

신입생교육_ROS2, Gazebo

ROS란 무엇인가?

- Robot Operating System의 약자
- 로봇 어플리케이션 개발에 필요한 tools, library를 제공하는 open source platform



"The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source."

- **Operating system:** 컴퓨터에서 특정 프로그램을 작성하고 구동시키는 것처럼, 로봇의 구동에 필요한 요소들을 제공해준다.
- 예를 들어, 로봇의 제어 코드를 작성하고 하드웨어를 작동시키려 한다. 제어 코드 작성에 필요한 library, 로봇과 PC 간의 메시지 통신, 패키지 관리 등을 직접 구현하려면 많은 개발 인력이 필요하고, 오랜 시간이 걸릴 것이다. 이를 쉽게 만들어주는 것이 ROS라고 생각하면 된다.

왜 ROS2인가?

- 2007년에 개발이 시작된 ROS1을 이용하며 단점을 보완할 필요성 제기
- 약 10년간 ROS 1을 사용한 경험으로 사용자의 요구를 수용하며 새로 개발한 ROS2 출시

ROS1

- 단일 로봇
- 워크스테이션급 컴퓨터
- 실시간 제어 지원 X
- 안정된 네트워크 환경 요구
- Linux 환경

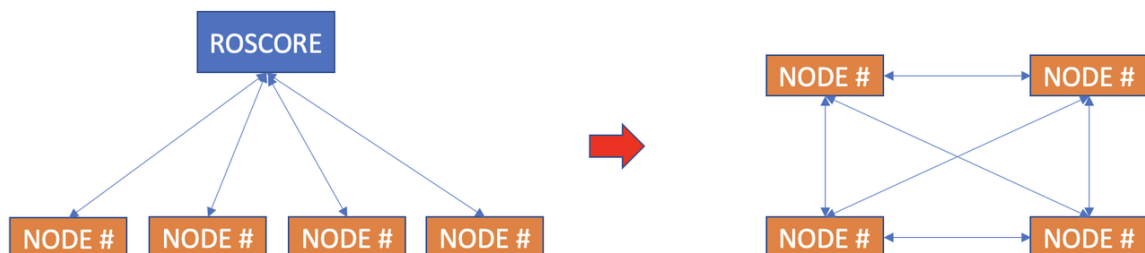
ROS2

- 복수 대의 로봇
- 임베디드 시스템에서의 사용
- 실시간 제어 지원 O
- 불안정한 네트워크 환경에서도 동작
- 멀티 플랫폼(Linux, macOS, Windows)

ROS1과 차이점


1. DDS(Data Distribution service) 아키텍처가 적용

- peer to peer로 실시간과 다양한 데이터 타입을 효과적으로 교환할 수 있다.
- ROS1에서는 roscore라는 중간다리가 있기 때문에 불필요한 코스트가 컸고, 노드간 frequency 문제가 있었다.



ROS2에서 ROSCORE 안쓰는 이유 · Chan Blog

Hi! thank you for visiting my blog!

 https://leechangyo.github.io/ros/2021/03/22/ROS2_ROSCORE/

2. Real-time

- 선별된 하드웨어, 리얼타임 운영체제, DDS의 RTPS(Real-time Publish-Subscribe Protocol)와 같은 통신 프로토콜을 사용 시 실시간성 지원
- 통신 프로토콜: 실시간 RTPS(Real Time Publish Subscribe)

- UDP 기반이지만, QoS 설정을 통해 UDP/TCP의 기능을 선택적으로 사용 가능함.
- ROS1은 TCP기반의 TCPROS 통신 방식을 사용했는데, 통신 방식의 특성 상 실시간성을 확보하기 어려웠다.

프로토콜 종류	TCP	UDP
연결 방식	연결형 서비스	비연결형 서비스
패킷 교환 방식	가상 회선 방식	데이터그램 방식
전송 순서	전송 순서 보장	전송 순서가 바뀔 수 있음
수신 여부 확인	수신 여부를 확인함	수신 여부를 확인하지 않음
통신 방식	1:1 통신	1:1 OR 1:N or N:N 통신
신뢰성	높다	낮다
속도	느리다	빠르다

3. Build system & tools

- **Build system:** 단일 시스템을 빌드하여 실행 가능한 파일을 생성
 - C++: CMake(Cross Platform Make) 기반의 catkin과 ament_cmake
 - Python: Python setuptools
- **Build tool:** ROS는 수많은 패키지가 함께 빌드하여 실행, 각 패키지별로 다른 빌드 시스템을 호출하고 의존성을 해결해야함 > 빌드 툴 사용
- ROS1은 catkin-CMake만 지원했지만, ROS2는 CMake를 사용하지 않는 Python 패키지 관리도 가능
- Build tool로는 **colcon**을 추천, ROS2 패키지를 작성, 테스트, 빌드하는 명령어 도구
- **Build option**
 - 크게 세 가지의 변화점

(1) Multiple workspace

ROS 2에서는 복수의 독립된 워크스페이스를 사용할 수 있어서 작업 목적 및 패키지 종류별로 관리할 수 있게 되었다.

(2) No non-isolated build

ROS 2에서는 모든 패키지를 별도로 빌드한다. 이 기능 변화를 통해 설치용 폴더를 분리하거나 병합할 수 있게 되었다.

(3) No devel space

ROS 2에서는 패키지를 빌드한 후 설치해야 패키지를 사용할 수 있다.

단, --symlink-install과 같은 옵션도 제공해 기존과 같이 사용할 수도 있다.

[ROS2]021: ROS 2의 빌드 시스템과 빌드 툴

<https://cafe.naver.com/openrt/24411> 1. ROS 2의 빌드 시스템 (build system)과 빌드 툴(build tools) 결론부터 말하자면 빌드 시스템은 단일 패키지를 대상으로 하며, 빌드 툴은 시스템 전체를 대상으로

<https://velog.io/@hwang-chaewon/ROS2035>

```
Cloning into '...'...
=== src/ament/googletest (git) ===
Cloning into '...'...
=== src/ament/ament_utilities_vendor (git) ===
Cloning into '...'...
=== src/ament/ament_utilities_vendor (git) ===
Cloning into '...'...
Note: switching to 'v1.0.13'.
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:
git switch -c <new-branch-name>
Or undo this operation with:
git switch -
```

설치

ROS2 설치

VSCode 연동

- VSCode Extension에서 WSL 설치

WSL & WSL2 설치와 VSCode 연동하기

WSL과 WSL2 설치 및 Remote - WSL을 통한 VSCode와의 연동

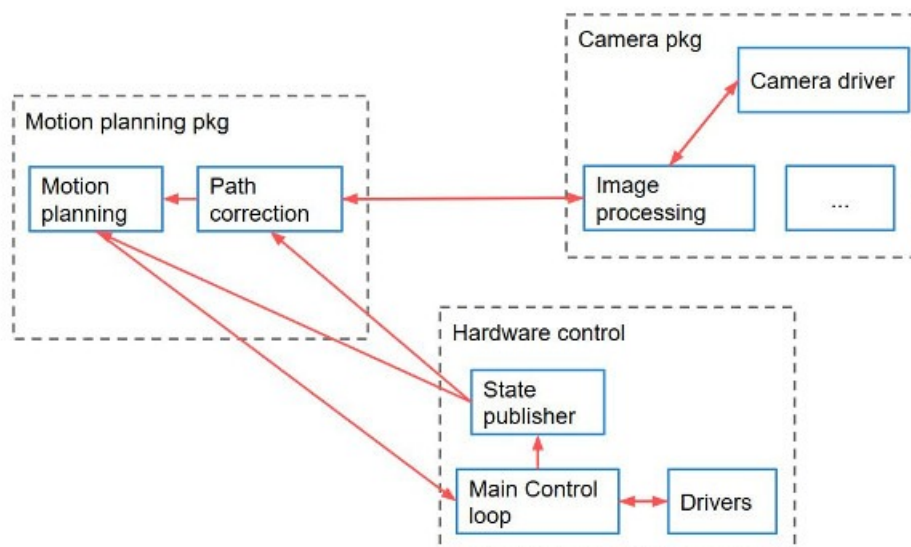
<https://velog.io/@gidskql6671/WSL-WSL2-설치-VSCode-연동>

기능을 사용하려면 해당 확인란을 선택하고 기능을 사용하지 않으려면 확인란의 선택을 취소하십시오. 확인란이 검은 사각형으로 채워진 경우는 해당 기능의 일부만이 사용되고 있는 것입니다.

☒ .NET Framework 3.5(.NET 2.0 및 3.0 포함)
☒ .NET Framework 4.8 Advanced Services
☒ Internet Explorer 11
☒ Linux용 Windows 하위 시스템
☒ Microsoft PDF로 인쇄
☒ Microsoft XPS Document Writer
☐ MSMQ(Microsoft Message Queue) Server
☐ SMB 1.0/CIFS 파일 공유 지원
☐ TFTP 클라이언트
☐ Windows Identity Foundation 3.5

ROS2 사용방법

- ROS 시스템은 크게 package, node, topic으로 구성됨



0. Package

ROS2: 패키지(package) 만들기

패키지(package)는 하나 이상의 노드(node)가 기능적 단위로 묶인 ROS 코드의 컨테이너입니다. 코드를 설치하거나 공유하기 위해서는 패키지를 구성해야 합니다. 1. 빌드 시스템(build system)과 빌드 툴

 https://www.robotstory.co.kr/king/?board_page=3&vid=884

```
3 <package format="3">
4   <name>ament_cmake</name>
5   <version>0.0.0</version>
6   <description>TODD: Package description</description>
7   <maintainer email="todd@todo.todo">todd</maintainer>
8   <license>TODD: License declaration</license>
9
10  <buildtool_depend>ament_cmake</buildtool_depend>
11
12  <test_depend>ament_lint_auto</test_depend>
13  <test_depend>ament_lint_common</test_depend>
14
15  <export>
16    <build_type>ament_cmake</build_type>
17  </export>
18 </package>
```

- 하나 이상의 노드(Node)가 묶인 ROS 코드의 컨테이너, 코드를 설치하거나 공유하기 위해서 패키지를 구성해야 함
- **package 생성**
 - 명령어: `ros2 pkg create [패키지 이름] --build-type [빌드 타입]`

```
cd ~/dev_ws/src
ros2 pkg create freshman_ros --build-type ament_cmake
```

- ROS package 구조

```
dev_ws
├── src
│   ├── freshman_ros # 패키지 이름
│   │   ├── include
│   │   ├── src
│   │   ├── CMakeLists.txt
│   │   └── package.xml
```

1) package.xml

- ROS 패키지의 필수 구성요소, 패키지 정보를 기술

2) CMakeLists.txt

- CMake 빌드 설정 파일, 빌드에 필요한 정보 기술

```

package.xml (~dev_ws/src/cmake_pkg)
파일(F) 편집(E) 보기(V) 검색(S) 도구(T) 문서(D) 도움말(H)
package.xml X
1 <?xml version="1.0"?>
2 <?xml-model href="http://download.ros.org/schema/package_format3.xsd"
3 schemaTypes="http://www.w3.org/2001/XMLSchema"?>
4 <package format="3">
5   <name>ament_cmake</name>
6   <description>TODO: Package description</description>
7   <maintainer email="TODO" />
8   <license>TODO: License declaration</license>
9   <buildtool_depend>ament_cmake</buildtool_depend>
10
11
12   <test_depend>ament_lint_auto</test_depend>
13   <test_depend>ament_lint_common</test_depend>
14
15   <export>
16     <build_type>ament_cmake</build_type>
17   </export>
18 </package>

```

```

CMakeLists.txt (~dev_ws/src/cmake_pkg)
파일(F) 편집(E) 보기(V) 검색(S) 도구(T) 문서(D) 도움말(H)
CMakeLists.txt X
1 cmake_minimum_required(VERSION 3.5)
2 project(cmake_pkg)
3
4 # Default to C99
5 if(NOT CMAKE_C_STANDARD)
6   set(CMAKE_C_STANDARD 99)
7 endif()
8
9 # Default to C++14
10 if(NOT CMAKE_CXX_STANDARD)
11   set(CMAKE_CXX_STANDARD 14)
12 endif()
13
14 if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
15   add_compile_options(-Wall -Wextra -Wpedantic)
16 endif()
17
18 # find dependencies
19 find_package(ament_cmake REQUIRED)
20 # uncomment the following section in order to fill in
21 # further dependencies manually.
22 # find_package(<dependency> REQUIRED)
23
24 if(BUILD_TESTING)
25   find_package(ament_lint_auto REQUIRED)
26   # the following line skips the linter which checks for copyrights
27   # uncomment the line when a copyright and license is not present in all source files
28   #set(ament_cmake_copyright_FOUND TRUE)
29   # the following line skips cpplint (only works in a git repo)
30   # uncomment the line when this package is not in a git repo
31   #set(ament_cmake_cpplint_FOUND TRUE)
32   ament_lint_auto_find_test_dependencies()
33 endif()
34
35 ament_package()

```

3) include & src

- include: 코드에서 사용할 library 등이 포함
- src: 실행 혹은 참고할 코드 포함, 유저 코드가 들어가는 곳

• package 빌드

- 패키지를 작성하고 사용하기 위해서는 빌드 과정을 거쳐야한다. 뒷 부분에서 예시와 함께 알아보자.

▼ Build system

Build System

- colcon: ROS1의 catkin_make와 비슷한 기능을 하는 툴

workspace 생성

```
mkdir -p ~/dev_ws/src
```

코드 복제(or 작성)

```
git clone https://github.com/ros/ros_tutorials.git -b foxy
```

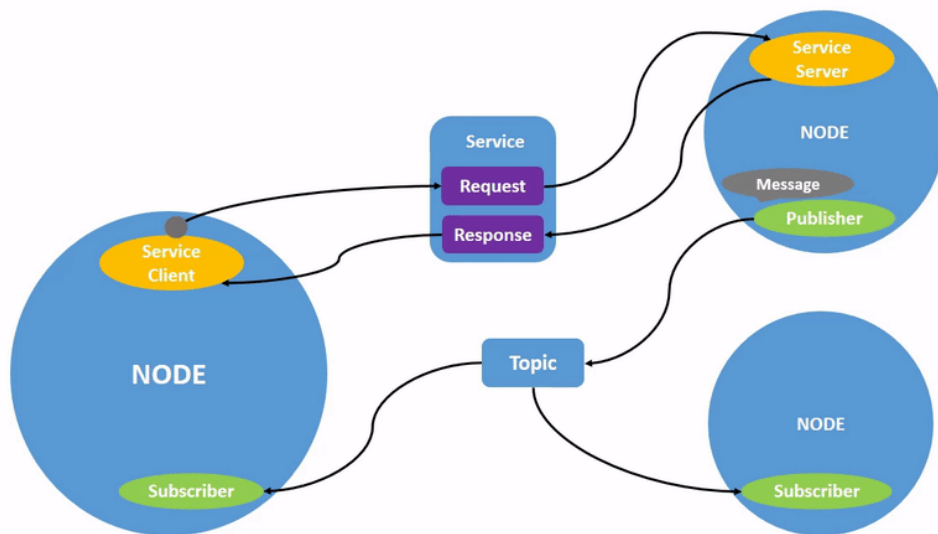
빌드

```
cd ~/dev_ws && colcon build --symlink-install
```

- --symlink-install : 빌드 시 install 폴더로 파일 복사하는 것이 아니라 심볼릭 파일 생성

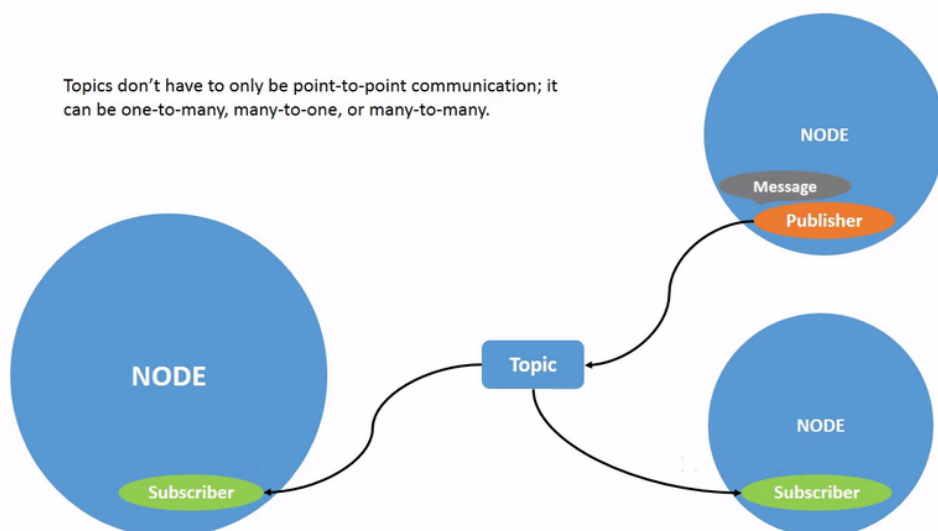
1. Node

- ROS2의 최소 실행 단위, 하나의 목적을 가진 실행가능한 프로그램을 의미한다.
- 각 노드는 메시지 통신으로 데이터를 주고받는다.



2. Topics

Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.



- 각 노드가 메시지를 주고받는 방식 중 하나이다.

- 정보를 보내는 Publisher 노드와 정보를 받는 Subscriber 노드가 Topic 메시지 형태로 정보를 송수신한다.
- 특징
 - 비동기식 단방향 메시지 송수신 방식
 - Publisher와 Subscriber 간의 통신
 - 1:N, N:1, N:N 통신도 가능
 - 통신을 중단하기 전까지 연속적으로 메시지를 보내는 것
- 다른 메시지 송수신 방식으로는 service, action 등이 있다.

Exercise: Simple message publish/subscribe code

Writing a simple publisher and subscriber (C++) — ROS 2 Documentation: Foxy documentation

You're reading the documentation for a version of ROS 2 that has reached its EOL (end-of-life), and is no longer officially supported.

If you want up-to-date information, please have a look at Iron.

🔗 <https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html>

0. Create a package

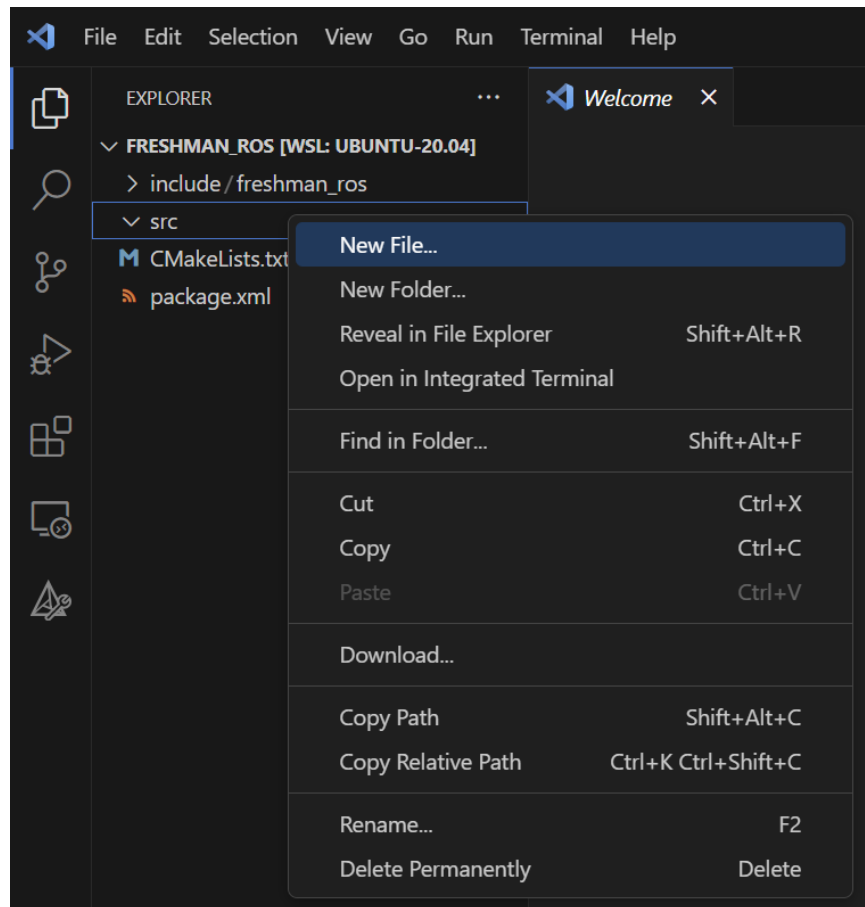
- 앞서 만든 freshman_ros 패키지 이용

1. Write the publisher node

- VSCode로 freshman_ros 패키지 open

```
cd dev_ws/src/freshman_ros
code .
```

- src 우클릭 > New File > simple_publisher.cpp



- 아래의 코드 copy

▼ simple_publisher.cpp

```
#include <chrono>
#include <functional>
#include <memory>
#include <string>
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using namespace std::chrono_literals;

class CppPublisher : public rclcpp::Node
{
public:
    CppPublisher()
    : Node("simple_publisher"), count_(0)
    {
        publisher_ = this->create_publisher<std_msgs::msg::I
```

```

        timer_ = this->create_wall_timer(500ms, std::bind(&C
    }

private:
    void timer_callback()
    {
        auto message = std_msgs::msg::Int32();
        message.data = count_++;
        RCLCPP_INFO(this->get_logger(), "Publishing: %d", me
        publisher_->publish(message);
    }
    rclcpp::TimerBase::SharedPtr timer_;
    rclcpp::Publisher<std_msgs::msg::Int32>::SharedPtr pub
    size_t count_;
};

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<CppPublisher>());
    rclcpp::shutdown();
    return 0;
}

```

2. Write the subscriber node

- publisher 만들때와 동일하게 simple_subscriber.cpp 생성

▼ simple_subscriber.cpp

```

#include <memory>
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using std::placeholders::_1;

class CppSubscriber : public rclcpp::Node
{
public:

```

```

    CppSubscriber()
    : Node("simple_subscriber")
    {
        subscription_ = this->create_subscription<std_msgs::Int32>("topic", 10, std::bind(&CppSubscriber::topic_callback, this, _1));
    }

private:
    void topic_callback(const std_msgs::msg::Int32 & msg)
    {
        RCLCPP_INFO(this->get_logger(), "Hello, Robotory: %d", msg.data);
    }
    rclcpp::Subscription<std_msgs::msg::Int32>::SharedPtr subscription_;
};

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<CppSubscriber>());
    rclcpp::shutdown();
    return 0;
}

```

3. Add dependancy and make executable file

package.xml

- rclcpp, std_msgs에 대한 dependancy 추가

```

...
<buildtool_depend>ament_cmake</buildtool_depend>

<depend>rclcpp</depend>
<depend>std_msgs</depend>
...

```

▼ package.xml

```

<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_f
<package format="3">
  <name>freshman_ros</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="junha@todo.todo">junha</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>

```

CMakeLists.txt

- Dependency 포함, executable 추가

```

...
# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)

add_executable(simple_pub src/simple_publisher.cpp)
ament_target_dependencies(simple_pub rclcpp std_msgs)
add_executable(simple_sub src/simple_subscriber.cpp)
ament_target_dependencies(simple_sub rclcpp std_msgs)

```

```
install(TARGETS
  simple_pub
  simple_sub
  DESTINATION lib/${PROJECT_NAME})
...
```

▼ CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(freshman_ros)

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)

add_executable(simple_pub src/simple_publisher.cpp)
ament_target_dependencies(simple_pub rclcpp std_msgs)
add_executable(simple_sub src/simple_subscriber.cpp)
ament_target_dependencies(simple_sub rclcpp std_msgs)

install(TARGETS
  simple_pub
  simple_sub
  DESTINATION lib/${PROJECT_NAME})

ament_package()
```

4. Build and Run

- 만든 패키지 빌드

```
cd ~/dev_ws  
colcon build
```

- install

```
source ~/dev_ws/install/setup.bash
```

- 나중에 빌드할때마다 입력하기 귀찮으니 .bashrc 파일 수정하는 것 추천

```
...  
source ~/dev_ws/install/setup.bash  
  
alias eb='gedit ~/.bashrc'  
alias sb='source ~/.bashrc'  
alias cb='cd ~/dev_ws && colcon build'  
...
```