ROS2: Build Package using colcon

운영체제의 실제

안인규 (Inkyu An)

ROS build tools: colcon

- ROS packages are often divided into multiple smaller packages, and some developers modify several packages simultaneously. Building each package one by one and manually setting dependencies is cumbersome and prone to errors.

- In ROS 1, there were several build tools like *catkin_make*, *catkin_make_isolated*, and *catkin_tools*. In the early days of ROS 2, ament_tools was used. However, these tools had overlapping functionalities and many shortcomings, and also posed a maintenance burden.

- Therefore, the need for a "unified build tool" emerged, one that could build both ROS 1 and ROS 2 packages, and even packages from outside of ROS, all together.

ROS build tools: colcon

도구	특징 / 장점	단점
catkin_make	- 기본적인 build tool - 모든 package를 하나의 Cmake 프로젝트로 취급 (모든 소스 코드를 하나 한번에 처리)	- 모든 package가 하나의 build 공간을 공유하기 때문에, package간 충돌이 발생할 위험이 있음 (예: A package → B package)
catkin_make_isolated	- Package를 독립된 (isolated) 공간에서 하나씩 순서대로 build - Package간 간섭이 없음	- Package 수만큼 build를 반복 (속도가 매우 느림) - Package의 build 순서를 결정하지 않음 (A → B → C)
catkin_tools	- Package의 dependency를 파악한 뒤, dependency가 없는 package를 먼저 build	- ROS2 지원 X

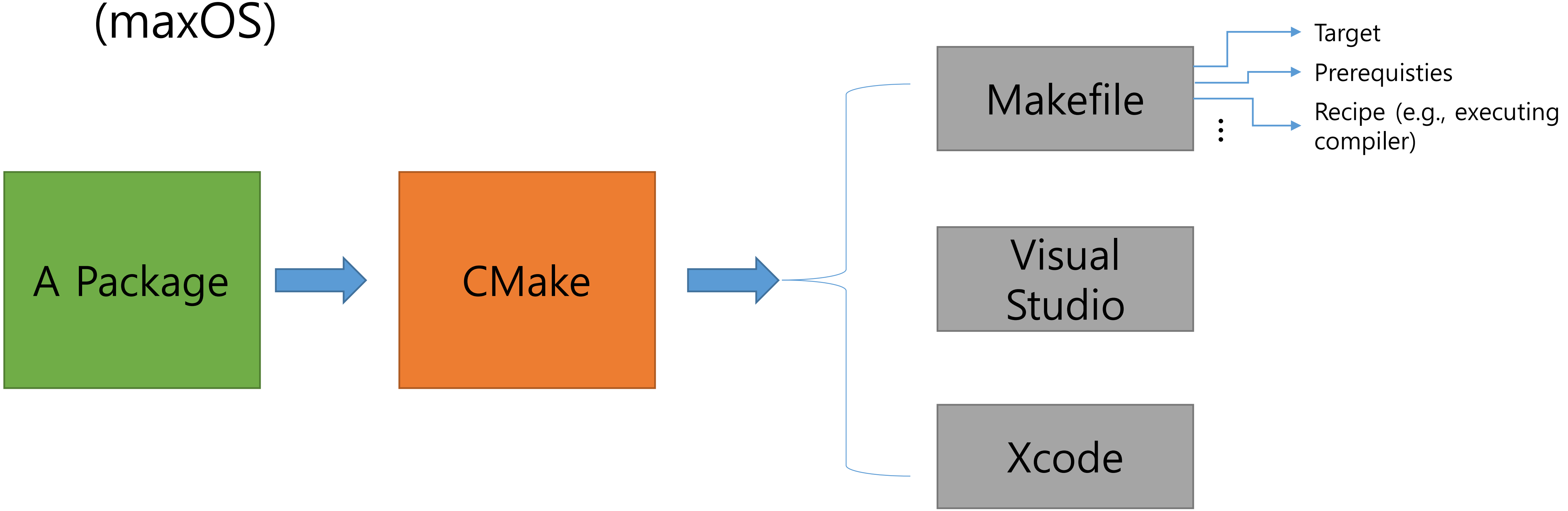
ROS build tools: colcon

도구	특징 / 장점	단점
catkin_make	• 기본적인 build tool • 모든 package를 하나의 Cmake 프로젝트로 취급 (모든 소스 코드를 하나 한번에 처리)	• 모든 package가 하나의 build 공간을 공유하기 때문에, package간 충돌이 발생할 위험이 있음 (예: A package → B package)
catkin_make_isolated	• Package를 독립된 (isolated) 공간에서 하나씩 순서대로 build • Package간 간섭이 없음	• Package 수만큼 build를 반복 (속도가 매우 느림) • Package의 build 순서를 결정하지 않음 (A → B → C)
catkin_tools	• Package의 dependency를 파악한 뒤, dependency가 없는 package를 먼저 build	• ROS2 지원 X

colcon 은 ROS1 build tool의 장점은 취하고, 단점을 보완

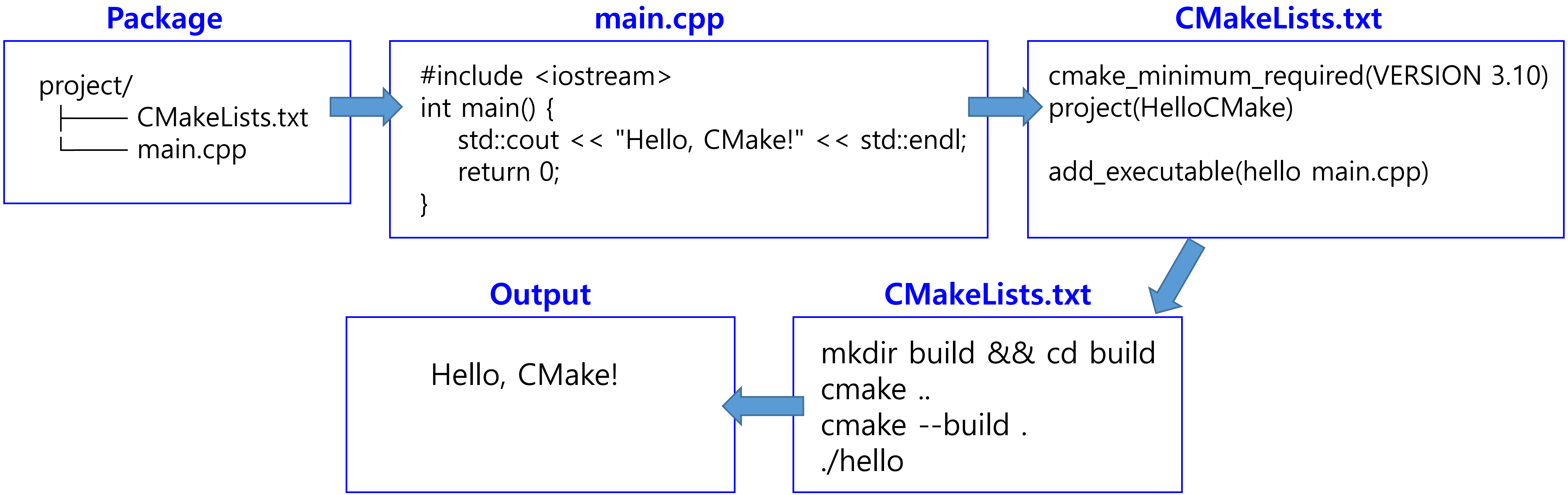
What is CMake?

- **CMake**: Cross platform build system generator
- **Background**: Each platform (e.g., OS) has a different build system: Makefile (Linux/Unix), Visual Studio (Windows), Xcode (maxOS)



What is CMake?

- **CMake**: Cross platform build system generator



Package

```
project/
├──── CMakeLists.txt
└──── main.cpp
```

main.cpp

```cpp
#include <iostream>
int main() {
    std::cout << "Hello, CMake!" << std::endl;
    return 0;
}
```

CMakeLists.txt

```cmake
cmake_minimum_required(VERSION 3.10)
project(HelloCMake)

add_executable(hello main.cpp)
```

CMakeLists.txt

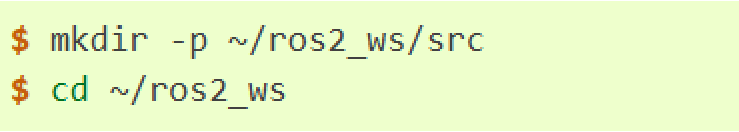
```
mkdir build && cd build
cmake ..
cmake --build .
./hello
```

Output

```
Hello, CMake!
```

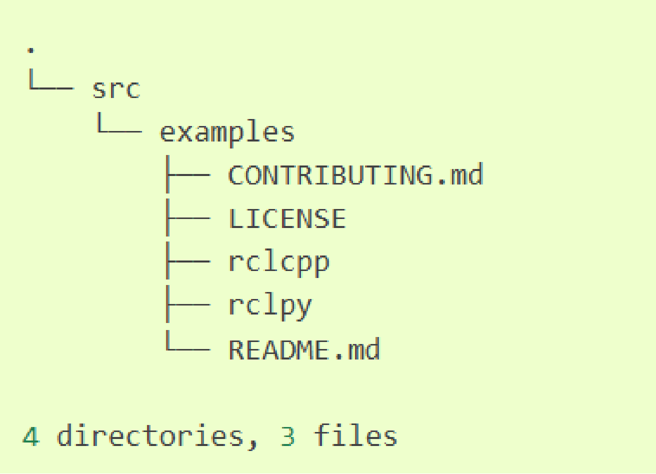
ROS build tools: colcon

- colcon's Workspace
  - A subdirectory: src (the source code of ROS packages is located)
  - This directory usually starts out empty otherwise



```
$ mkdir -p ~/ros2_ws/src
$ cd ~/ros2_ws
```

- Add some sources (example)
  - *git clone https://github.com/ros2/examples src/examples -b humble*



```
.
└── src
    └── examples
        ├── CONTRIBUTING.md
        ├── LICENSE
        ├── rclcpp
        ├── rclpy
        └── README.md

4 directories, 3 files
```

ROS build tools: colcon

- Underlay and Overlay
  - We can mange multiple colcon's workspaces
  - ROS2 recommends utilizing multiple workspaces

Q: Why do we have to separate them?

```
/opt/ros/humble/        ← underlay (기존 설치)
~/ros2_ws/              ← overlay (내가 개발 중인 패키지)
    ├──── src/          ← 내가 작성한 패키지 소스
    ├──── build/
    ├──── install/
    └──── log/
```

- To utilize overlay, we have to source them

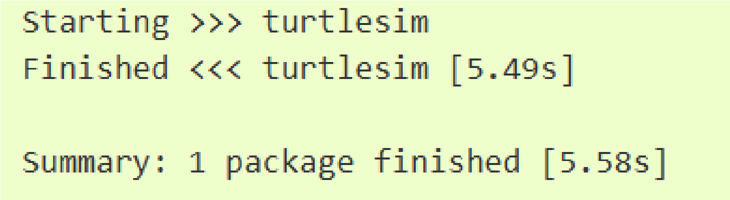
```
source /opt/ros/humble/setup.bash
source ~/ros2_ws/install/setup.bash
```

Q: What if a package with the same name exists in both?

ROS build tools: colcon

- Build the workspace
  - *colcon build --symlink-install*

Q: What is the symbolic link?



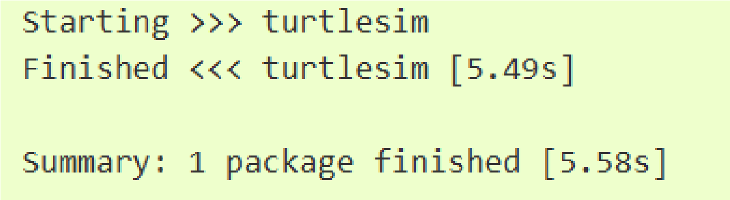
```
Starting >>> turtlesim
Finished <<< turtlesim [5.49s]

Summary: 1 package finished [5.58s]
```

ROS build tools: colcon

- Build the workspace
  - *colcon build --symlink-install*

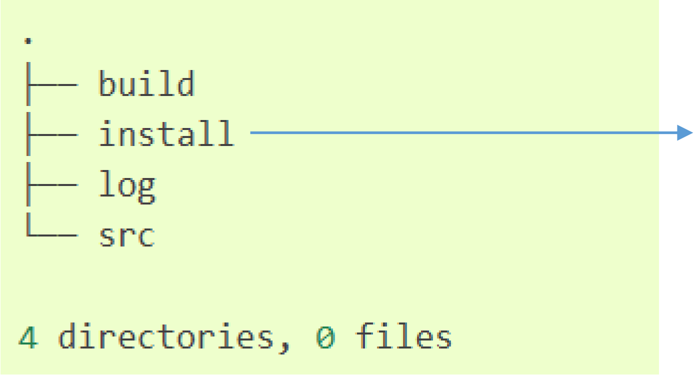
Q: What is the symbolic link?

Instead of copying directly into the install space, **a symbolic link** is created



```
Starting >>> turtlesim
Finished <<< turtlesim [5.49s]

Summary: 1 package finished [5.58s]
```

- After the build is finished, we should see the *build*, *install*, and *log* directory

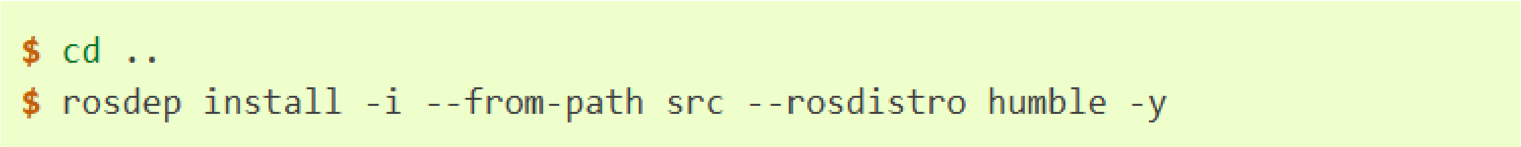


```
.
├── build
├── install
├── log
└── src

4 directories, 0 files
```

This is where our workspace's setup files are, which we can use to source our overlay

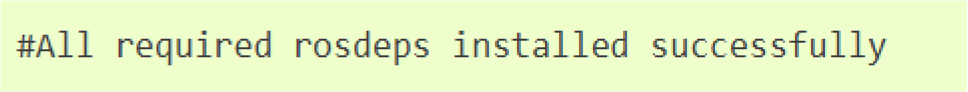
ROS build tools: colcon

- Resolve dependencies (<u>optional</u>)
  - After building the workspace, if the build fails, we can see that a package dependency is required



```
$ cd ..
$ rosdep install -i --from-path src --rosdistro humble -y
```

Check dependencies from "package.xml" and install the dependencies



```
#All required rosdeps installed successfully
```

ROS build tools: colcon

- “Run tests” automatically runs the tests written in ROS 2 and general packages.
  - *colcon test*

  - For this to actually work, the test code must already be defined within the package

CMakeLists.txt (gtest 사용)

```
ament_add_gtest(test_talker test/test_talker.cpp)
ament_target_dependencies(test_talker rclcpp std_msgs)
```

ROS build tools: colcon

- Source the environment
  - *source install/setup.bash*

- Try a demo
  - A subscriber node:
    ros2 run examples_rclcpp_minimal_subscriber subscriber_member_function
  - A Publisher node
    ros2 run examples_rclcpp_minimal_publisher publisher_member_function

ROS build tools: colcon

- Create our own package: what makes up a ROS 2 package?
  - ROS 2 Python and CMake packages each have their own minimum required contents:

<CMake (C/C++)>

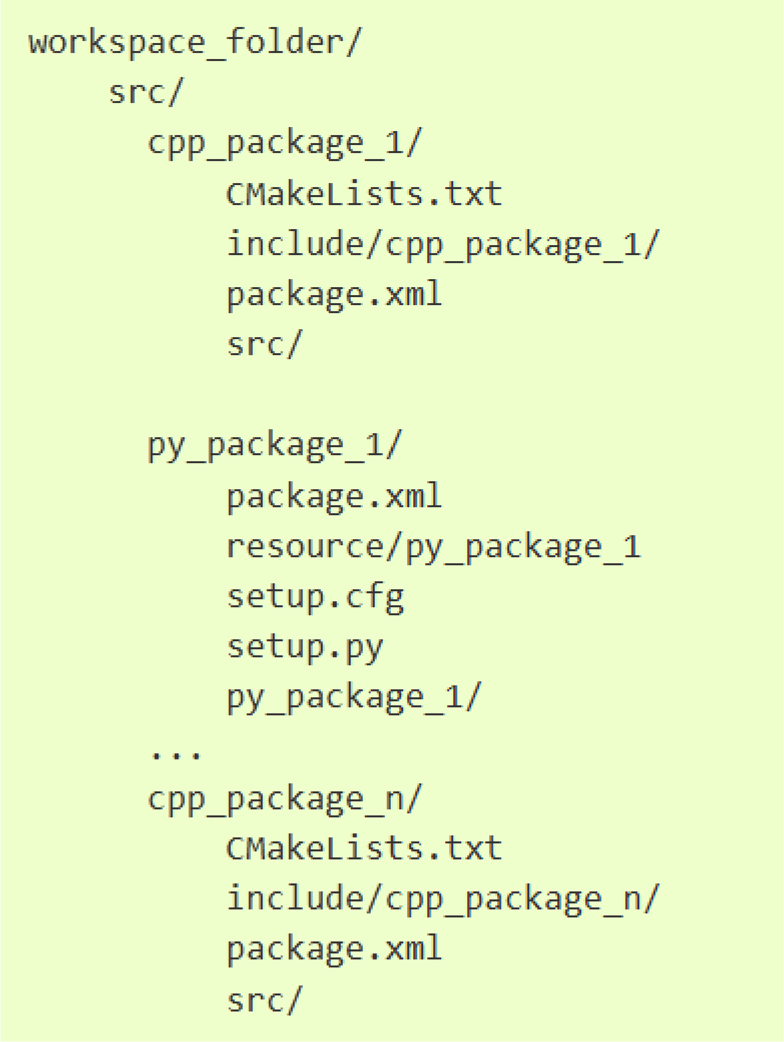
- **CMakeLists.txt** file that describes how to build the code within the package
- **include/<package_name>** directory containing the public headers for the package
- **package.xml** file containing meta information about the package
- **src** directory containing the source code for the package

<Python>

- **package.xml** file containing meta information about the package
- **resource/<package_name>** marker file for the package
- **setup.cfg** is required when a package has executables, so ros2 run can find them
- **setup.py** containing instructions for how to install the package
- **<package_name>** - a directory with the same name as your package, used by ROS 2 tools to find your package, contains **__init__.py**

ROS build tools: colcon

- **Examples**: 3 Packages in a workspace
  - A single workspace can contain as many packages as you want, each in their own folder
  - You can also have packages of different build types in one workspace (CMake, Python, etc.)
  - Best practice is to have a src folder within your workspace, and to create your packages in there
  - This keeps the top level of the workspace “clean”



```
workspace_folder/
    src/
      cpp_package_1/
          CMakeLists.txt
          include/cpp_package_1/
          package.xml
          src/

      py_package_1/
          package.xml
          resource/py_package_1
          setup.cfg
          setup.py
          py_package_1/
      ...
      cpp_package_n/
          CMakeLists.txt
          include/cpp_package_n/
          package.xml
          src/
```

ROS build tools: colcon

1. Go to the src folder:
   - *cd ~/ros2_ws/src*
2. Create a new command
   - *ros2 pkg create --build-type ament_python --license Apache-2.0 <package_name>*

- ament_cmake → C++
- ament_python → Python

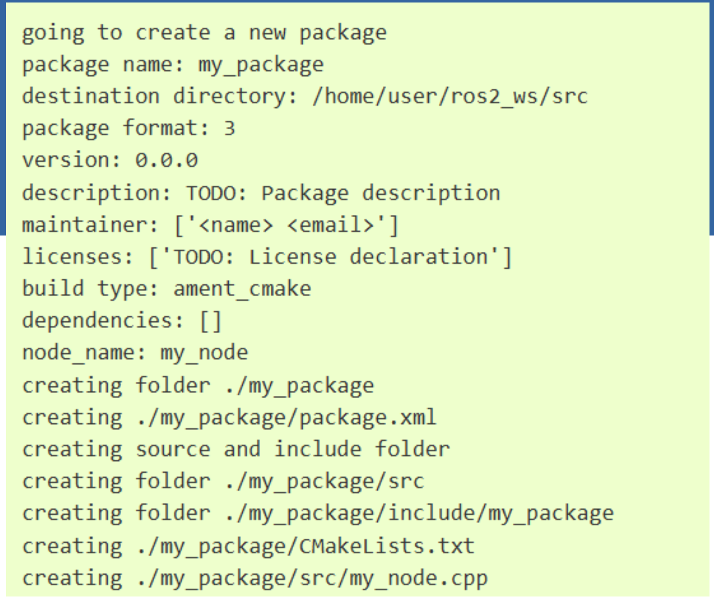
```
CMakeLists.txt   include   package.xml   src
```

- *ros2 pkg create --build-type ament_cmake --license Apache-2.0 <package_name>*

```
my_package   package.xml   resource   setup.cfg   setup.py   test
```

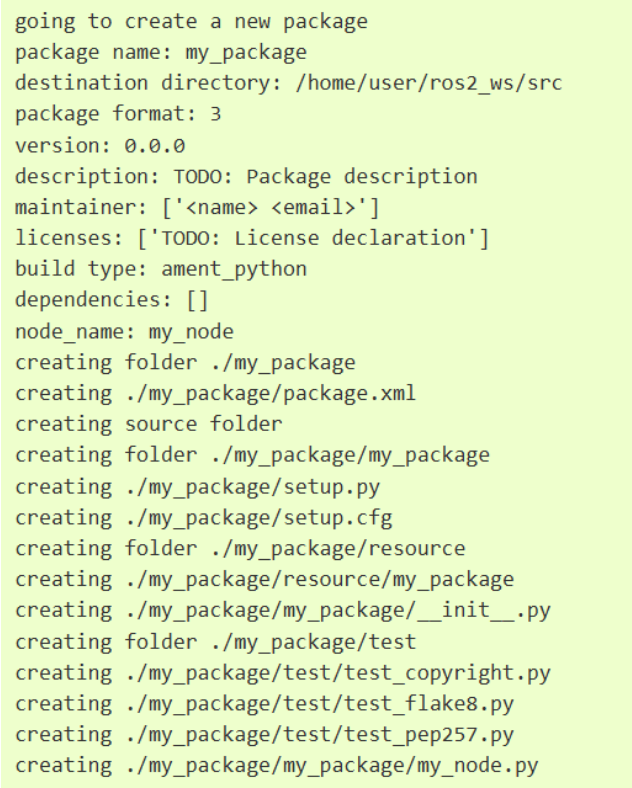
ROS build tools: colcon

- Examples
  - *ros2 pkg create --build-type ament_cmake --license Apache-2.0 --node-name my_node my_package*



```
going to create a new package
package name: my_package
destination directory: /home/user/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['<name> <email>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: []
node_name: my_node
creating folder ./my_package
creating ./my_package/package.xml
creating source and include folder
creating folder ./my_package/src
creating folder ./my_package/include/my_package
creating ./my_package/CMakeLists.txt
creating ./my_package/src/my_node.cpp
```

  - *ros2 pkg create --build-type ament_python --license Apache-2.0 --node-name my_node my_package*



```
going to create a new package
package name: my_package
destination directory: /home/user/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['<name> <email>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
node_name: my_node
creating folder ./my_package
creating ./my_package/package.xml
creating source folder
creating folder ./my_package/my_package
creating ./my_package/setup.py
creating ./my_package/setup.cfg
creating folder ./my_package/resource
creating ./my_package/resource/my_package
creating ./my_package/my_package/__init__.py
creating folder ./my_package/test
creating ./my_package/test/test_copyright.py
creating ./my_package/test/test_flake8.py
creating ./my_package/test/test_pep257.py
creating ./my_package/my_package/my_node.py
```

ROS build tools: colcon

1. Go to the src folder:
   - *cd ~/ros2_ws/src*
2. Create a new command
   - *ros2 pkg create --build-type ament_python --license Apache-2.0 <package_name>*
   - *ros2 pkg create --build-type ament_cmake --license Apache-2.0 <package_name>*

   - ament_cmake → C++
   - ament_python → Python
3. *Build our package*
   - *colcon build*
   - *or*
   - *colcon build --packages-select my_package*