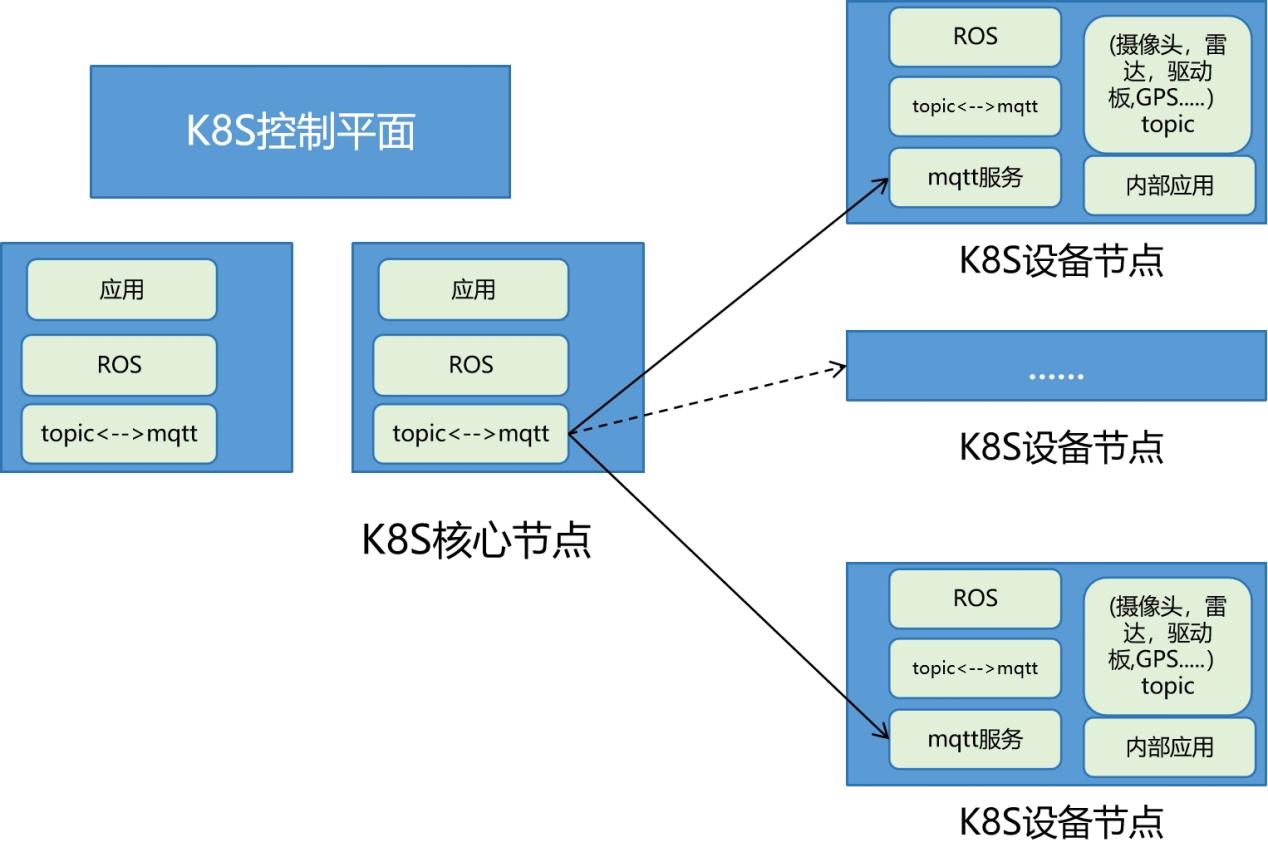
# **机器人远程操控环境搭建**

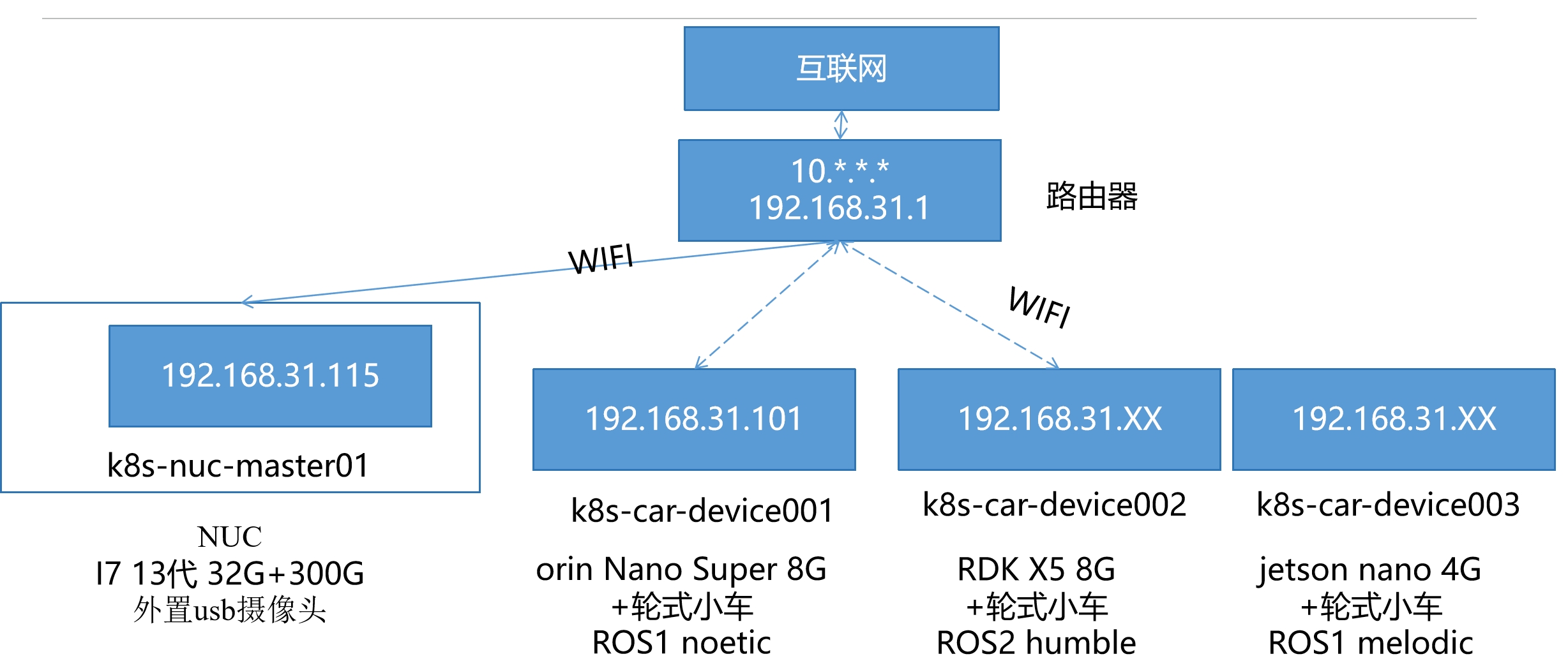
# **整体架构**



设计目标：

1. 设备节点独立工作，对外通过mqtt服务将自己需要暴露的服务提供出去，不需要暴露的topic流量在内部闭环。从核心节点看，所有的设备都对外提供mqtt服务。
2. 远程应用按需连接，需要使用时才增加流量。如果完成不访问，设备侧跟核心节点之间没有任何topic的流量。
3. 核心节点开发使用和本地一致的ROS环境，工具和使用方式一致。
4. 可以调整topic频率适应设备和核心节点的通信能力（如地图，位置，视频等），比如想得到的画面一秒一帧。

# **组网环境**



当前涵盖的几种主要配置和ROS版本。

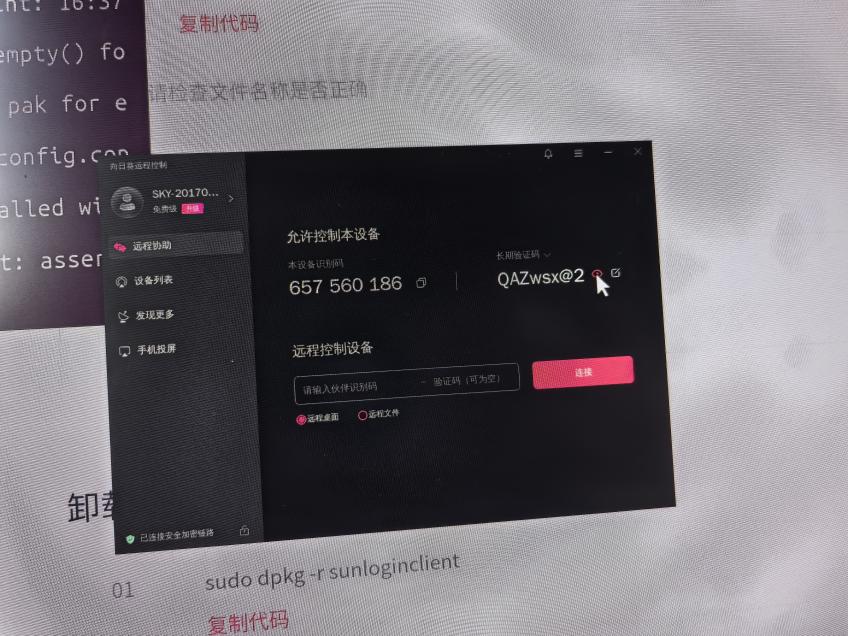
# **访问方式**

nuc远程向日葵

657 560 186

长期密码：QAZwsx@2

如果重启，可能会更换密码，并且无法访问，需要联系我重置该密码



k8s-nuc-master01

192.168.31.115

用户名：nesc

密码：QAZwsx@2134

k8s-car-device001

192.168.31.101

用户名：wheeltec

密码:dongguan

其他待补充，当前wifi都有问题。

k8s-car-device002

192.168.31.X

用户名：wheeltec

密码:dongguan

k8s-car-device003

192.168.31.X

用户名：wheeltec

密码:dongguan

# **Shifu安装**

sudo kubectl apply -f <https://gitee.com/edgenesis/shifu/raw/v0.65.0/pkg/k8s/crd/install/shifu_install.yml>

参考:

<https://shifu.dev/zh-Hans/docs/guides/install/install-shifu-prod>

# **构建本地镜像库**

sudo docker run -d -p 5000:5000 -v /opt/data/registry:/var/lib/registry docker.io/registry

查看镜像库的镜像信息

curl -XGET [http://192.168.31.115:5000/v2/\_catalog](http://192.168.31.150:5000/v2/_catalog)

# **构建本地镜像**

都在设备侧进行，确保镜像都是arm版本的。

1.重新打包mqtt服务器镜像

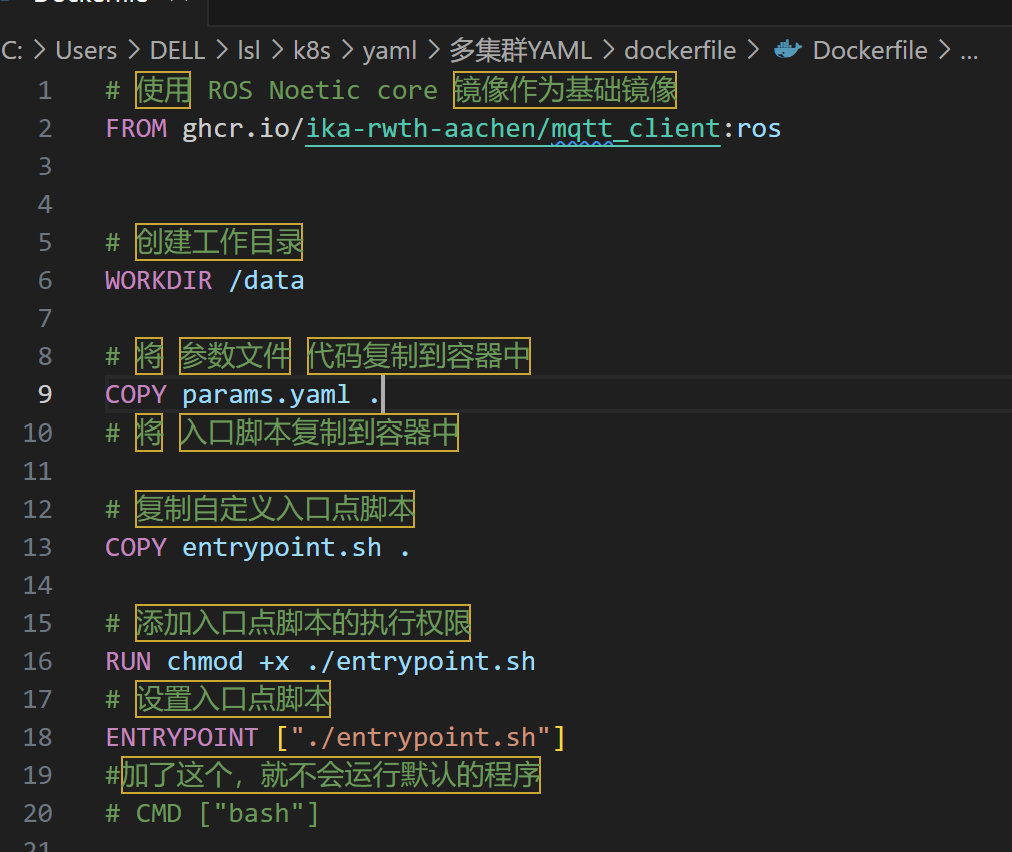
sudo docker tag eclipse-mosquitto:2.0.14 192.168.31.115:5000/arm/eclipse-mosquitto

sudo docker push 192.168.31.115:5000/arm/eclipse-mosquitto

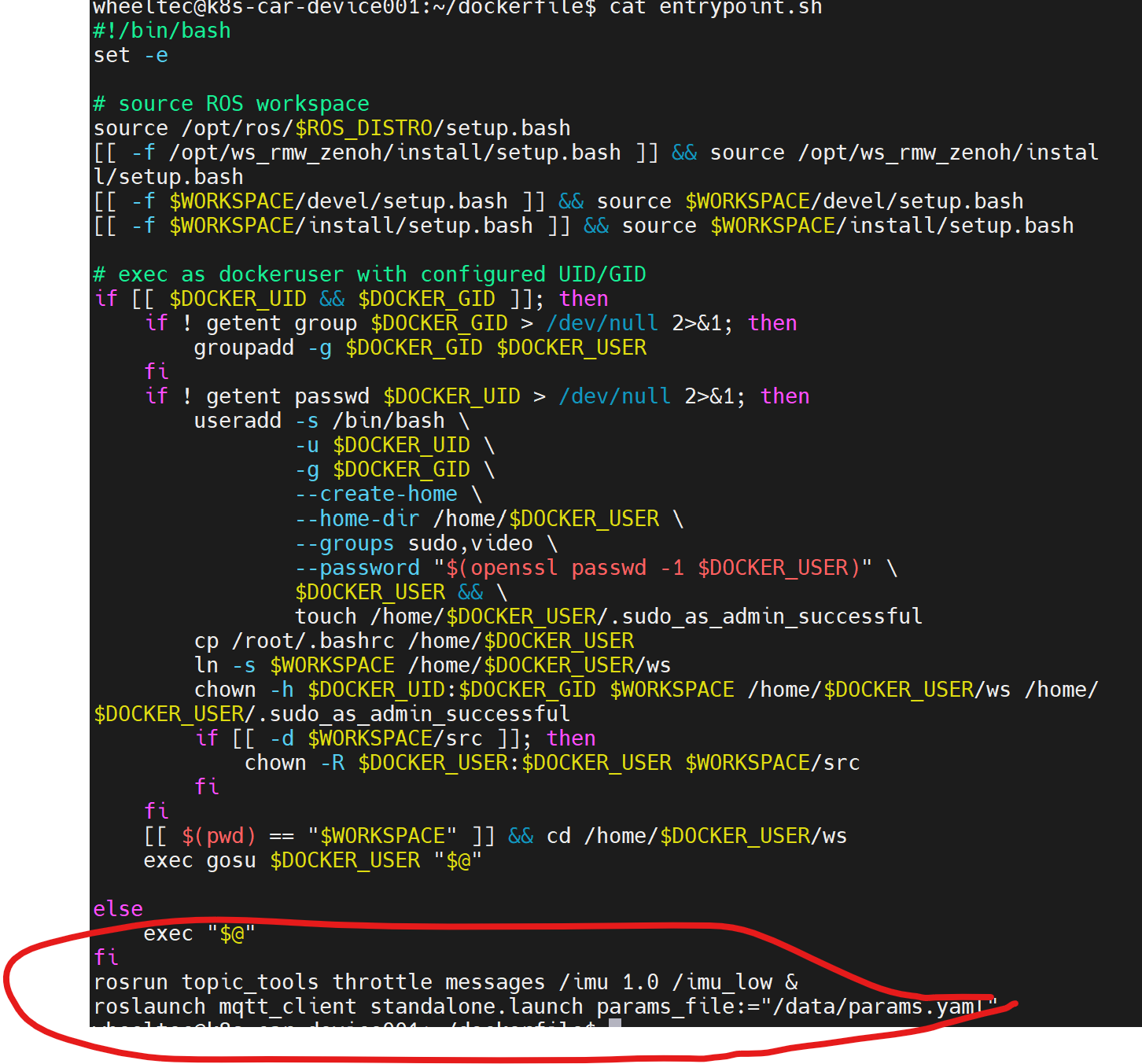
2.ROS转MQTT镜像

将dockerfile.zip解压里面的文件copy到设备中。

Dockerfile很简单，使用的是ghcr.io/ika-rwth-aachen/mqtt\_client:ros作为基础镜像，这个镜像有基础的mqtt\_client的环境。

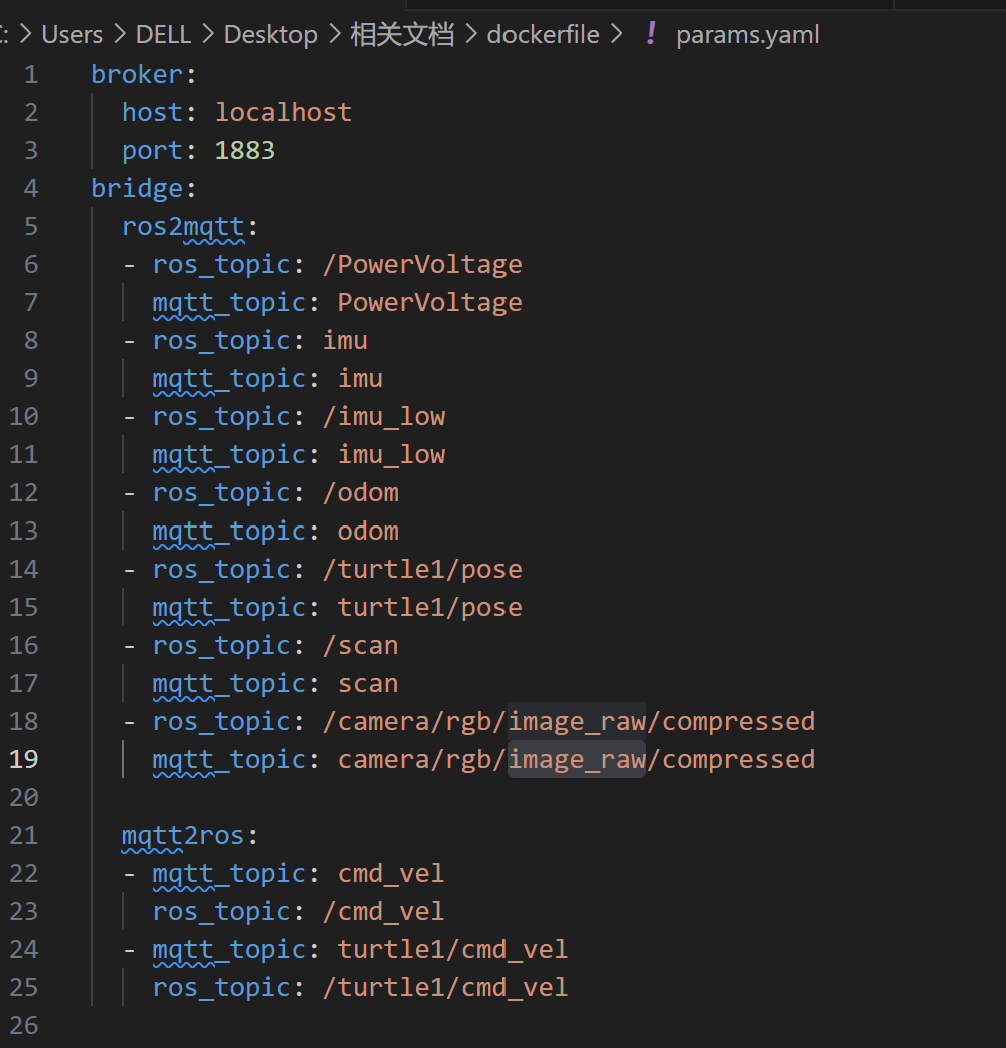


入口脚本：画红色的部分之上，是原始的入口文件的内容copy下来的，红色部分第一个命令是将imu的频率降低到1hz，发布为/imu\_low。第二个命令是使用mqtt\_client进行topic和mqtt的转换。



参数文件表达的含义是：

Mqtt服务器是本机，端口1883，需要从本机转出去的topic和从其他设备转过来的topic，这里面有小车的电压，里程计，惯性测量信息。以及/imu\_low，这个是将/imu频率降低到1Hz的topic。另外/turtle1/pose是小乌龟demo程序的topic。/camera/rgb/image\_raw/compressed是深度摄像机的图像topic。需要转回来的是小车的控制命令和小乌龟的控制命令。



生成本地镜像，注意到对应的本地镜像地址和端口需要放在前面

sudo docker build -f Dockerfile -t 192.168.31.115:5000/arm/nesc/ros-to-mqtt:1.0 .

生成之后，将镜像上传到本地镜像库

sudo docker push 192.168.31.115:5000/arm/nesc/ros-to-mqtt:1.0

# **远程操控部署**

先将两个设备打上标签

kubectl label nodes k8s-car-device001 cpu=arm

考虑到在设备上部署，相关的服务只能部署一个，所以在部署的时候使用的是kind: StatefulSet模式，确保一个node上只部署了一个相应的应用。

给每个设备部署一个mqtt服务器，使用

sudo kubectl apply -f local\_mosquitto\_device\_arm.yaml

查看部署情况

kubectl get pods -owide -n devices

确保镜像是部署到不同的device设备上

部署ROS转mqtt

kubectl apply -f local\_ros-mqtt.yaml

查看信息

kubectl get pods -owide -n devices

这个时候设备侧应该对应的服务都部署好了，登陆上执行一下启动小车的launch，

启动小车，包含运动控制和里程计等

roslaunch turn\_on\_wheeltec\_robot turn\_on\_wheeltec\_robot.launch

启动雷达

roslaunch turn\_on\_wheeltec\_robot wheeltec\_lidar.launch

启动摄像头

roslaunch turn\_on\_wheeltec\_robot wheeltec\_camera.launch

# **本地应用开发**

环境有限，在nuc上进行远程开发，其实只要能连接到集群网络即可。

打开摄像头：

cheese

确保小车在可见范围内移动，考虑到要长时间工作，当前小车大部分时间是连接充电线，不能让小车运动太远。

1. 本地启动roscore，使得本地具备基本环境

roscore

1. 启动ROS和MQTT转换

roslaunch mqtt\_client standalone.launch params\_file:="params\_master-192.168.31.101.yaml" &

1. 查看topic 情况

rostopic list

rostoic echo /imu

rostopic echo /imu\_low

查看两个话题的频率

rostoic hz /imu

rostopic hz /imu\_low

1. 控制远程小车

把catkin\_ws里面的wheeltec\_robot\_rc复制过去，

cd ~/catkin\_ws/src/wheeltec\_robot\_rc/scripts

确保可执行

chmod +x turtlebot\_teleop\_key.py

cd ~/catkin\_ws

编译，

catkin\_make

source devel/setup.bash

调用

roslaunch wheeltec\_robot\_rc keyboard\_teleop.launch

可以在键盘下控制远程小车了，请确保在视线范围内移动。

1. 查看远程图像

rqt\_image\_view

找到对应的topic，就可以查看深度图像了。

# **当前遗留问题：**

1. 设备节点如果重启，不能够正常的加入集群，出现not ready状态,虚拟机节点则不会这样，待解决。

# **待进一步开发**

1. 按照shifu的接口标准进一步简化整个步骤
2. ROS2接口对接和融合