Web Service and Cloud Based - 2024 February

The project is structured as follows.

- wscb
  - docker-compose.yml
  - README.md
  - **Authentication_Service**
    - app.py
    - models.py
    - mysql_config.py
    - utils.py
    - Dockerfile
    - wait-for-it.sh
    - requirements.txt
  - **Url_Shorten_Service**
    - app.py
    - models.py
    - mysql_config.py
    - utils.py
    - Dockerfile
    - wait-for-it.sh
    - requirements.txt
  - **mysql**
    - Dockerfile
    - init_db.sql
  - **nginx**
    - Dockerfile
    - nginx.conf
  - test
    - A bunch of test scripts of Canvas
  - docs
    - A bunch of assignment descriptions of Canvas
    - Reports
  - deprecate

- Codes no longer used

Container Virtualization

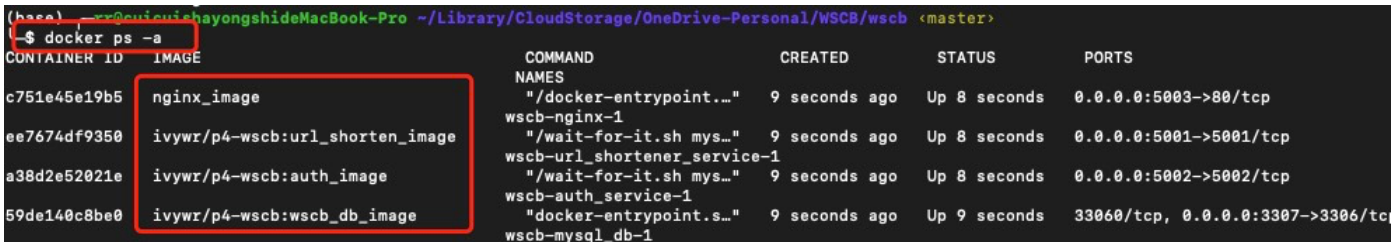
This part is about to start two Flask applications (Authentication_Service and Url_Shorten_Service) and MySQL database containers using Docker Compose and let the containers to communicate with database and the database data to persist. We use **nginx proxy** to make two services available on one single entry.

How to Run Demo with Docker Compose

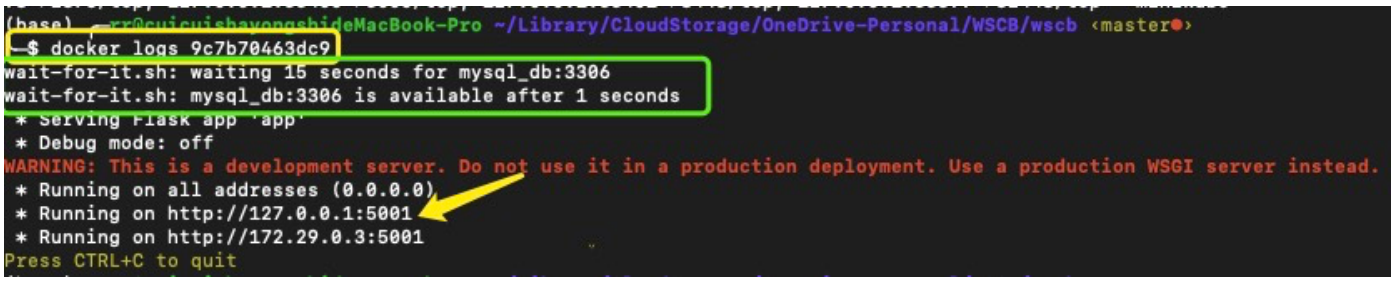
Navigate to the project directory (that has docker-compose.yml there)and execute the following command:

```
docker-compose up -d
```


```
docker ps
```




```
docker logs <url_shorten_container_id>
```



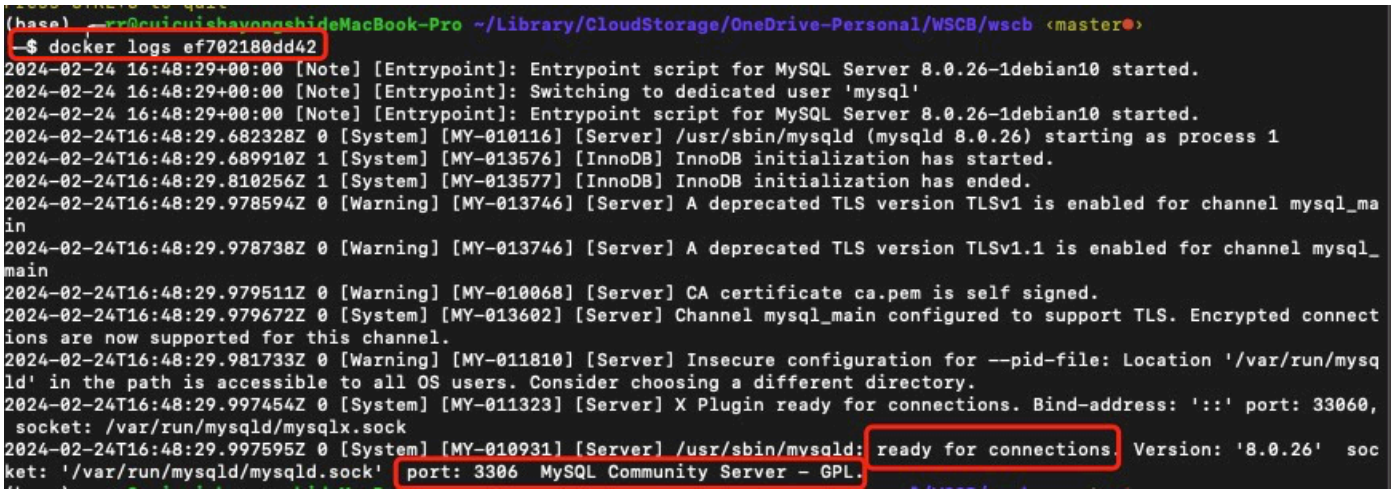
By checking the container logs, you can see that the url shorten service has been started on port 5001 (which we specified).

Inside the green box, it shows that we used the *wait-for-it.sh* script before running the python command to start the flask application. The reason for using this third-party wait script in the startup command of the Flask app is to **wait for the database port to become available and then start flask application**.

> You can get wait-for-it.sh by executing this on command line:
>
> wget https://raw.githubusercontent.com/vishnubob/wait-for-it/master/wait-for-it.sh

Similarly you can see authentication service has been started on port 5002 (which we speicified).

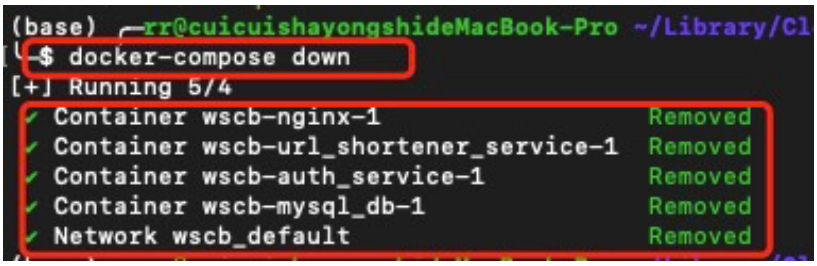
You can also check logs of MySQL database and see the mysql starts on port 3306. Note that this port can only be accessed within the containers (can only access by URL_shorten_service_container and Authentication_service_container).



You can use Postman to test by sending requests to http://0.0.0.0:5003/ for url shorten service and http://0.0.0.0:5003/users/ for identity authentication service. We use **nginx proxy** to make two services available on one single entry.

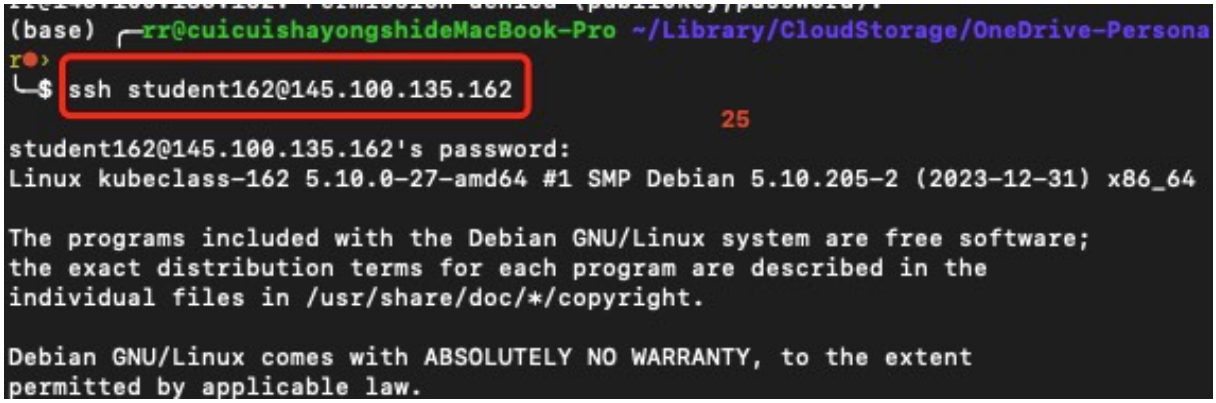
To remove the containers started by docker-compose up, you can use

```
docker-compose down
```



Containers Orchestration

Envrionment: VMs of University of Amsterdan. 首先用ssh登录到uva的虚拟机

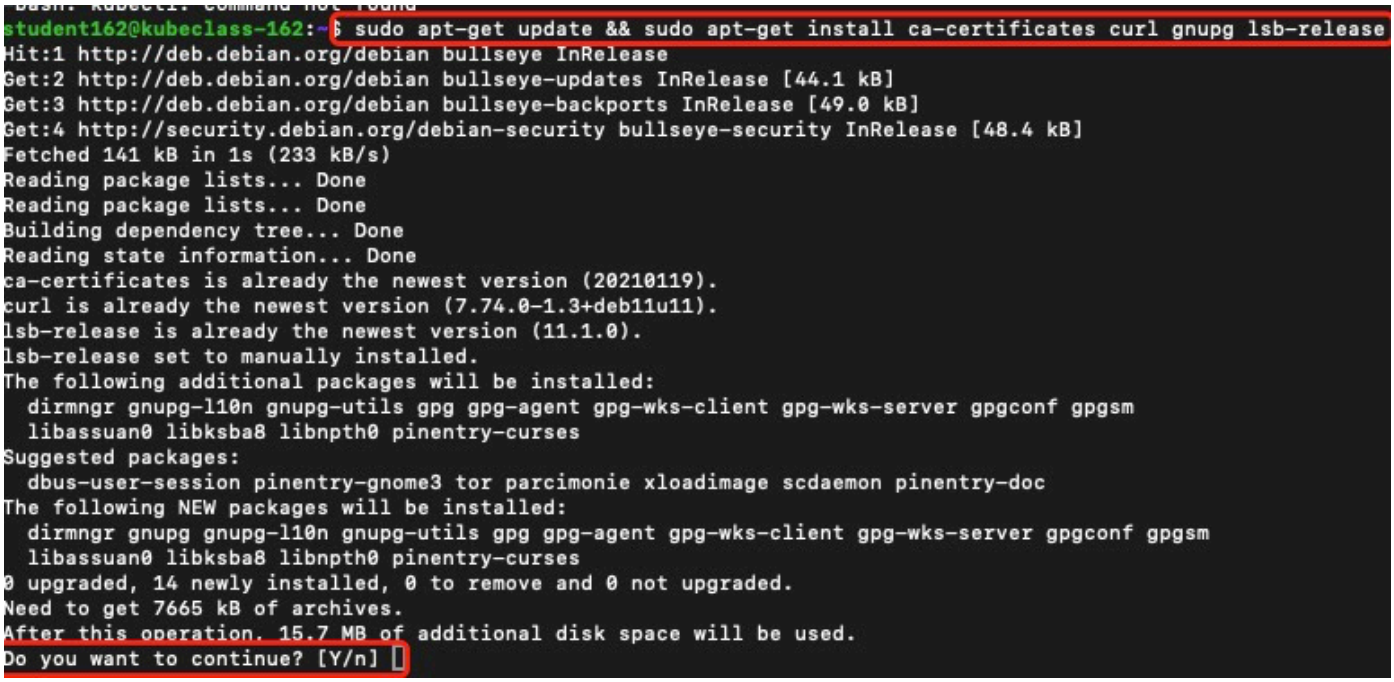



```
ssh student162@145.100.135.162 #然后输入密码
```

1. Installing Docker

```
sudo apt-get update && sudo apt-get install ca-certificates curl gnupg lsb-release
```

- apt-get是ubuntu和debian系统的包管理工具
- `sudo apt-get install`：安装指定的软件包。
- `ca-certificates`：包含了一系列的 CA（证书颁发机构）证书，这对于 HTTPS 连接是必需的，确保安全地下载软件。
- `curl`：一个命令行工具，用于发送和接收文件，常用于下载软件或检查网站是否可达。
- GNU Privacy Guard，一个加密软件，用于数字签名和加密通信。
- `lsb-release`：用于打印发行版特定信息的工具，这在安装特定于发行版的软件包时可能会用到。



输入Y，安装完毕。

> 在以后的命令中，直接在命令后加 -y 默认yes避免交互式确认

接下来执行一系列命令，目的是将Docker的软件仓库（repository）添加到Debian系统的apt包管理器中，这样可以从官方源安装Docker

- 创建密钥存储目录：

  这一步创建了一个新的目录 `/etc/apt/keyrings`，用于存放后续下载的 GPG 密钥，确保 `apt` 在从新添加的仓库安装软件包时可以验证其真实性。

```
sudo install -m 0755 -d /etc/apt/keyrings
```

- 下载并保存 Docker 的 GPG 密钥：

  使用 `curl` 命令从 Docker 的官方网站下载 GPG 公钥。选项 `-fsSL` 确保了静默下载并遵循重定向。将下载的 GPG 公钥通过 `gpg --dearmor` 处理，转换为适用于 `apt` 的格式，并保存到 `/etc/apt/keyrings/docker.gpg`。

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
```

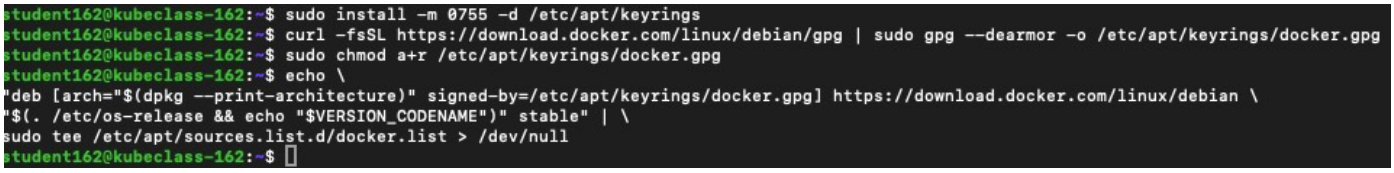
- 设置正确的文件权限：

  通过 `chmod a+r` 确保所有用户都有权限读取 `/etc/apt/keyrings/docker.gpg` 文件，这是 `apt` 在验证包签名时所必需的。

```
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

- 添加 Docker 仓库到 apt 源列表

```
echo \
"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```



完成以上步骤后即成功将 Docker 的官方仓库添加到了 Debian 系统的 `apt` 包管理器中，现在可以使用 `apt` 命令来安装 Docker 了。

- 更新软件包列表:

  确保了尝试安装软件包时，`apt` 能够获取到最新版本的信息和依赖关系。

```
sudo apt-get update
```

```
student162@kubeclass-162:~$ sudo apt-get update
Hit:1 http://security.debian.org/debian-security bullseye-security InRelease
Hit:2 http://deb.debian.org/debian bullseye InRelease
Hit:3 http://deb.debian.org/debian bullseye-updates InRelease
Hit:4 http://deb.debian.org/debian bullseye-backports InRelease
Get:5 https://download.docker.com/linux/debian bullseye InRelease [43.3 kB]
Get:6 https://download.docker.com/linux/debian bullseye/stable amd64 Packages [31.8 kB]
Fetched 75.1 kB in 1s (90.4 kB/s)
Reading package lists... Done
```

- 安装 Docker Engine 和相关工具

```
sudo apt-get -y install docker-ce docker-ce-cli containerd.io
```

- `sudo apt-get -y install`：以管理员权限安装指定的软件包。选项 `-y` 自动回答所有提示为 "是"，从而避免安装过程中的交互式确认。
- `docker-ce`：Docker 社区版（Community Edition）引擎，提供 Docker 运行时的核心功能。
- `docker-ce-cli`：Docker 社区版的命令行界面工具，允许用户与 Docker 引擎交互，执行如构建、运行 Docker 容器等操作。
- `containerd.io`：一个开源容器运行时，Docker Engine 依赖于它来管理容器的生命周期。

```
student162@kubeclass-162:~$ sudo apt-get update && sudo apt-get install ca-certificates curl gnupg lsb-release
Hit:1 http://deb.debian.org/debian bullseye InRelease
Get:2 http://deb.debian.org/debian bullseye-updates InRelease [44.1 kB]
Get:3 http://deb.debian.org/debian bullseye-backports InRelease [49.0 kB]
Get:4 http://security.debian.org/debian-security bullseye-security InRelease [48.4 kB]
Fetched 141 kB in 1s (233 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20210119).
curl is already the newest version (7.74.0-1.3+deb11u11).
lsb-release is already the newest version (11.1.0).
lsb-release set to manually installed.
The following additional packages will be installed:
  dirmngr gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client gpg-wks-server gpgconf gpgsm
  libassuan0 libksba8 libnpth0 pinentry-curses
Suggested packages:
```

成功安装了 Docker 引擎，接下来完成两个额外的步骤，以便更便捷地使用 Docker。

- 自动启动Docker引擎：

  当虚拟机（VM）启动时，希望 Docker 引擎自动启动，这样就不需要每次都手动启动 Docker。通过使用 `systemctl` 命令实现

```
sudo systemctl enable docker
```

- 允许无需sudo权限使用Docker：通过将用户添加到 `docker` 组，可以允许该用户无需 `sudo` 权限即可运行 Docker 命令

```
sudo usermod -aG docker "$USER"
```

将当前用户（`$USER` 环境变量代表当前登录的用户）添加到 `docker` 组中。选项 `-aG` 分别表示向组添加一个用户，并指定组名为 `docker`。

```
student162@kubeclass-162:~$ 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
student162@kubeclass-162:~$ sudo usermod -aG docker "$USER"
student162@kubeclass-162:~$
```

- 重新登录

为了使组成员资格的更改生效，注销当前会话并重新登录。通过执行 `exit` 命令退出当前的 SSH 会话，然后重新 SSH 连接到虚拟机。

```
exit
```

```
student162@kubeclass-162:~$ exit
logout
Connection to 145.100.135.162 closed.
```

- 验证权限

重新登录后，验证是否正确地被添加到了 `docker` 组。这个命令会列出当前用户所属的所有组。

```
groups
```

```
student162@kubeclass-162:~$ groups
student162 docker
```

- 在所有虚拟机上重复这个过程

  如果要在多台虚拟机上部署 Kubernetes 集群，就需要在所有想要添加到集群的虚拟机上重复上述过程。

2. Installing Kubernetes

- 在安装 Kubernetes 之前，需要确保系统安装了一些额外的依赖项：

```
sudo apt-get -y install apt-transport-https net-tools
```

- `apt-transport-https`：允许 `apt` 通过 HTTPS 协议从远程仓库下载包。这对于安全地从 Kubernetes 的仓库安装软件包是必需的。
- `net-tools`：提供网络相关的工具，例如 `ifconfig`。这在配置和诊断网络问题时可能会用到。

```
student162@kubeclass-162 ~$ sudo apt-get -y install apt-transport-https net-tools
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
net-tools is already the newest version (1.60+git20181103.0eebece-1).
net-tools set to manually installed.
The following NEW packages will be installed:
  apt-transport-https
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 160 kB of archives.
After this operation, 166 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bullseye/main amd64 apt-transport-https all 2.2.4 [160 kB]
Fetched 160 kB in 0s (5081 kB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 31817 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.2.4_all.deb ...
Unpacking apt-transport-https (2.2.4) ...
Setting up apt-transport-https (2.2.4) ...
student162@kubeclass-162:~$
```

- 配置Kubernetes仓库：

  与添加 Docker 仓库类似，安装 Kubernetes 软件也需要先添加 Kubernetes 的软件仓库。

  - 导入 GPG 密钥并添加仓库地址到 `apt` 源列表。这条命令使用 `curl` 下载 Kubernetes 仓库的 GPG 密钥，并保存到 `/usr/share/keyrings/kubernetes-archive-keyring.gpg`。这个密钥用于验证从 Kubernetes 仓库下载的包的真实性。

```
sudo curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

  - 将 Kubernetes 仓库添加到 `apt` 的源列表中

```
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

完成这些步骤后，系统就已经配置好了 Kubernetes 的软件仓库。接下来，运行 `sudo apt-get update` 来更新软件包索引，然后就可以安装 `kubeadm`、`kubelet` 和 `kubectl` 了。

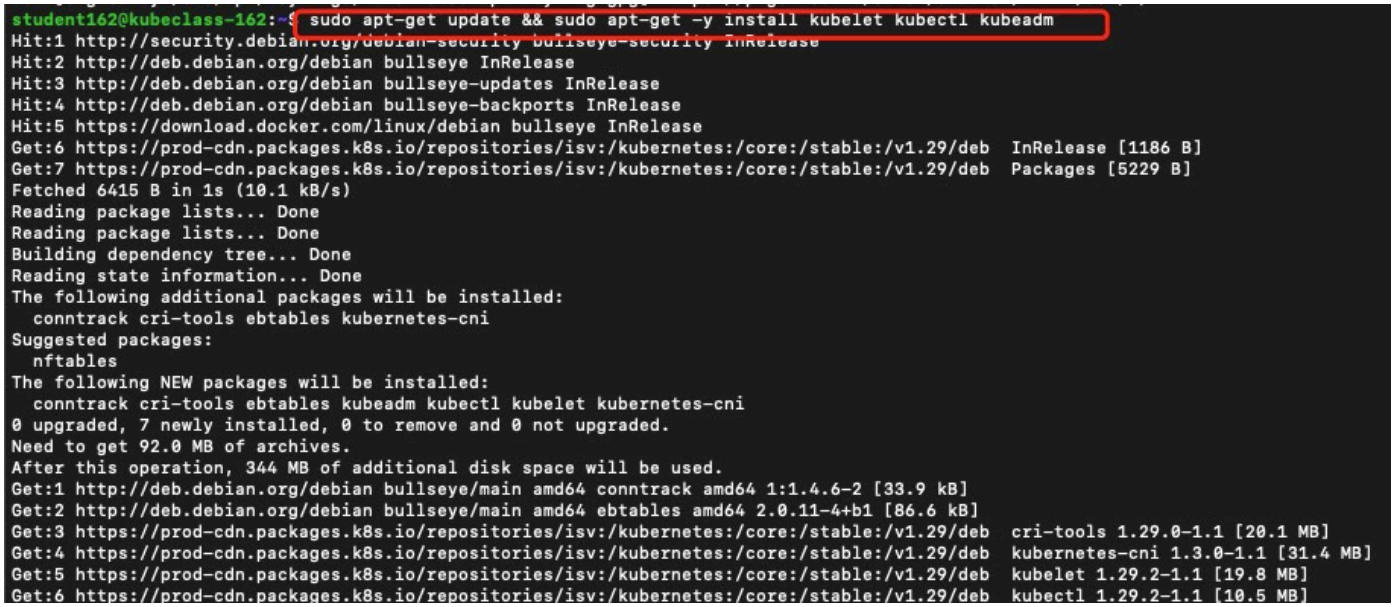
- 安装Kubernetes包

```
sudo apt-get update && sudo apt-get -y install kubelet kubectl kubeadm
```

要确保 `apt` 的软件包列表是最新的，这样当你尝试安装 `kubelet`、`kubectl` 和 `kubeadm` 时，`apt` 能找到这些包。没有运行这个命令可能会遇到找不到包的错误。

安装 Kubernetes 集群管理所需的三个核心组件。

- **`kubelet`**：在每个集群节点上运行的守护进程，负责启动和管理容器和镜像。
- **`kubectl`**：用于与集群控制平面交互的命令行工具，可以用来管理 Pods、服务等。
- **`kubeadm`**：帮助自动化集群设置的工具，通过自动生成所需的配置文件来简化集群的初始化和管理。



- 固定 Kubernetes 包版本

```
sudo apt-mark hold kubelet kubectl kubeadm
```

这个命令会将 `kubelet`、`kubectl` 和 `kubeadm` 标记为 "hold" 状态，这意味着即使有新版本可用，这些包也不会被自动更新。这是因为 Kubernetes 集群的更新通常需要仔细规划和执行，集群更新可能涉及节点间的依赖和配置变更，错误的更新可能会导致集群不稳定或服务中断。因此，推荐使用 Kubernetes 提供的工具手动管理集群的版本更新。

- 在所有虚拟机上重复这个过程

和 Docker 一样，如果想将某个虚拟机（VM）作为集群的一部分，必须在每个集群节点上安装这些 Kubernetes 包。这样，每个节点都能作为集群的一部分参与工作负载的运行和管理。

3. 配置Control Node

在设置 Kubernetes 集群时，通常将节点分为两类：控制节点（构成control plane的一部分）和工作节点。

这一节将关注于配置控制节点。通过执行一系列命令，`kubeadm` 工具能够初始化集群的controle plane。

- 初始化kubeadm

  ```
  IP=$(ip -4 -o a | grep -i "ens3" | cut -d ' ' -f 2,7 | cut -d '/' -f 1 | awk '{print $2}') sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --control-plane-endpoint=$IP --apiserver-cert-extra-sans=$IP
  ```

  分解命令：

  - 第一部分：获取当前节点上名为 `ens3` 网络接口的 IPv4 地址，并将这个地址赋值给变量 `IP`。在第二部分中用于指定控制平面节点的 IP 地址。

- `ip -4 -o a`：这部分的命令使用 `ip` 工具列出所有活动的网络接口及其地址，只针对 IPv4 地址（`-4` 选项），并以一行一个地址的格式输出（`-o` 选项）。
- `grep -i "ens3"`：使用 `grep` 命令从 `ip` 命令的输出中筛选出包含 “ens3” 的行，`-i` 选项使搜索不区分大小写。
- `cut -d ' ' -f 2,7`：这个 `cut` 命令用于从每行中提取出网络接口名称和其对应的 IP 地址。它通过指定分隔符为空格（`-d ' '`）并选择第2和第7个字段（`-f 2,7`）。
- `cut -d '/' -f 1`：第二个 `cut` 命令进一步处理每行，以 `/` 为分隔符，只取第1个字段，目的是从 IP 地址/子网掩码格式中只保留 IP 地址部分。
- `awk '{print $2}'`：最后，使用 `awk` 从剩下的输出中提取出 IP 地址。由于前面的 `cut` 命令已经将输出限制为网络接口和 IP 地址，这里 `awk` 命令实际上是从这两个字段中选择 IP 地址（即第2个字段）。
- 第二部分：使用第一部分提取的 IP 地址，初始化 Kubernetes 控制平面
- `--pod-network-cidr=192.168.0.0/16` 指定 Pod 网络使用的 IP 地址范围。
- `--control-plane-endpoint=$IP` 设置控制平面的endpoint为当前节点的 IP 地址。
- `--apiserver-cert-extra-sans=$IP` 确保 API 服务器证书包含此 IP 地址，允许通过它安全地访问 API 服务器。

⚠️⚠️⚠️⚠️⚠️实践中发现直接执行这个命令会报错：[ERROR CRI]: container runtime is not running⚠️⚠️⚠️⚠️⚠️

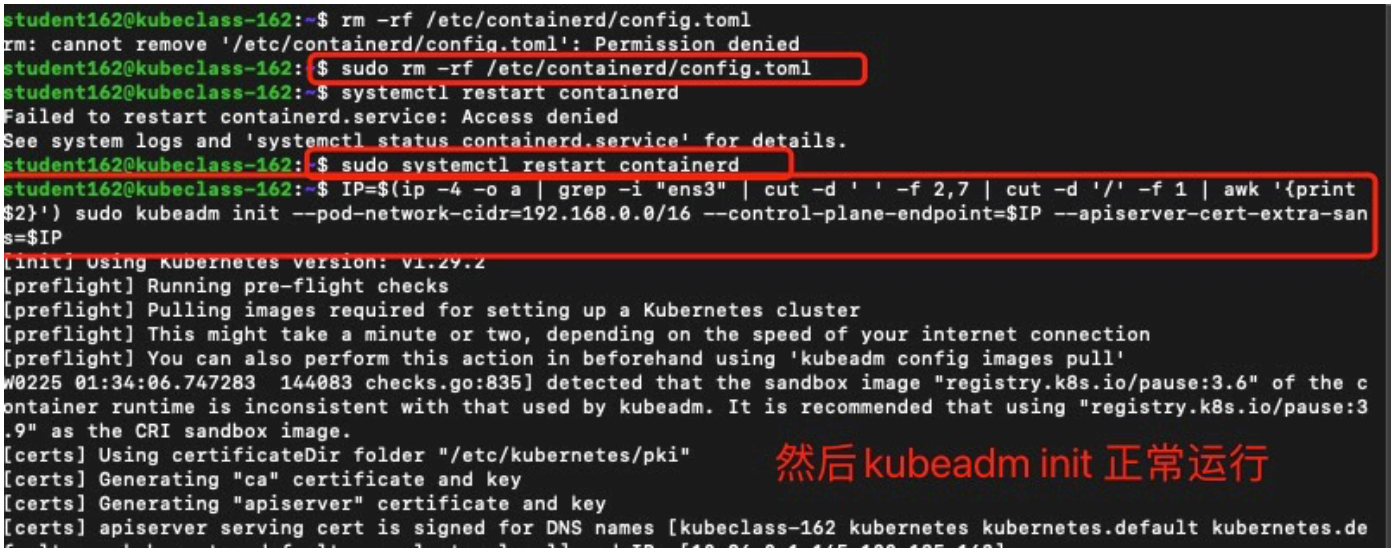
解决方法：参考了 https://blog.csdn.net/qq_43580215/article/details/125153959

运行之前先运行以下两个命令：

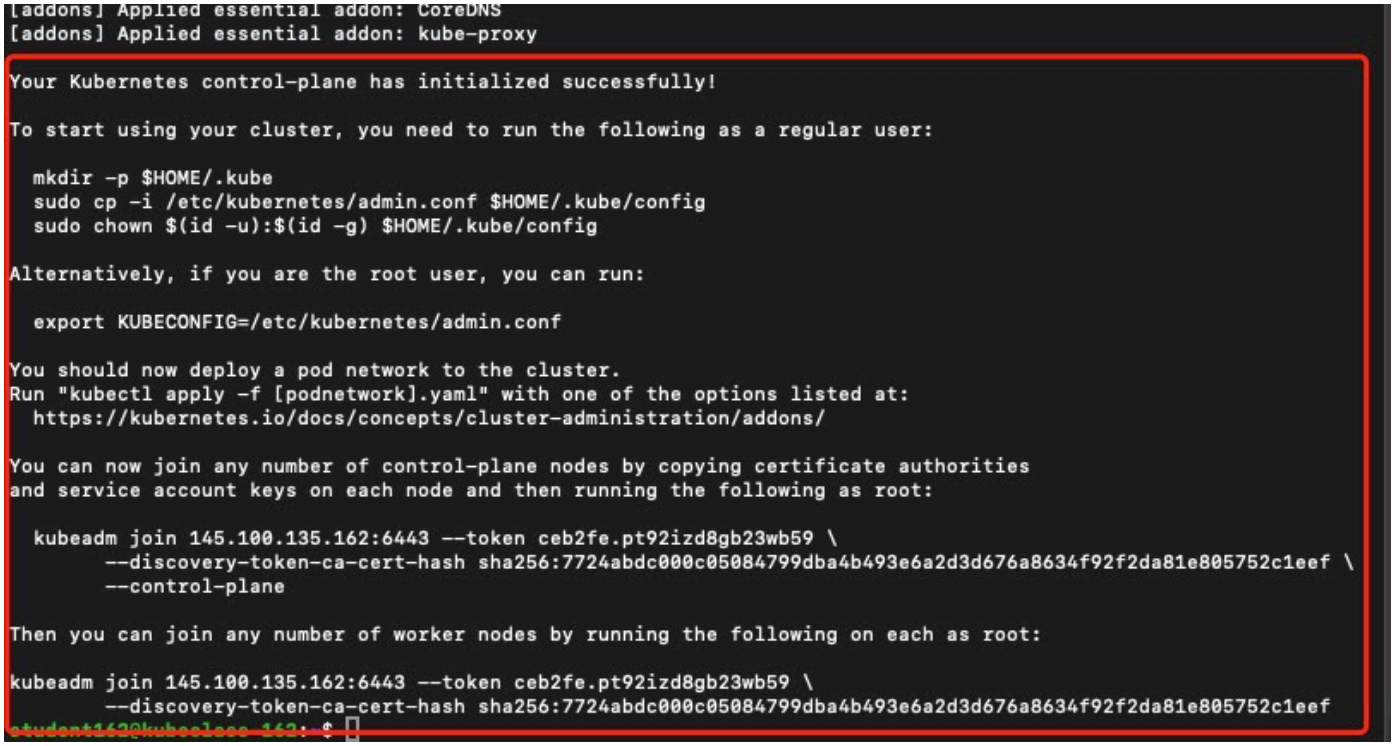
```
rm -rf /etc/containerd/config.toml
```

```
systemctl restart containerd
```

然后再次执行即可正常运行：



执行完毕后，`kubeadm` 会输出一些信息，告诉我们如何将更多的控制平面节点或工作节点加入到集群中：




```
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/

You can now join any number of control-plane nodes by copying certificate authorities
and service account keys on each node and then running the following as root:

  kubeadm join 145.100.135.162:6443 --token ceb2fe.pt92izd8gb23wb59 \
  --discovery-token-ca-cert-hash
sha256:7724abdc000c05084799dba4b493e6a2d3d676a8634f92f2da81e805752c1eef \
  --control-plane

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

kubeadm join 145.100.135.162:6443 --token ceb2fe.pt92izd8gb23wb59 \
  --discovery-token-ca-cert-hash
sha256:7724abdc000c05084799dba4b493e6a2d3d676a8634f92f2da81e805752c1eef
```

这些信息包括两个重要的 `kubeadm join` 命令：

- 加入控制平面节点

第一个 `kubeadm join` 命令是用于将额外的控制平面节点加入到集群中。由于在我们作业的情况下，只有一个控制平面节点（即我们目前配置的节点），因此我们不需要使用这个命令。

- 加入工作节点

第二个 `kubeadm join` 命令是用于将工作节点加入到集群中。这个命令包含了集群的 API 服务器地址（`<IP>:6443`），一个用于节点加入时身份验证的令牌（`<token>`），以及用于发现集群并验证 API 服务器证书的 CA 证书哈希值（`<hash>`）。

这个命令复制并保存下来，因为每个想要加入到这个集群的工作节点都需要执行它。(前面要加sudo)

```
sudo kubeadm join 145.100.135.162:6443 --token ceb2fe.pt92izd8gb23wb59 \
  --discovery-token-ca-cert-hash
sha256:7724abdc000c05084799dba4b493e6a2d3d676a8634f92f2da81e805752c1eef
```

- 划重点：

  初始化 Kubernetes 集群后，`kubeadm` 提供的 `join` 命令是用于将新的节点加入到集群中，扩展集群的工作节点。

  在我们的作业中，由于只有一个控制平面节点，只需关注如何将工作节点加入集群。

在向集群添加工作节点之前，需要在控制节点上再进行一些配置，以确保 `kubelet` 服务以正确的参数运行。这些参数对于集群的网络和节点通信至关重要。

- 配置kubelet参数

```
sudo echo KUBELET_KUBEADM_ARGS=\"--network-plugin=cni --pod-infra-container-
image=k8s.gcr.io/pause:3.2 --node-ip=$(ip -4 -o a | grep -i "ens3" | cut -d ' ' -f 2,7 |
cut -d '/' -f 1 | awk '{print $2}')\" | sudo tee /var/lib/kubelet/kubeadm-flags.env
```

使用 `ip` 命令获取当前节点的 IP 地址，这与前面获取控制平面 IP 地址的方法相同。设置`KUBELET_KUBEADM_ARGS`环境变量，其中包含了 `kubelet`启动时需要的一些关键参数：

`--network-plugin=cni`：指定使用 CNI (Container Network Interface) 作为网络插件。CNI 允许 Kubernetes 通过各种网络插件配置 Pod 网络。

`--pod-infra-container-image=k8s.gcr.io/pause:3.2`：指定 pod 基础设施容器（即 pause 容器）的镜像。pause 容器作为每个 Pod 的基础，用于承载 Pod 内所有容器的网络命名空间。

`--node-ip=...`：设置节点的 IP 地址，确保节点间通信使用正确的网络接口。

使用 `tee` 命令将这些参数写入 `/var/lib/kubelet/kubeadm-flags.env` 文件中，这个文件由 `kubelet` 读取，以获取启动时应使用的参数。

- 重启kubelet服务

```
sudo systemctl restart kubelet.service
```

通过重启 `kubelet` 服务，确保它使用了我们上一步在 `/var/lib/kubelet/kubeadm-flags.env` 文件中设置的新参数。

完成这些配置后，控制节点就准备好了用于将其他节点加入集群。

```
student162@kubeclass-162:~$ sudo echo KUBELET_KUBEADM_ARGS=\"--network-plugin=cni --pod-infra-container-image=k8s.gcr.io/pause:3.2 --node-ip=$(ip -4 -o a | grep -i "ens3" | cut -d ' ' -f 2,7 | cut -d '/' -f 1 | awk '{print $2}')\" | sudo tee /var/lib/kubelet/kubeadm-flags.env
KUBELET_KUBEADM_ARGS="--network-plugin=cni --pod-infra-container-image=k8s.gcr.io/pause:3.2 --node-ip=145.100.135.162"
student162@kubeclass-162:~$ sudo systemctl restart kubelet.service
student162@kubeclass-162:~$
```

4. 配置Worker Node

首先从上一节的控制节点logout（执行`exit`），然后登录到想要添加的工作节点。为了将工作节点添加到 Kubernetes 集群中，以下步骤要在每一个工作节点上重复。

- 登出再登录

```
student162@kubeclass-162:~$ exit
logout
Connection to 145.100.135.162 closed.
(base) rr@cuicuishavongshideMacBook-Pro ~/Library/CloudStorage/OneDrive-Personal/WSCB/wscb <master●>
$ ssh student162@145.100.135.162
student162@145.100.135.162's password:
Linux kubeclass-162 5.10.0-27-amd64 #1 SMP Debian 5.10.205-2 (2023-12-31) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Feb 25 00:19:03 2024 from 80.113.224.196
```

- 使用kubeadm加入集群

```
sudo kubeadm join 145.100.135.162:6443 --token ceb2fe.pt92izd8gb23wb59 \
  --discovery-token-ca-cert-hash
sha256:7724abdc000c05084799dba4b493e6a2d3d676a8634f92f2da81e805752c1eef
```

⚠️⚠️⚠️⚠️⚠️实践中发现直接执行这个命令会报错：

[ERROR FileAvailable--etc-kubernetes-kubelet.conf]: /etc/kubernetes/kubelet.conf already exists

[ERROR FileAvailable--etc-kubernetes-pki-ca.crt]: /etc/kubernetes/pki/ca.crt already exists

```
Last login: Sun Feb 25 00:19:03 2024 from 80.113.224.196
student162@kubeclass-162:~$ sudo kubeadm join 145.100.135.162:6443 --token ceb2fe.pt92izd8gb23wb59 \
        --discovery-token-ca-cert-hash sha256:7724abdc000c05084799dba4b493e6a2d3d676a8634f92f2da81e805752c1eef
[preflight] Running pre-flight checks
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR FileAvailable--etc-kubernetes-kubelet.conf]: /etc/kubernetes/kubelet.conf already exists
        [ERROR FileAvailable--etc-kubernetes-pki-ca.crt]: /etc/kubernetes/pki/ca.crt already exists
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher
```

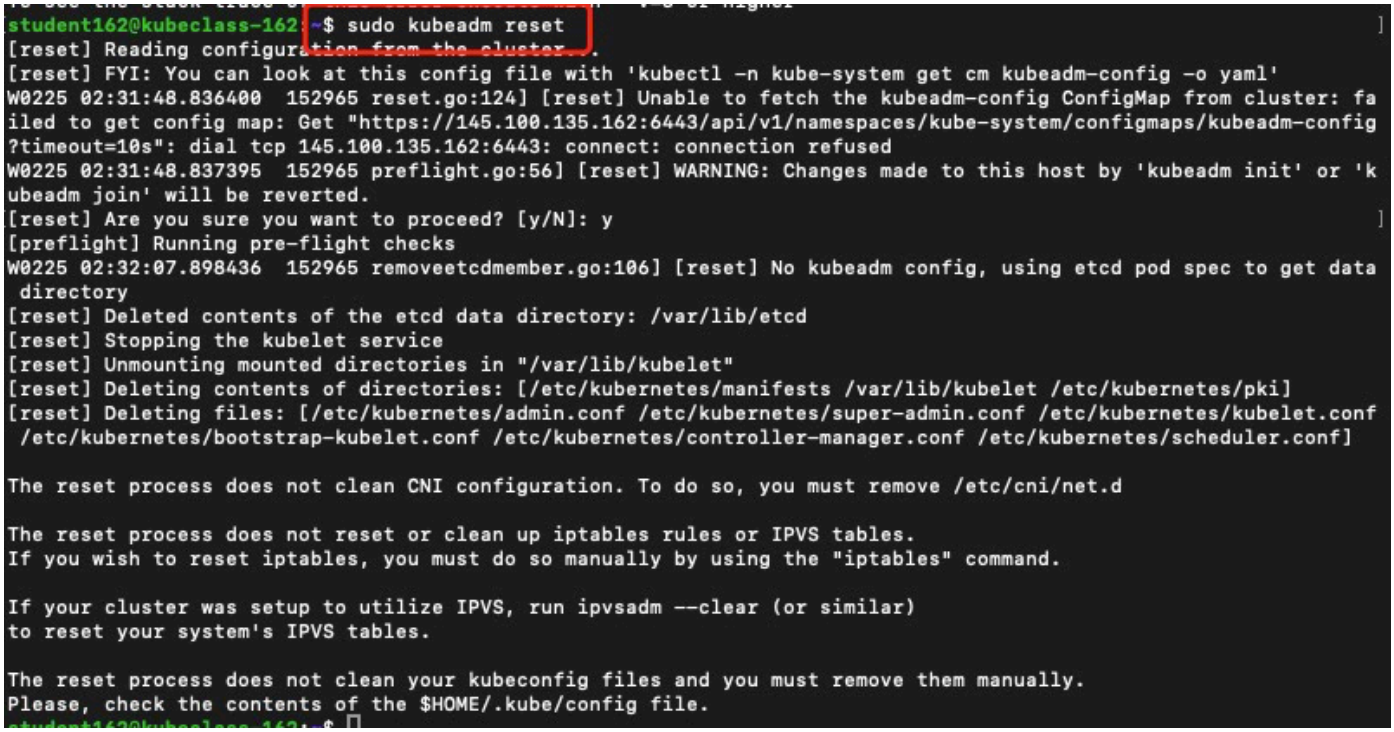
😭😭😭😭😭😭接下来是踩坑记录：

有些解决方法说，

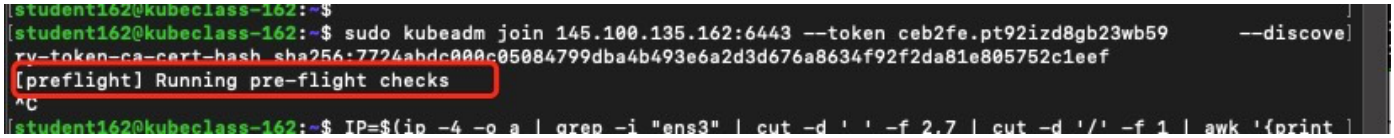
> 当使用 `sudo kubeadm join` 命令将节点加入到 Kubernetes 集群时，遇到的错误提示 `/etc/kubernetes/kubelet.conf` 和 `/etc/kubernetes/pki/ca.crt` 文件已存在，这表明该节点之前可能已经被配置为集群的一部分，或者 `kubeadm init` 或 `kubeadm join` 命令在此节点上之前运行过，但没有完全清理。为了解决这个问题，重置 `kubeadm` 的配置来清理这些遗留文件，然后再次尝试加入集群。

所以我执行了

```
sudo kubeadm reset
```



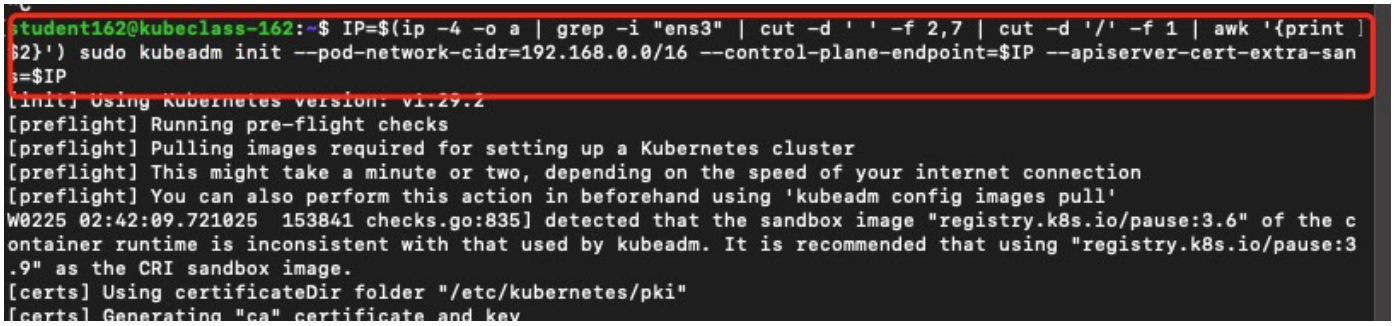
再次重新执行kubeadm join时，发现卡在[preflight] Running pre-flight checks



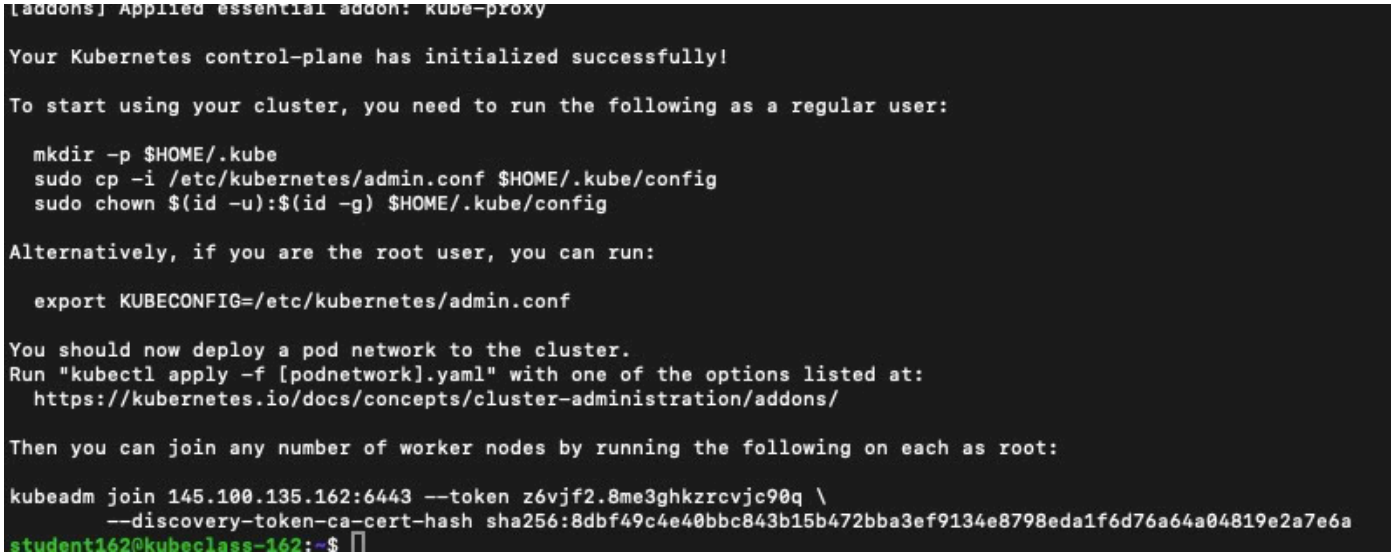
查了下卡住的原因，参考了https://blog.csdn.net/weixin_40668374/article/details/123641294 （k8s node节点加入到集群时卡住 “[preflight] Running pre-flight checks”），原因可能是” token过期，重新生成一下“ 。

猜测可能我上一步`sudo kubeadm reset`，实际上导致删除了control node的token，导致worker node现在无法加入集群。

因此回到《3.配置Control Node》中，重新做了一次kubeadm init初始化的步骤：



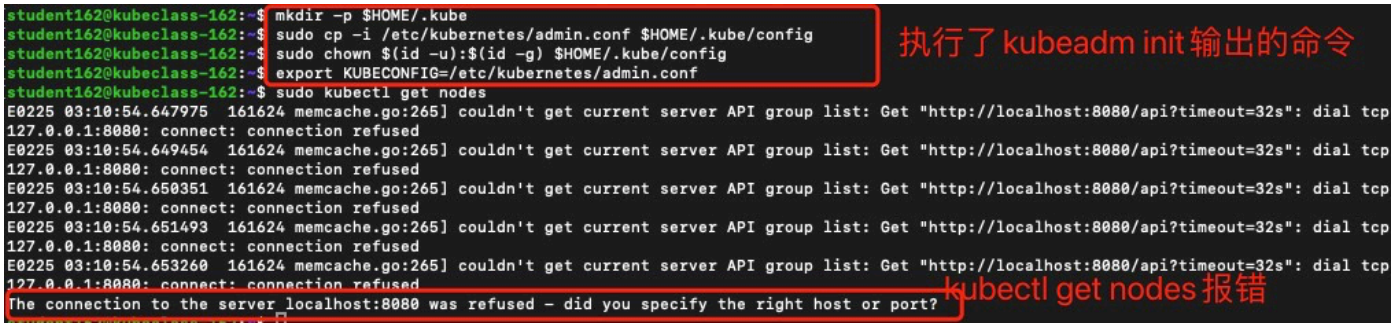
执行完毕后如之前一样，`kubeadm` 会输出一些信息，但区别是这次只告诉我们如何将更多的工作节点加入到集群中，而没有如何加入控制平面节点：




```
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 145.100.135.162:6443 --token z6vjf2.8me3ghkzrcvjc90q \
  --discovery-token-ca-cert-hash
sha256:8dbf49c4e40bbc843b15b472bba3ef9134e8798eda1f6d76a64a04819e2a7e6a
```

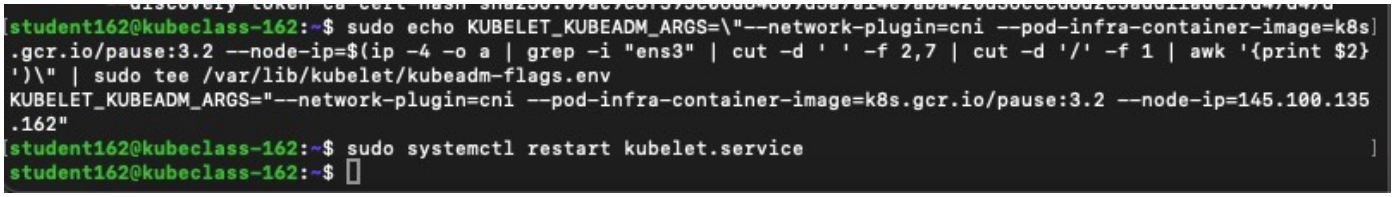
可以看到重新生成了token（z6vjf2.8me3ghkzrcvjc90q ）。

然后执行了上面输出信息中提到的所有命令：



发现运行kubectl get nodes看不到节点列表，报错：connection to the server localhost:8080 was refused – did you specify the right host or port?

理论上，然后再重复上一节的配置kubelet参数、重启kubelet服务，然后就完成了control node的配置。



然后，从控制节点logout（执行`exit`），然后登录到想要添加的工作节点。

下一步是配置工作节点。另一个问题是：不知道要怎么登录到想要的添加为工作节点的虚拟机啊？现在好像没法登，节点列表都还看不到。。。