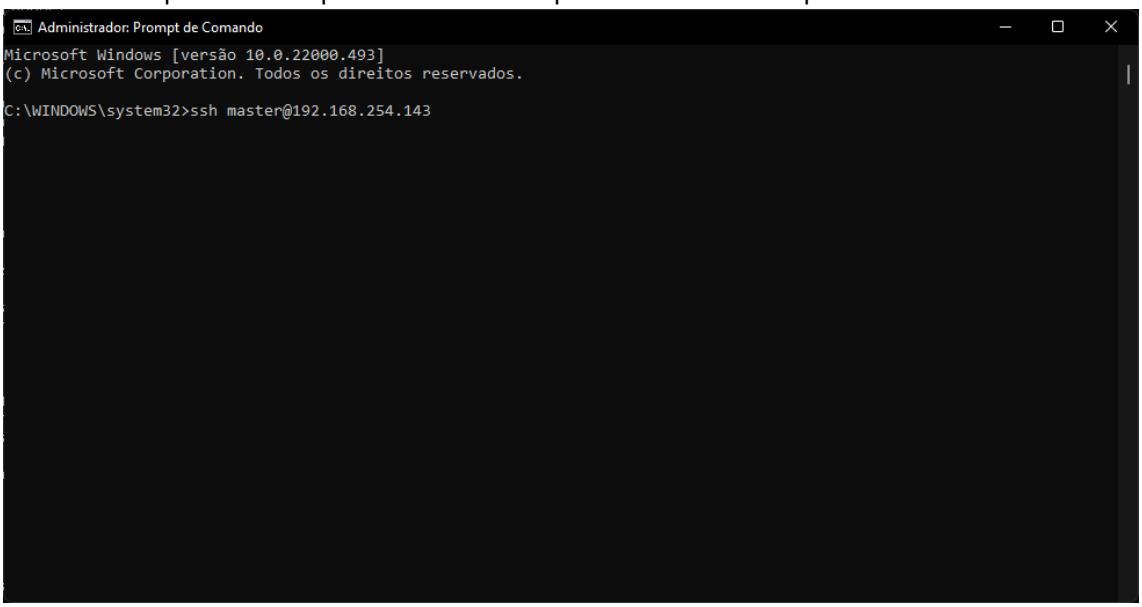
4. Instalação do SO, nesse caso irei utilizar o Ubuntu server 18.04



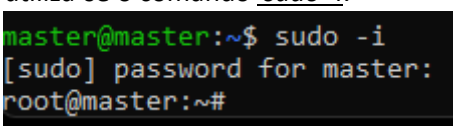
A instalação do Docker e Kubernetes

Neste momento estaremos instalando o Docker em ambas as máquinas, por isso colocarei prints de apenas uma delas, basta repetir o procedimento em ambas.

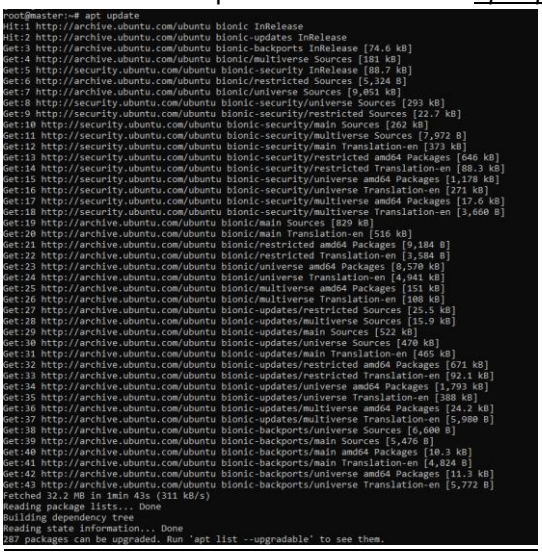
1. Como estamos trabalhando com máquinas virtuais, estarei acessando primeiramente via SSH a máquina Master para trabalhar sem problemas de desempenho do VMWare.



2. Logo após, entrarei em modo root para não ter problemas com permissões. Para isso utiliza-se o comando *sudo -i*.



3. Atualizaremos os pacotes do linux com *apt update*



4. O primeiro passo que faremos é desligar a memória swap com o comando *swapoff -a*

```
root@master:~# swapoff -a
```

5. Depois instalaremos o Docker com o comando *curl -fsSL https://get.docker.com | bash* após completar a instalação conferimos a versão com *docker --version*

```
root@master:~# curl -fsSL https://get.docker.com | bash
# Executing docker install script, commit: 93d2499759296ac1f9c510605fef85052a2c32be
```

```
root@master:~# docker --version
Docker version 20.10.13, build a224086
```

6. Nesse ponto, utilizaremos uma série de comandos para carregar os modulos necessários para o kubernetes e configurar as bridges, seguem os comandos:
modprobe br_netfilter
lsmod | grep br_netfilter
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl --system

```
root@master:~# modprobe br_netfilter
root@master:~# lsmod | grep br_netfilter
br_netfilter           24576  0
bridge                155648  1 br_netfilter
```

```
root@master:~# cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
> br_netfilter
> EOF

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl --systembr_netfilter
```

7. Agora verificaremos se as portas requeridas pelo kubernetes estão abertas. Para abrir usaremos os seguintes comandos:
iptables -I INPUT -p tcp --dport 6443 -j ACCEPT

iptables -I INPUT -p tcp --dport 2379 -j ACCEPT
iptables -I INPUT -p tcp --dport 2389 -j ACCEPT
iptables -I INPUT -p tcp --dport 10250 -j ACCEPT
iptables -I INPUT -p tcp --dport 10259 -j ACCEPT
iptables -I INPUT -p tcp --dport 10257 -j ACCEPT

8. Agora atualizaremos os pacotes do Linux com o comando *apt-get update*

```
root@master:~# apt-get update
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:2 http://archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Hit:5 https://download.docker.com/linux/ubuntu bionic InRelease
Fetched 252 kB in 3s (76.7 kB/s)
Reading package lists... Done
```

9. Feito isso, alguns pré requisitos do kubernetes, como os certificados com o comando *apt-get install -y apt-transport-https ca-certificates curl*

```
root@master:~# sudo apt-get install -y apt-transport-https ca-certificates curl
Reading package lists... Done
Building dependency tree
Reading state information... Done
ca-certificates is already the newest version (20210119~18.04.2).
curl is already the newest version (7.58.0-2ubuntu3.16).
apt-transport-https is already the newest version (1.6.14).
0 upgraded, 0 newly installed, 0 to remove and 280 not upgraded.
```

10. Adicionaremos uma chave publica do google cloud com o comando:
curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg

    logo após adicionaremos o repositório kubernetes com o comando:
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list

```
root@master:~# curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg
root@master:~# echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main
root@master:~#
```

11. Após adicionar o repositório, atualizaremos novamente os pacotes e por fim instalaremos o kubbernetes com a seguinte sequência de comandos:
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl

```
root@master:~# sudo apt-get update
root@master:~# sudo apt-get install -y kubelet kubeadm kubectl
root@master:~# sudo apt-mark hold kubelet kubeadm kubectl
```

12. Executaremos o comando *kubeadm – init* para iniciar nosso mastere ao fim deste comando será gerado um token, lembre-se de salvá-lo, ele será necessário no futuro. No meu caso é o seguinte:

    kubeadm join 192.168.254.147:6443 --token vmurg9.m20zvlq7pd7myruf \ --discovery-token-ca-cert-hash

sha256:cd930035a44414aba217c3081632ad37035caaead039032dffd41b60bde26e0d

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.254.147:6443 --token vmurg9.m20zvlq7pd7myruf \
        --discovery-token-ca-cert-hash sha256:cd930035a44414aba217c3081632ad37035caaead039032dffd41b60bde26e0d
root@master:~#
```

13. Agora iremos configurar o kubernetes de acordo com a própria resposta anterior utilizando os comandos indicados:
    mkdir -p $HOME/.kube
    sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
    sudo chown $(id -u):$(id -g) $HOME/.kube/config

```
root@master:~#   mkdir -p $HOME/.kube
udo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/configroot@master:~#   sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
root@master:~#   sudo chown $(id -u):$(id -g) $HOME/.kube/config
root@master:~# kubectl getnodes
error: unknown command "getnodes" for "kubectl"
```

14. Verificamos com o comando *kubectl get nodes* que já temos nosso node Master no estado de NotReady.

```
root@master:~# kubectl get nodes
NAME     STATUS     ROLES                  AGE   VERSION
master   NotReady   control-plane,master   64m   v1.23.4
```

15. Agora iremos adicionar nosso woker no nosso cluster. Precisaremos do comando com o token no passo 12, basta copiá-lo e executar na máquina worker.

```
root@worker:~# kubeadm join 192.168.254.147:6443 --token vmurg9.m20zvlq7pd7myruf \
>         --discovery-token-ca-cert-hash sha256:cd930035a44414aba217c3081632ad37035caaead039032dffd41b60bde26e0d
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
W0311 04:10:52.793496   72410 utils.go:69] The recommended value for "resolvConf" in "KubeletConfiguration" is: /run/
systemd/resolve/resolv.conf; the provided value is: /run/systemd/resolve/resolv.conf
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

16. Agora verificaremos no nosso master com o comando *kubectl get nodes* que temos o master e o worker no nosso cluster, ambos em estado NotReady.

```
root@master:~# kubectl get nodes
NAME     STATUS     ROLES                  AGE   VERSION
master   NotReady   control-plane,master   74m   v1.23.4
worker   NotReady   <none>                 14s   v1.23.4
```

17. Agora executaremos o comando que irá deixar nosso status como pronto:
    kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"

```
root@master:~# kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n
')"
serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created
```

18. Executamos novamente o comando *kubectl get nodes* e observamos que nosso cluster está prontinho.

```
root@master:~# kubectl get nodes
NAME     STATUS   ROLES                  AGE   VERSION
master   Ready    control-plane,master   89m   v1.23.4
worker   Ready    <none>                 15m   v1.23.4
```

Fazendo o Deployment de uma aplicação

Agora faremos um deployment de uma aplicação no nosso cluster kubernetes. Para esse passo escolhemos a aplicação nginx.

1. Para isso iremos iniciar criando um arquivo yaml com o comando kubectl apply -f https://k8s.io/examples/application/deployment.yaml

```
root@master:~# kubectl apply -f https://k8s.io/examples/application/deployment.yaml
deployment.apps/nginx-deployment created
```

2. Após isso iremos verificar as informações sobre nosso deployment com o comando kubectl describe deployment nginx-deployment

```
root@master:~# kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Sat, 12 Mar 2022 23:47:06 +0000
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx:1.14.2
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deployment-9456bbbf9 (2/2 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  3m20s  deployment-controller  Scaled up replica set nginx-deployment-9456bbbf9 to 2
```

3. Depois verificaremos os pods gerados com o comando kubectl get pods -l app=nginx

```
root@master:~# kubectl get pods -l app=nginx
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-9456bbbf9-7hqdc    1/1     Running   0          4m26s
nginx-deployment-9456bbbf9-z5st5    1/1     Running   0          4m26s
```

4. Agora podemos verificar as informações sobre os pods gerados com o comando kubectl describe pod <pode-name> no nosso caso, usaremos como exemplo o pod de nome nginx-deployment-9456bbbf9-7hqdc

```
root@master:~#  kubectl describe pod nginx-deployment-9456bbbf9-7hqdc
Name:         nginx-deployment-9456bbbf9-7hqdc
Namespace:    default
Priority:     0
Node:         worker/192.168.254.148
Start Time:   Sat, 12 Mar 2022 23:47:06 +0000
Labels:       app=nginx
              pod-template-hash=9456bbbf9
Annotations:  <none>
Status:       Running
IP:           10.44.0.2
IPs:
  IP:           10.44.0.2
Controlled By:  ReplicaSet/nginx-deployment-9456bbbf9
Containers:
  nginx:
    Container ID:   docker://0fcc1cc6bce384f33ed1bfa74b000697f83ae5280abcdc9842dbdff5b01753f8
    Image:          nginx:1.14.2
    Image ID:       docker-pullable://nginx@sha256:f7988fb6c02e0ce69257d9bd9cf37ae20a60f1df7563c3a2a6abe24160306b8d
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sat, 12 Mar 2022 23:47:19 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-twbp8 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  kube-api-access-twbp8:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  5m29s  default-scheduler  Successfully assigned default/nginx-deployment-9456bbbf9-7hqdc to work
er
  Normal  Pulling    5m28s  kubelet            Pulling image "nginx:1.14.2"
  Normal  Pulled     5m17s  kubelet            Successfully pulled image "nginx:1.14.2" in 11.351030303s
  Normal  Created    5m16s  kubelet            Created container nginx
  Normal  Started    5m16s  kubelet            Started container nginx
```

Verificamos assim que nosso pod está rodando em nosso worker, ou seja, nosso deployment funcionou, Uhu!

Erro no Kubelet

Fique atento a esse tópico pois temos alguns passos que só serão executados no nosso master, esses passos estão marcados com a tag **[APENAS NA MÁQUINA MASTER],** se por acaso for executado no worker, pode comprometer todo nosso cluster.

1. Existe a possibilidade de alguns erros serem apresentados devido a problemas com o kubeletes, o seguinte erro é apresentado:

```
        Unfortunately, an error has occurred:
                timed out waiting for the condition

        This error is likely caused by:
                - The kubelet is not running
                - The kubelet is unhealthy due to a misconfiguration of the node in some way (required cgroups disabled)

        If you are on a systemd-powered system, you can try to troubleshoot the error with the following commands:
                - 'systemctl status kubelet'
                - 'journalctl -xeu kubelet'

        Additionally, a control plane component may have crashed or exited when started by the container runtime.
        To troubleshoot, list all containers using your preferred container runtimes CLI.

        Here is one example how you may list all Kubernetes containers running in docker:
                - 'docker ps -a | grep kube | grep -v pause'
                Once you have found the failing container, you can inspect its logs with:
                - 'docker logs CONTAINERID'

error execution phase wait-control-plane: couldn't initialize a Kubernetes cluster
To see the stack trace of this error execute with --v=5 or higher
```

2. Nesse caso seguiremos os seguintes procedimentos:
   Primeiro desabilitaremos novamente o swap e após isso verificaremos o grupo do docker com a seguinte sequência de comandos:
   sudo sed -i '/ swap / s/^/#/' /etc/fstab
   sudo swapoff -a
   docker info |grep -i cgroup

```
root@master:~# sudo sed -i '/ swap / s/^/#/' /etc/fstab
root@master:~# sudo swapoff -a
root@master:~# docker info |grep -i cgroup
```

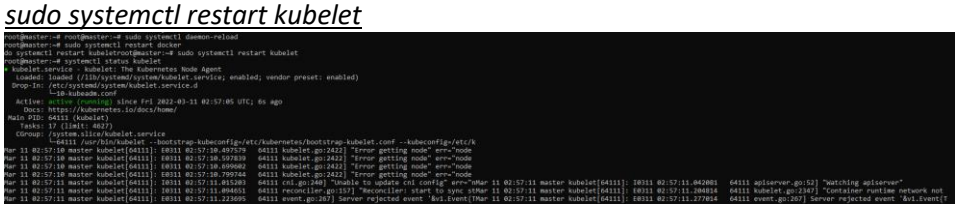
3. Após isso editaremos o arquivo com o comando *nano /etc/docker/daemon.json* e adicionaremos o seguinte texto:
   {
   "exec-opts": ["native.cgroupdriver=systemd"]
   }

```
root@master:~# nano /etc/docker/daemon.json
```

```
  GNU nano 2.9.3

{
    "exec-opts": ["native.cgroupdriver=systemd"]
}
```

4. Agora executaremos a seguinte sequência de comandos para reiniciar alguns serviços e verificar o kubelet:
   sudo systemctl daemon-reload
   sudo systemctl restart docker
   sudo systemctl restart kubelet



5. **[APENAS NA MÁQUINA MASTER]** Depois iremos refazer a fase de tokens do kubeadm com o comando
   kubeadm init phase bootstrap-token

```
root@master:~# kubeadm init phase bootstrap-token
[bootstrap-token] Using token: b4ewfs.9bqtvslvy14uykej
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootsroot@master:~#
[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
```

6. Agora desabilitaremos o firewall com o comando *ufw disable*

```
root@master:~# ufw disable
Firewall stopped and disabled on system startup
```

7. **[APENAS NA MÁQUINA MASTER]** Finalmente executaremos o comando *kubeadm init –ignore-preflight-errors=all*

```
root@master:~# kubeadm init --ignore-preflight-errors=all
```

Corrigimos nossos erros, agora voltamos ao passo que paramos e seguimos para concluir nosso cluster, Felicidade meus amigos!