

1. UTM 가상머신을 사용하게 된 이유 + MicroROS 연결방법

저의 경우 기존에 인텔맥 노트북을 사용해오고 있었습니다. 평소에 맥의 부트캠프 기능을 통해 윈도우 <-> 맥간 듀얼 부팅을 세팅해두고 사용했었습니다. 그런데 ROS의 경우 기본적으로 Ubuntu 환경 위에서 돌아가기에... 이게 참 곤란했습니다.

듀얼 부팅으로 Ubuntu까지 설치하기엔 이미 윈도우 <-> 맥 듀얼 부팅까지 설정해 둔 상태인지라 노트북이 망가질까봐 엄두를 내지 못했습니다. 그래서 처음에는 윈도우 위에서 WSL2를 이용하여 Ubuntu 가상머신을 만들어 사용해 보자고 생각했었습니다. 그런데 Yahboom사에서 기본적으로 VMWare 이미지를 제공해주고 있었기에 분명히 여기에 내가 설치하지 못한 추가적인 파일들이 더 있을 것이라 생각하고 VMWare을 설치하여 이미지를 올렸습니다. 이후 Yahboom 자동차에 있는 ESP32에 동작하는 microROS와 통신하기 위해 아래 링크의 과정을 가상 머신 위에서 따라했었습니다.

<https://www.notion.so/knlabs/3-1a1c3d4d5f458192a200f157eada7598?pvs=4>

그런데 위 과정을 따라해보면서 치명적인 문제가 발생했습니다.

```
yahboom@yahboom-VM: ~/yahboomcar_ws
[Processing: robot_localization]
[Processing: robot_localization]
Finished <<< robot_localization [7min 1s]

Summary: 15 packages finished [7min 2s]
yahboom@yahboom-VM:~/yahboomcar_ws$ shs
humble/setup.bash loaded
yahboom@yahboom-VM:~/yahboomcar_ws$ lsb
install/local_setup.bash loaded
yahboom@yahboom-VM:~/yahboomcar_ws$ ros2 run micro_ros_agent micro_ros_age
nt udp4 --port 8090 -v4
[1742444658.246534] info | UDPv4AgentLinux.cpp | init
| running... | port: 8090
[1742444658.247399] info | Root.cpp | set_verbose_level
| logger setup | verbose_level: 4
^C[ros2run]: Interrupt
yahboom@yahboom-VM:~/yahboomcar_ws$ ros2 run micro_ros_agent micro_ros_age
nt udp4 --port 8090 -v4
[1742444719.028716] info | UDPv4AgentLinux.cpp | init
| running... | port: 8090
[1742444719.029453] info | Root.cpp | set_verbose_level
| logger setup | verbose_level: 4

yahboom@yahboom-VM: ~/monica24/yahboom_car 151x23
if __name__ == '__main__':
    robot = MicroROS_Robot(port='/dev/ttyUSB0', debug=False)
    print("Rebooting Device, Please wait.")
    robot.reboot_device()

    robot.set_wifi_config([REDACTED])
    robot.set_udp_config([192, 168, 179, 128], 8090)
    robot.set_car_type(robot.CAR_TYPE_COMPUTER)
    # robot.set_car_type(robot.CAR_TYPE_RPIS)
    robot.set_ros_domain_id(55)
    robot.set_ros_serial_baudrate(921600)
    robot.set_ros_namespace("")
    robot.set_pwm_servo_offset(1, 0)
    robot.set_pwm_servo_offset(2, 0)
    robot.set_motor_pid_parm(1, 0.2, 0.2)
    robot.set_lmu_yaw_pid_parm(1, 0, 0.2)

    time.sleep(.1)
    robot.print_all_firmware_parm()
    print("Please reboot the device to take effect, if you change some device config.")

    try:
        "config_robot.py" 523L, 18251B
504,30 98%
```

/dev/ttyUSB0번으로 포트도 잡아주고, Host PC의 와이파이도 그대로 넣어주고, 우분투 터미널 내에서 ifconfig를 실행하여 나온 IP도 잘 넣어주고, ROS_DOMAIN_ID까지 맞춰 주었는데 microROS agent와 ESP32의 microROS가 서로 통신을 하지 못하는 문제가 지속적으로 발생하였습니다.

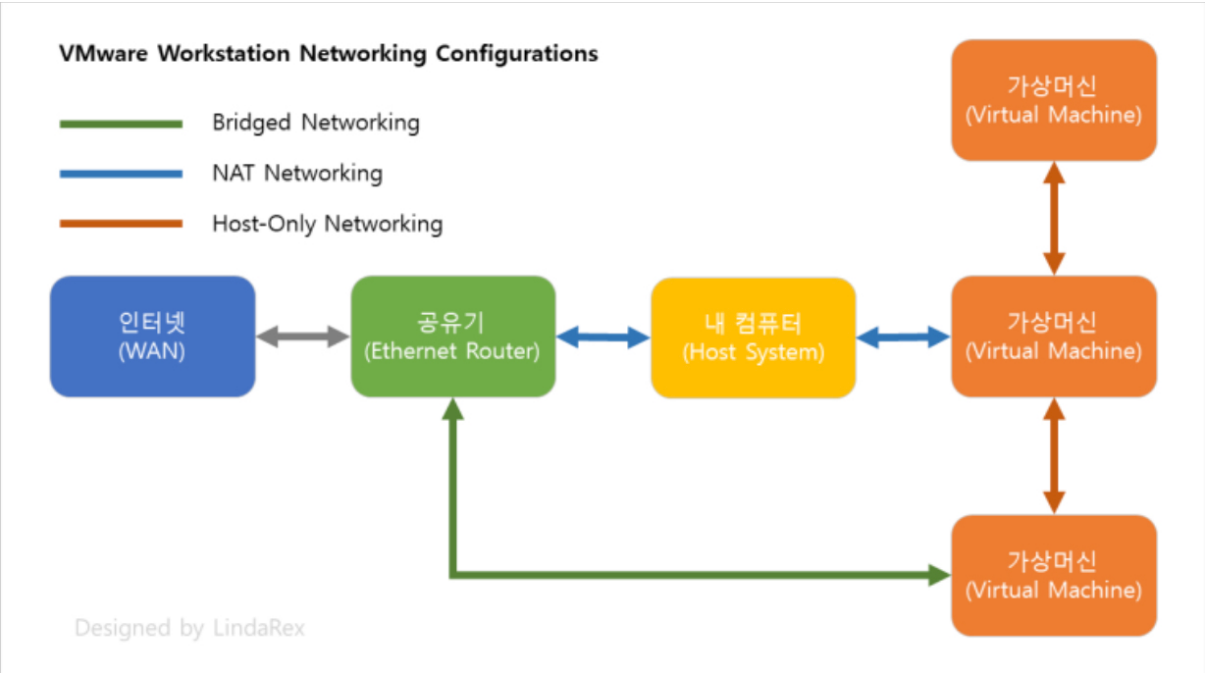
UTM이 문제인가 싶어 WSL2로 다시 해보려 했으나 이게 Ubuntu 부팅 iso파일이 겹쳐서 그런지 UTM도 정상적으로 작동이 되지 않았고... 도저히 해결 방법이 떠오르지 않던 도중 갑자기 한 가지 아이디어가 떠올랐습니다.

가상 머신에서 사용하는 인터넷이 Host PC가 사용하는 인터넷과 다를 수 있겠구나!!

그래서 조금 더 알아보니 VMWare의 네트워크 설정 방법에 관련하여 다음과 같은 개념들이 등장했습니다.

NAT Network VS Bridge Network

	Bridged	Host-Only	NAT
Virtual Adapter	VMnet0	VMnet1	VMnet8
다른 VM 연결	연결됨	연결됨	없음
호스트 연결	없음	연결됨	연결됨
외부 연결(인터넷 액세스)	Auto-bridging	없음	NAT
DHCP	없음	활성화됨	활성화됨
서브넷 주소		192.168.x.0	192.168.x.0
IP 주소		192.168.x.1	192.168.x.1
서브넷 마스크		255.255.255.0	255.255.255.0
기본 게이트웨이		없음	없음

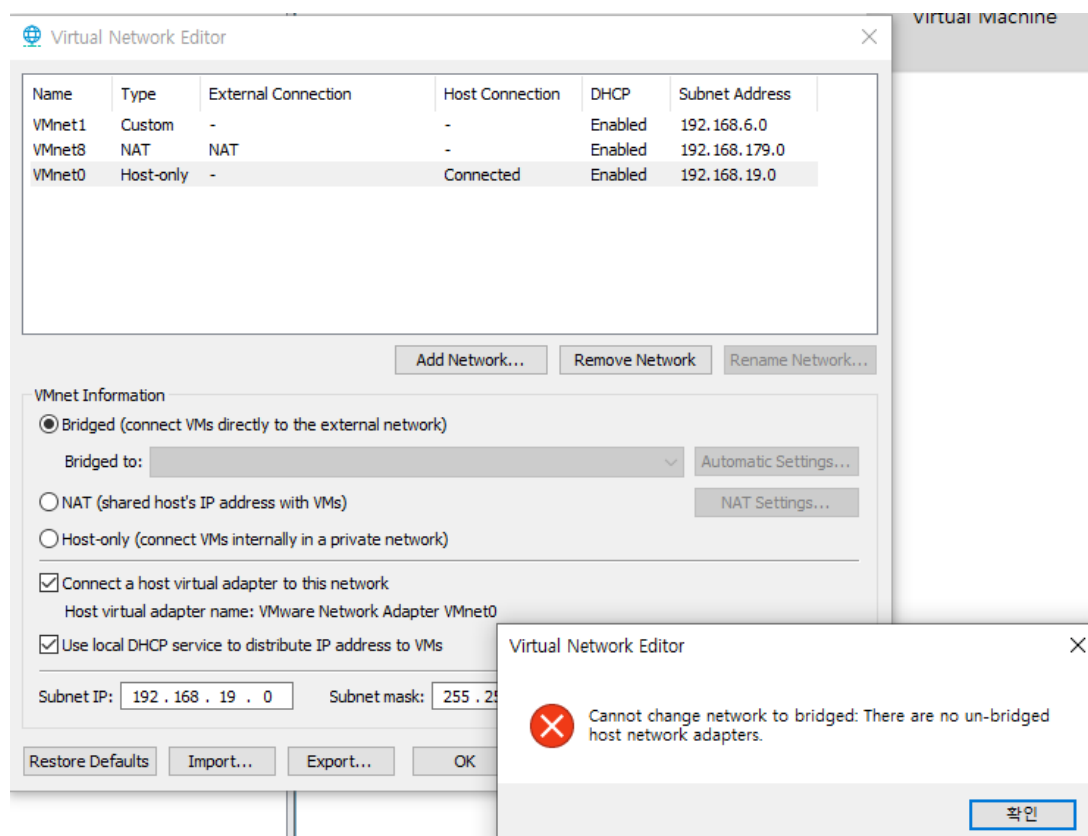


NAT과 Bridge 방식은 VMWare이 IP 할당을 어디서 받느냐의 차이로 나뉘다고 합니다. 결론적으로 VMWare을 NAT 방식으로 사용할 경우 IP를 Host PC로부터 받으며, Bridge 방식으로 사용할 경우 IP를 공유기 자체에서 부여받는다고 합니다. 따라서 VMWare Container의 네트워크 설정을 Bridge 방식으로 설정할 경우, 호스트 PC와 가상머신에 같은 네트워크 대역의 IP를 할당 받아 둘은 동일한 수준의 물리적 PC로 인식되고, 즉 Host PC가 사용하는 네트워크를 그대로 VMWare에서 사용하기 때문에 "microROS agent 연결이 정상적으로 이루어질 수 있겠구나" 라고 생각했었습니다.

출처 :

<https://blog.naver.com/PostView.naver?blogId=islove8587&logNo=223004216580>

그래서 VMWare Container의 네트워크 설정 방식을 Bridge 방식으로 바꿔 보려는 도중... 아래의 문제가 발생했습니다.



이유는 모르겠지만 un-bridge된 host network adapters가 없다는 문제가 계속 났습니다. 이 문제를 해결해 보려고 했는데 도저히 해결이 안되더군요... 그래서 또 다시 방법을 고민해 보다가, 일단 집에 가지고 있는 라즈베리파이 4에 Ubuntu 22-04 데스크톱 이미지를 올려서 여기에서 먼저 microROS agent를 설치해서 정상적으로 작동이 되는지 확인을 해보고, 만약 정상적으로 작동이 된다면 **라즈베리파이의 경우 공유기로부터 발급받은 ip주소를 그대로 사용하기 때문에 결국 위의 Bridge 관련 문제가 원인일 가능성이 매우 높다고 결론지을 수 있겠다**고 생각했었습니다.

그런데 하필이면 **가지고 있던 라즈베리파이 4가 고장난 상태**였습니다. 라즈베리파이 3에 이미지를 올려보려 했으나 Ubutnu 22.04 Desktop 이미지의 경우 라즈베리파이 4 이상에서만 지원을 한다고 하더군요. 그래서 위 방식은 해볼 수 없어서 또 다시 방법을 고민해 보던 중... **맥에서 동작하는 우분투 가상 머신이 있다는 사실**을 알게 되었습니다.

참고 : <https://solearn.tistory.com/275>

참고로 **인텔 맥의 경우에는 Ubuntu Server for AMD**를 다운받아야 합니다. 저의 경우 ARM으로 다운 받으니 정상적으로 작동이 되지 않았습니다.

위의 방식을 그대로 따라하여 Ubuntu 22.04 Desktop을 정상적으로 설치했고, 그리고 드디어!! 여기서는 Bridge 설정을 할 수 있었습니다. VMWare에서 위의 설정을 할 수 없었던 이유가 아마 제 노트북이 인텔 맥인 것이 원인이지 않을까 싶습니다.

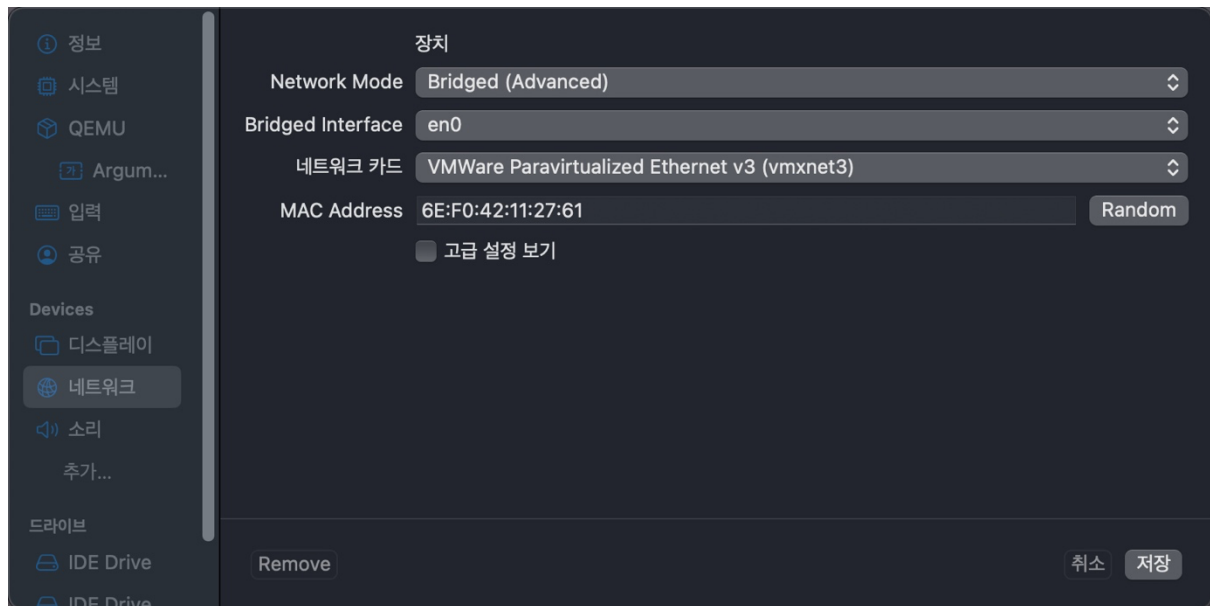
Bridge 네트워크로 설정하기 원하는 UTM 컨테이너에서 우클릭 -> Devices -> 네트워크에 들어간 다음, 아래의 그림과 같이

Network Mode : Bridged

Bridged Interface : en0

네트워크 카드 : VMWare Paravirtualized Ethernet v3 (vmxnet3)

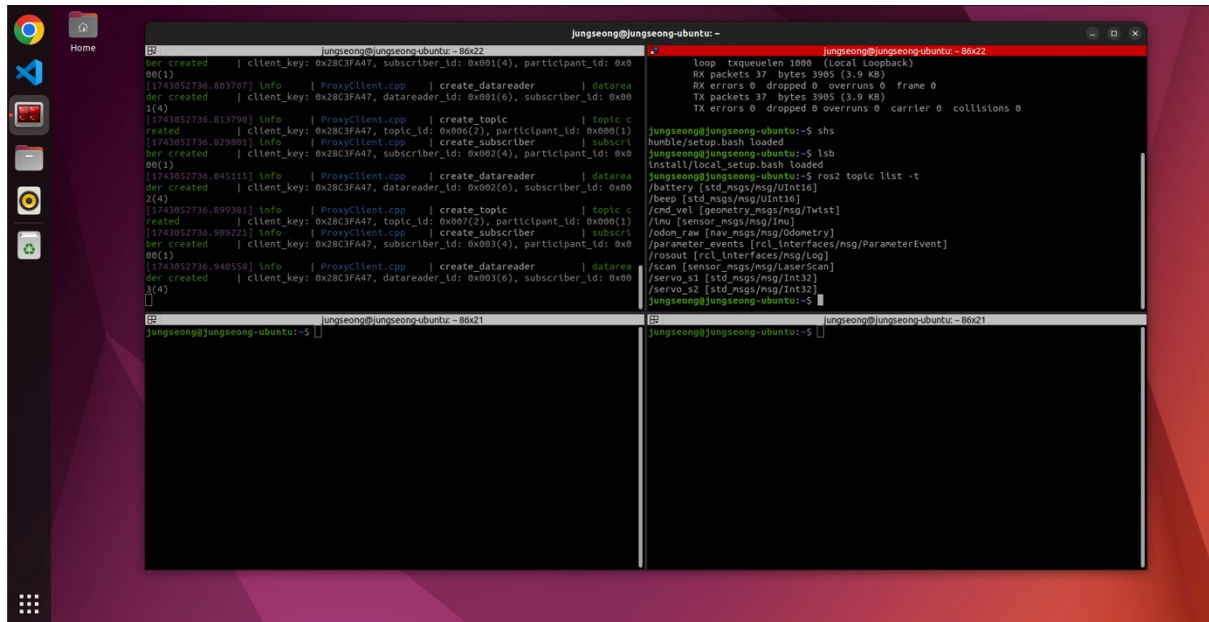
로 설정하시면 됩니다.



이후 해당 컨테이너에서 ROS2 Humble을 설치하고, 위의 ROS agent 설치 과정을 그대로 따라해 줍니다. ROS2 Humble 설치 링크는 아래와 같습니다.

<https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html>

결과 : 정상적으로 작동되는 것을 확인할 수 있었습니다.



The screenshot displays a desktop environment with three terminal windows running on a system named 'jungseong@jungseong-ubuntu'.

- Top Left Terminal (86x22):** Shows the output of a ROS2 node launch. It includes log messages for creating a datareader, subscribers, and topics. Key entries include:
 - client_key: 0x28C3FA47, subscriber_id: 0x001(4), participant_id: 0x000(4)
 - topic_id: 0x000(2), participant_id: 0x000(1)
 - subscriber_id: 0x002(4), participant_id: 0x000(1)
 - subscriber_id: 0x003(4), participant_id: 0x000(1)
 - subscriber_id: 0x003(4), participant_id: 0x000(1)
- Top Right Terminal (86x22):** Displays the output of the 'shh' command, showing network statistics for a local loopback connection:
 - loop txqueuelen 1000 (Local Loopback)
 - RX packets 37, bytes 3905 (3.9 KB)
 - TX packets 37, bytes 3905 (3.9 KB)
- Bottom Left Terminal (86x21):** Shows the output of the 'ros2 topic list -t' command, listing several topics and their data types:
 - /battery [std_msgs/msg/UInt16]
 - /beep [std_msgs/msg/UInt16]
 - /cmd_vel [geometry_msgs/msg/Twist]
 - /imu [sensor_msgs/msg/Imu]
 - /odom_raw [nav_msgs/msg/Odometry]
 - /parameter_events [rcl_interfaces/msg/ParameterEvent]
 - /rosout [rcl_interfaces/msg/Log]
 - /scan [sensor_msgs/msg/LaserScan]
 - /servo_s1 [std_msgs/msg/Int32]
 - /servo_s2 [std_msgs/msg/Int32]