ROS PROGRAMMING

Paloma de la Puente

INDUSTRIALES
ETSII | UPM

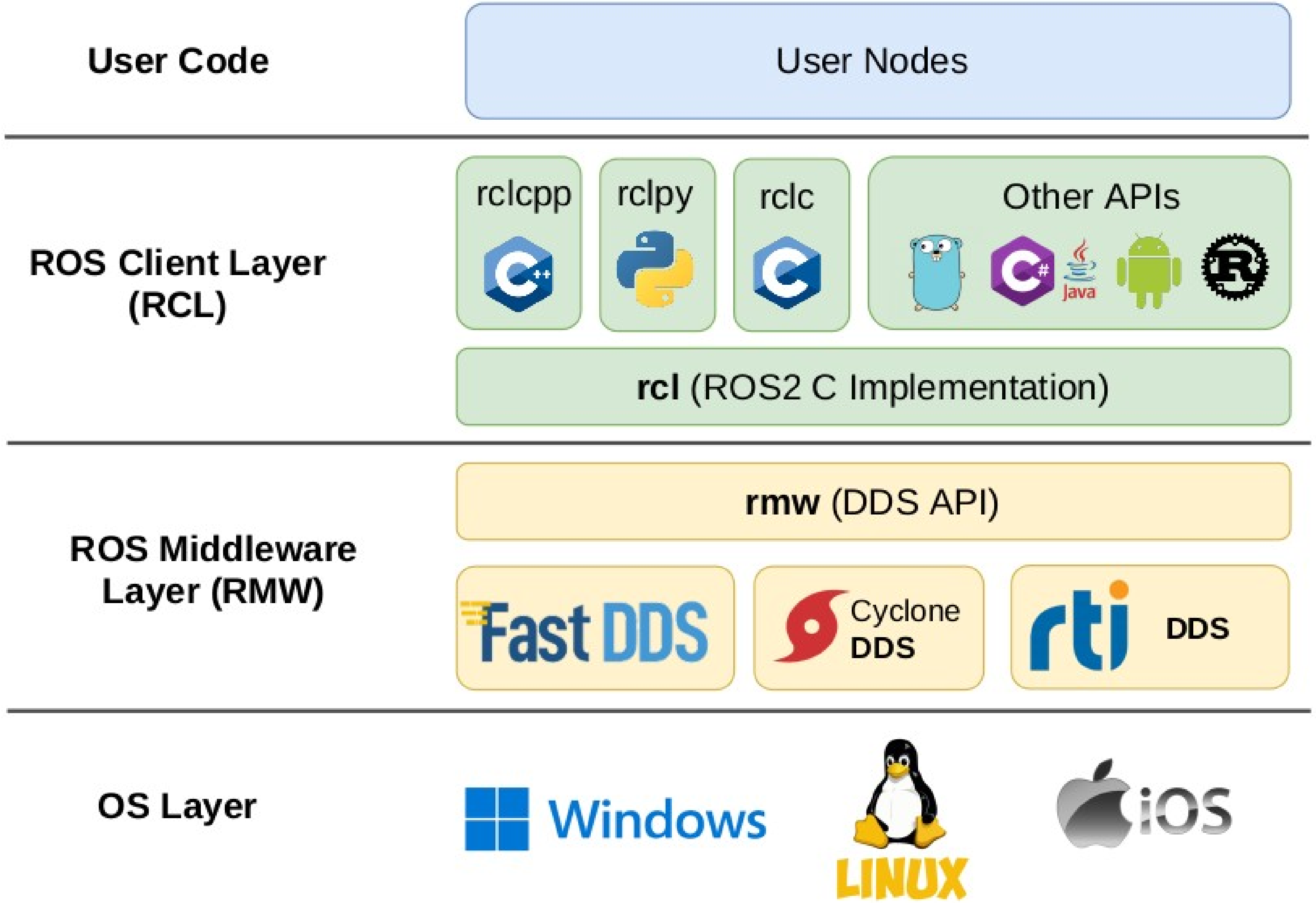
Content

- ROS2 design
- ROS packages
- rclcpp
- ROS publishers and subscribers
- ROS servicess

Introduction

ROS2 DESIGN

ROS2 Design


User Code
User Nodes
ROS Client Layer (RCL)
rclcpp
rclpy
rclc
Other APIs
Java
rcl (ROS2 C Implementation)
ROS Middleware Layer (RMW)
rmw (DDS API)
Fast DDS
Cyclone DDS
rti DDS
OS Layer
Windows
LINUX
iOS

ROS2 Design

- ROS 2 is built on top of DDS/RTPS as its middleware
  - Distributed discovery
  - Serialization
  - QoS Transportation
- DDS is an industry standard, several vendors

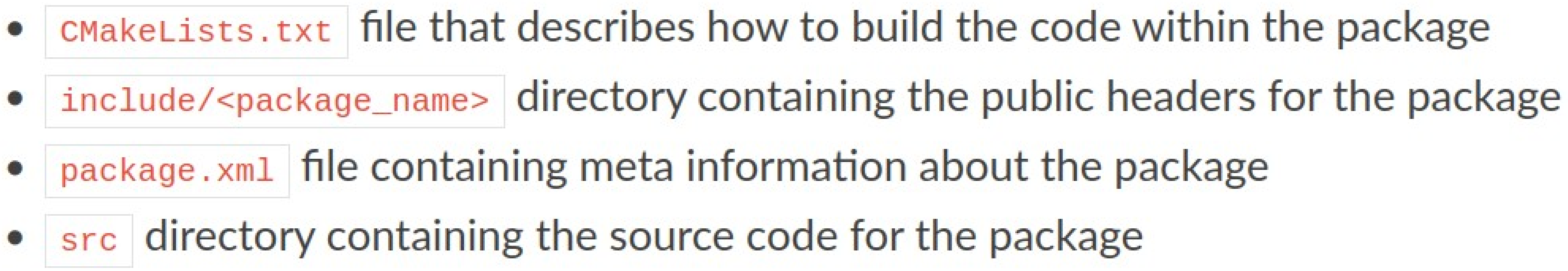
https://docs.ros.org/en/humble/Concepts/Intermediate/About-Different-Middleware-Vendors.html

User code

ROS PACKAGES

ROS Packages

- A package is an organizational unit for your ROS 2 code
- Package contents (C++)



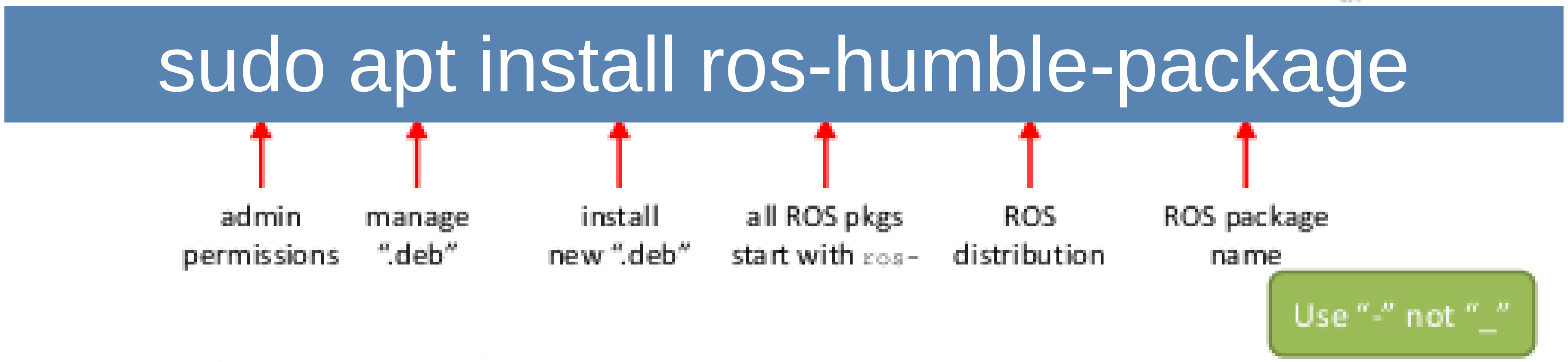
- `CMakeLists.txt` file that describes how to build the code within the package
- `include/<package_name>` directory containing the public headers for the package
- `package.xml` file containing meta information about the package
- `src` directory containing the source code for the package

https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

ROS Packages

- Installing 3rd party packages
  - Debian packages
    - Easy and fast
  - Source code repositories
    - Mostly Github
    - More effort, more control

ROS Packages



```
sudo apt install ros-humble-package
```

admin permissions

manage ".deb"

install new ".deb"

all ROS pkgs start with ros-

ROS distribution

ROS package name

Use "-" not "_"

From RosIN slides

Exercise

1. Create workspace

```
$ mkdir -p ~/turtlebot3_ws/src
```

2. Install turtlebot simulations package and dependencies

```
$ cd ~/turtlebot3_ws/src
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git -b humble-devel
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git -b humble-devel
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git -b humble-devel
$ git clone https://github.com/ROBOTIS-GIT/DynamixelSDK.git -b humble-devel
$ cd ~/turtlebot3_ws && colcon build
```

3. Select the model of your choice (burger or waffle):

```
$ echo "export TURTLEBOT3_MODEL=burger" >> ~/.bashrc
```

You can change it later

4. Run a simulation

```
$ source ~/turtlebot3_ws/install/setup.bash
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

Exercise

4. Check the available topics

```
$ ros2 topic list
```

5. Check the message type & structure for topic /cmd_vel
6. Publish velocity commands to teleoperate the robot

```
$ ros2 topic pub -1 /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

7. Select another topic and receive its messages on the terminal

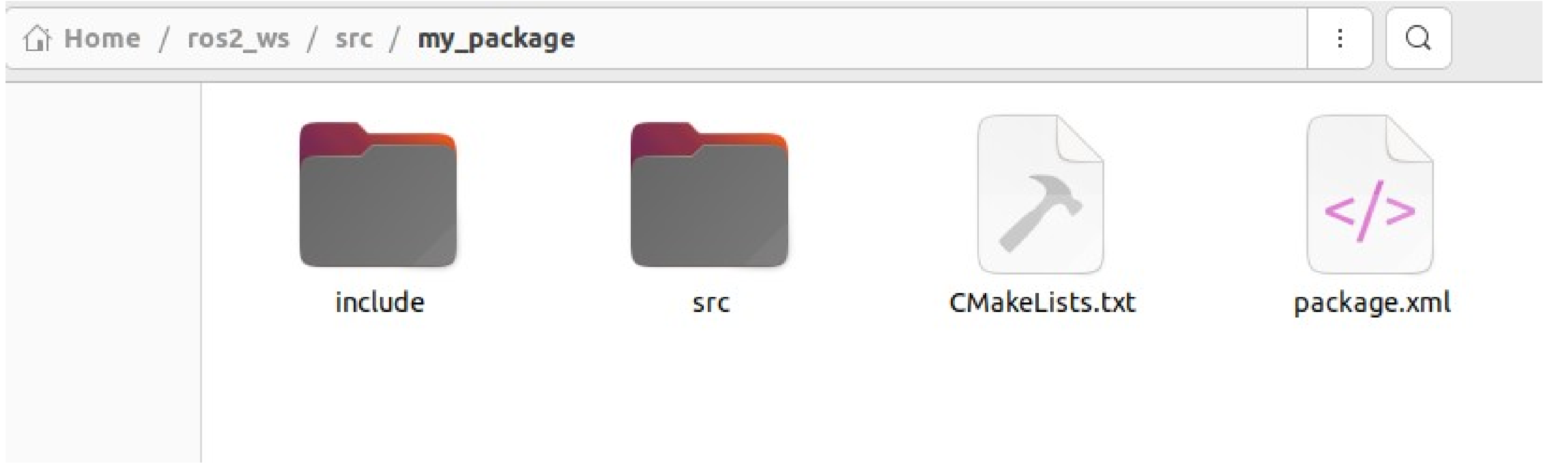
ROS Packages

- **colcon build must ALWAYS be executed from your ros2_ws**
  - DEBUG mode by default
  - Packages can be ignored by adding an empty file with the name COLCON_IGNORE

ROS Packages

- Creating a ROS package

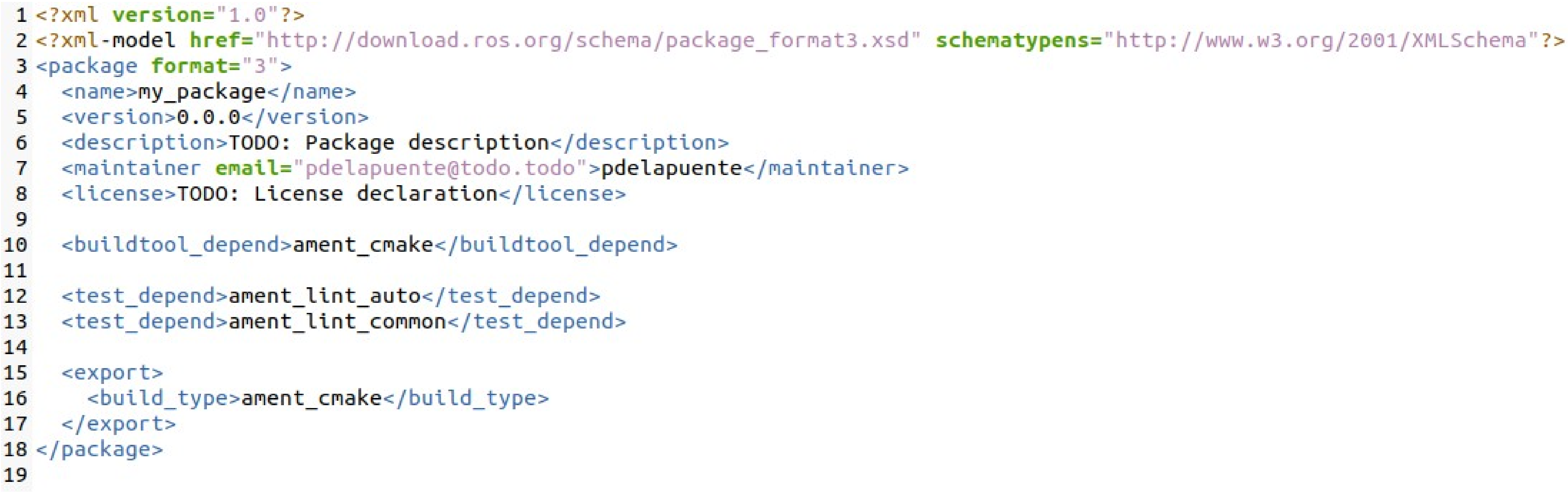
```
$ cd ~/ros2_ws/src
$ ros2 pkg create --build-type ament_cmake --node-name my_node my_package

$ cd ..
$ colcon build --packages-select my_package
```



https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

ROS Packages

- Package.xml



```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>my_package</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="pdelapuente@todo.todo">pdelapuente</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

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

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

```

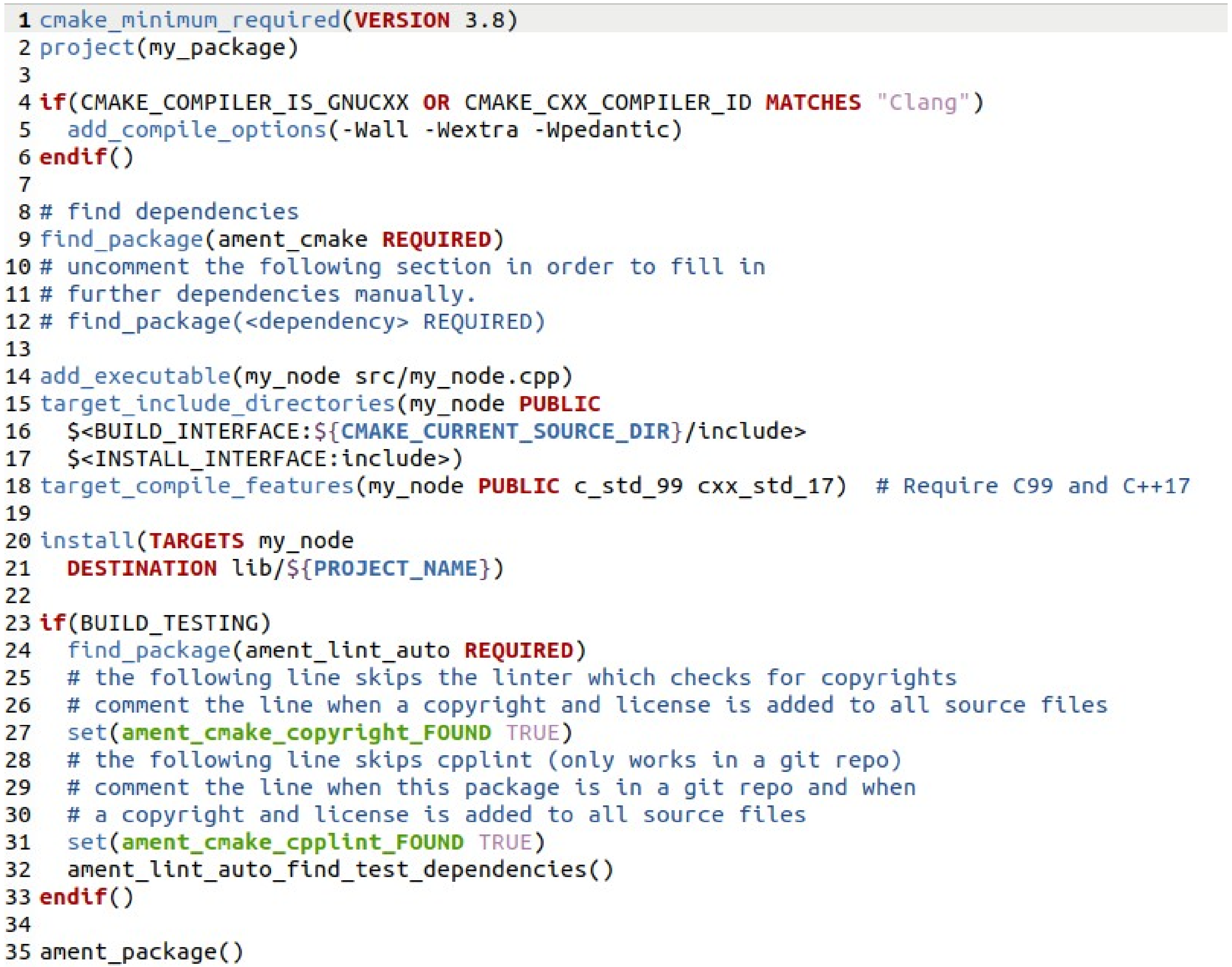
https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

ROS Packages

- Package.xml
  - **<build_depend>** Declares a rosdep key or ROS package name that this package requires at build-time
  - **<build_export_depend>**Declares a rosdep key or ROS package name that this package needs as part of some build interface it exports
  - **<buildtool_depend>** Declares a rosdep key or ROS package name for a tool that is executed during the build process
  - **<buildtool_export_depend>** Declares a rosdep key or ROS package name that this package exports which must be compiled and run on the build system, not the target system
  - **<exec_depend>** Declares a rosdep key or ROS package name that this package needs at execution-time
  - **<depend>** Declares a rosdep key or ROS package name that this package needs for multiple reasons. A <depend> tag is equivalent to specifying <build_depend>, <build_export_depend> and <exec_depend>, all on the same package or key.
  - **<doc_depend>** Declares a rosdep key or ROS package name that your package needs for building its documentation
  - **<test_depend>** Declares a rosdep key or ROS package name that your package needs for running its unit tests

ROS Packages

- CmakeLists.txt



```
cmake_minimum_required(VERSION 3.8)
project(my_package)

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

# find dependencies
find_package(ament_cmake REQUIRED)
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)

add_executable(my_node src/my_node.cpp)
target_include_directories(my_node PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)
target_compile_features(my_node PUBLIC c_std_99 cxx_std_17)  # Require C99 and C++17

install(TARGETS my_node
  DESTINATION lib/${PROJECT_NAME})

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source files
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

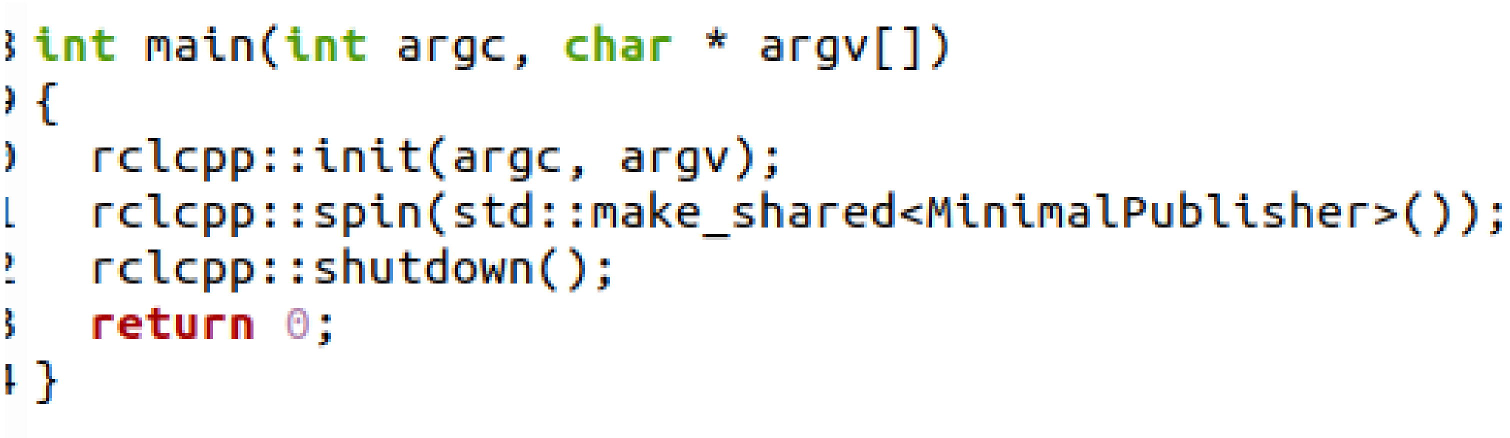
Introduction

RCLCPP

RCLCPP

- Canonical C++ API for interacting with ROS
- Main components
  - Node
  - Publisher and subscription
  - Service server and client
  - Timers and rates
  - Parameters
  - Executors
  - Instrospection of ROS graph
  - Logging → Including macros, e.g. RCLCPP_INFO()

RCLCPP



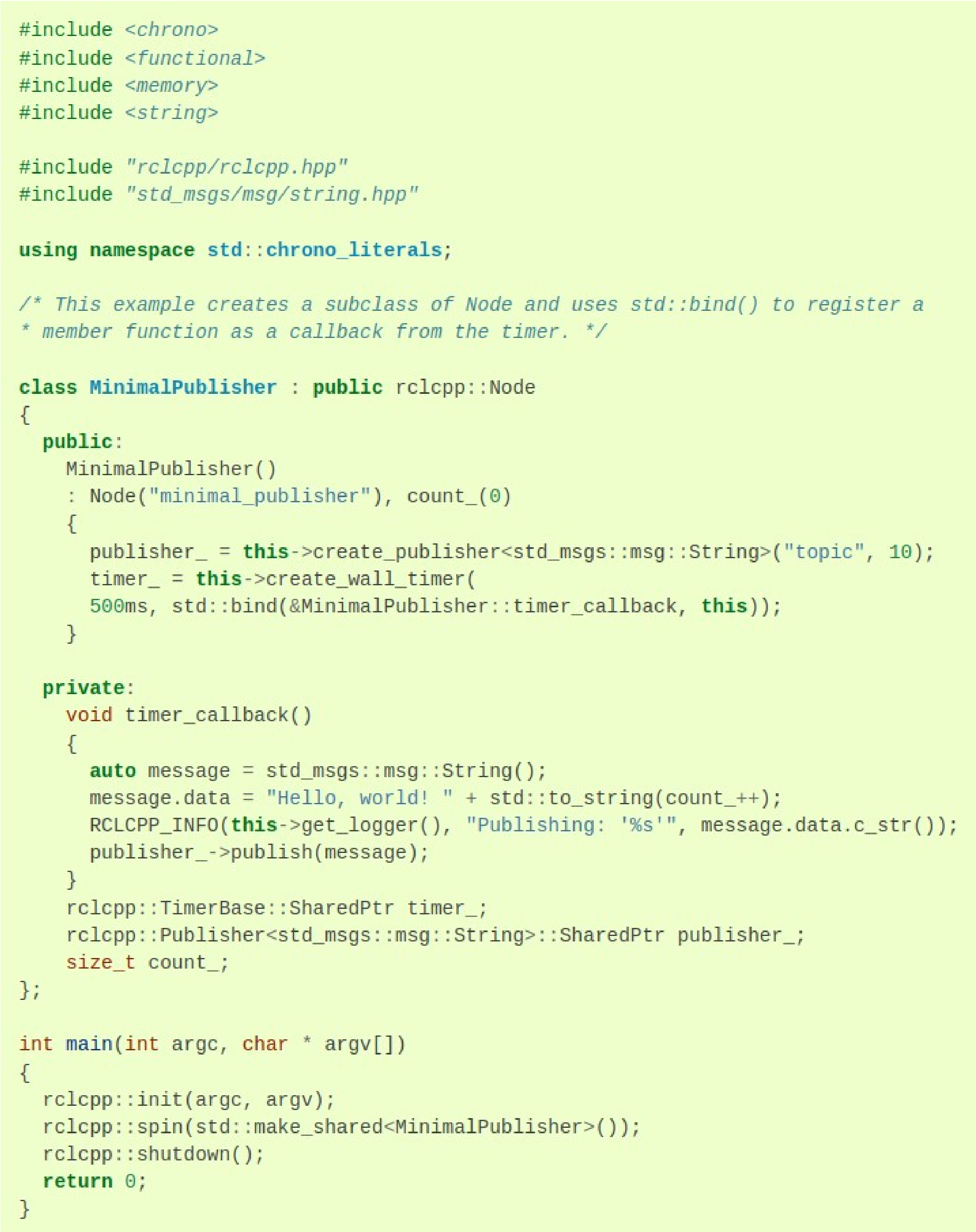
```
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<MinimalPublisher>());
  rclcpp::shutdown();
  return 0;
}
```

Introduction

ROS PUBLISHERS AND SUBSCRIBERS

ROS Publisher

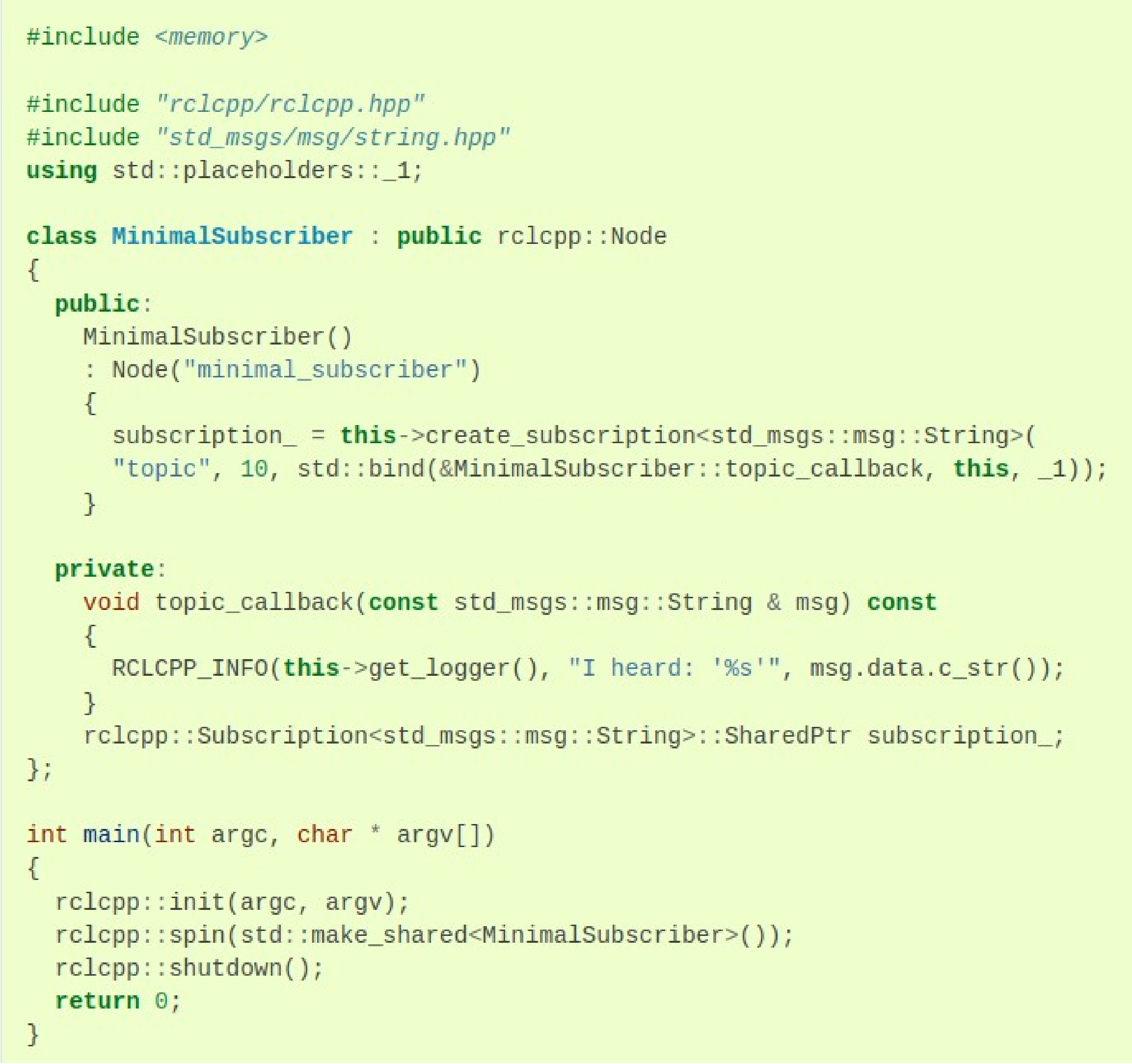
https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html



```cpp
#include <chrono>
#include <functional>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using namespace std::chrono_literals;

/* This example creates a subclass of Node and uses std::bind() to register a
* member function as a callback from the timer. */

class MinimalPublisher : public rclcpp::Node
{
  public:
    MinimalPublisher()
    : Node("minimal_publisher"), count_(0)
    {
      publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10);
      timer_ = this->create_wall_timer(
      500ms, std::bind(&MinimalPublisher::timer_callback, this));
    }

  private:
    void timer_callback()
    {
      auto message = std_msgs::msg::String();
      message.data = "Hello, world! " + std::to_string(count_++);
      RCLCPP_INFO(this->get_logger(), "Publishing: '%s'", message.data.c_str());
      publisher_->publish(message);
    }
    rclcpp::TimerBase::SharedPtr timer_;
    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
    size_t count_;
};

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

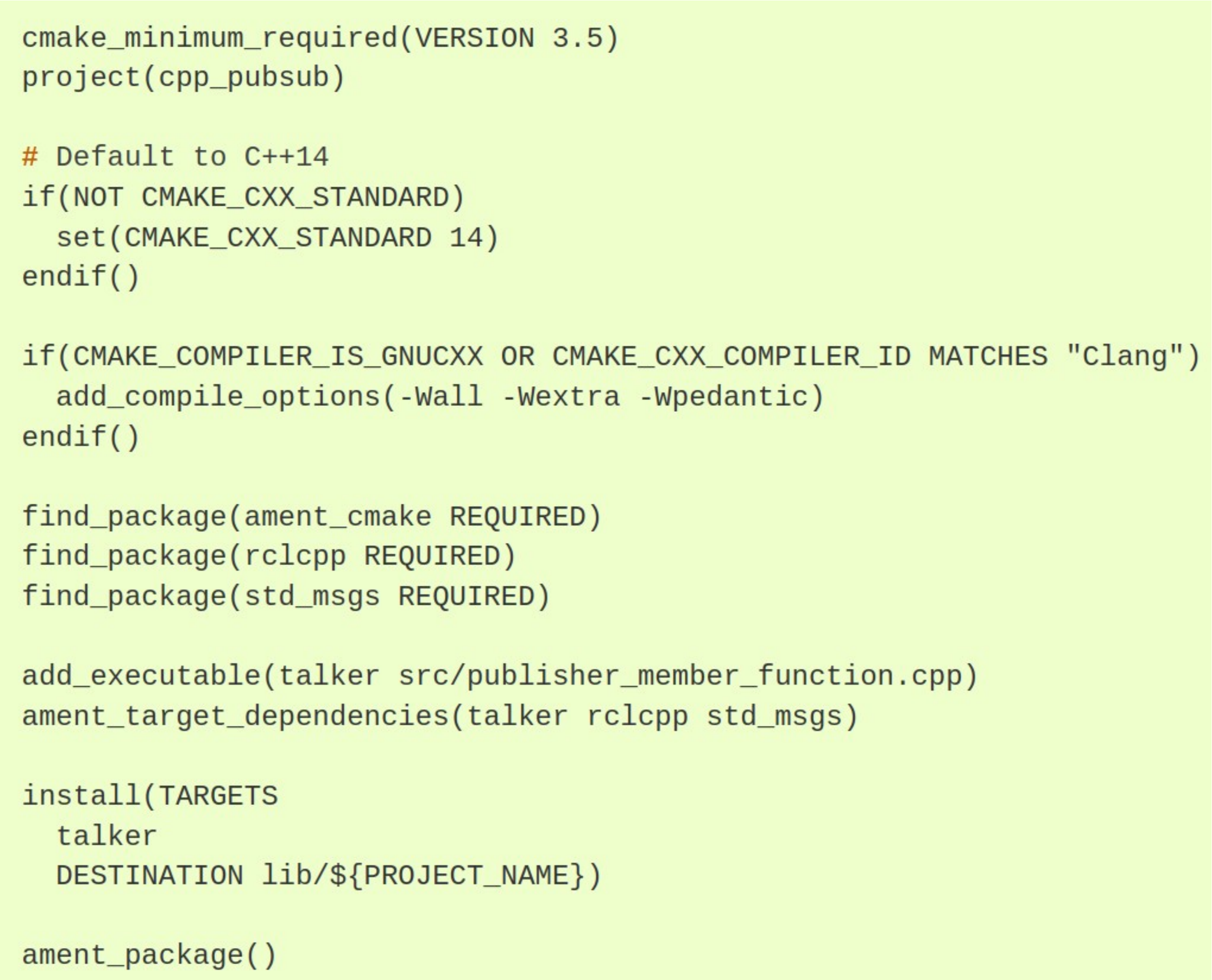
ROS Subscriber

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

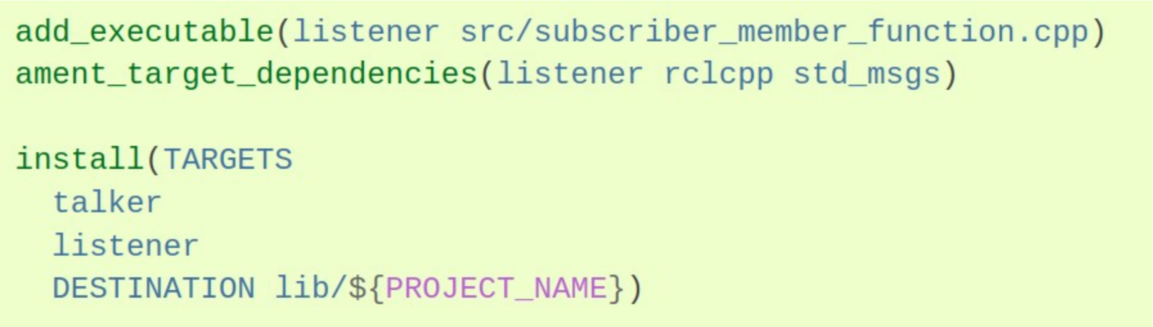


```
#include <memory>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
using std::placeholders::_1;

class MinimalSubscriber : public rclcpp::Node
{
  public:
    MinimalSubscriber()
    : Node("minimal_subscriber")
    {
      subscription_ = this->create_subscription<std_msgs::msg::String>(
      "topic", 10, std::bind(&MinimalSubscriber::topic_callback, this, _1));
    }

  private:
    void topic_callback(const std_msgs::msg::String & msg) const
    {
      RCLCPP_INFO(this->get_logger(), "I heard: '%s'", msg.data.c_str());
    }
    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;
};

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

ROS Publisher & Subscriber



```
cmake_minimum_required(VERSION 3.5)
project(cpp_pubsub)

# 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 "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)

add_executable(talker src/publisher_member_function.cpp)
ament_target_dependencies(talker rclcpp std_msgs)

install(TARGETS
  talker
  DESTINATION lib/${PROJECT_NAME})

ament_package()
```



```
add_executable(listener src/subscriber_member_function.cpp)
ament_target_dependencies(listener rclcpp std_msgs)

install(TARGETS
  talker
  listener
  DESTINATION lib/${PROJECT_NAME})
```

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

Exercise

1. Create a new package to move the simulated turtlebot

   Why do we need these dependencies?

   ```
   $ ros2 pkg create robot_controller --dependencies  rclcpp nav_msgs geometry_msgs
   --build-type ament_cmake
   ```

2. Create a subscriber to receive and print odometry data from the turtlebot simulation
3. Create a publisher to move the turtlebot by sending random commands to topic /cmd_vel
4. Check the publisher frequency
5. Chek your nodes info
6. (Optional) Use the odometry data to move the robot towards a target point

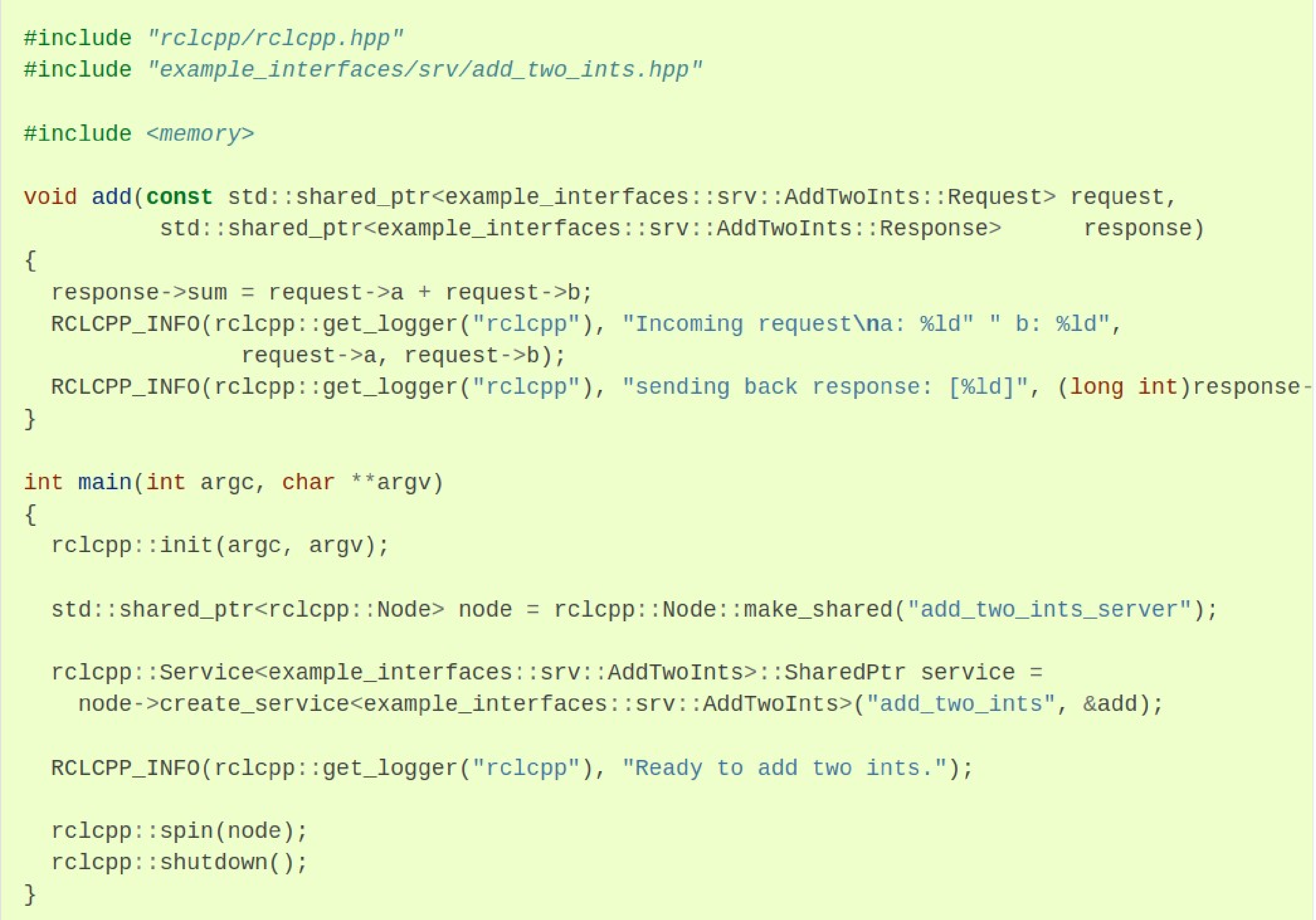
Introduction

ROS SERVICES

ROS Services

$ ros2 pkg create --build-type ament_cmake cpp_srvcli --dependencies rclcpp example_interfaces

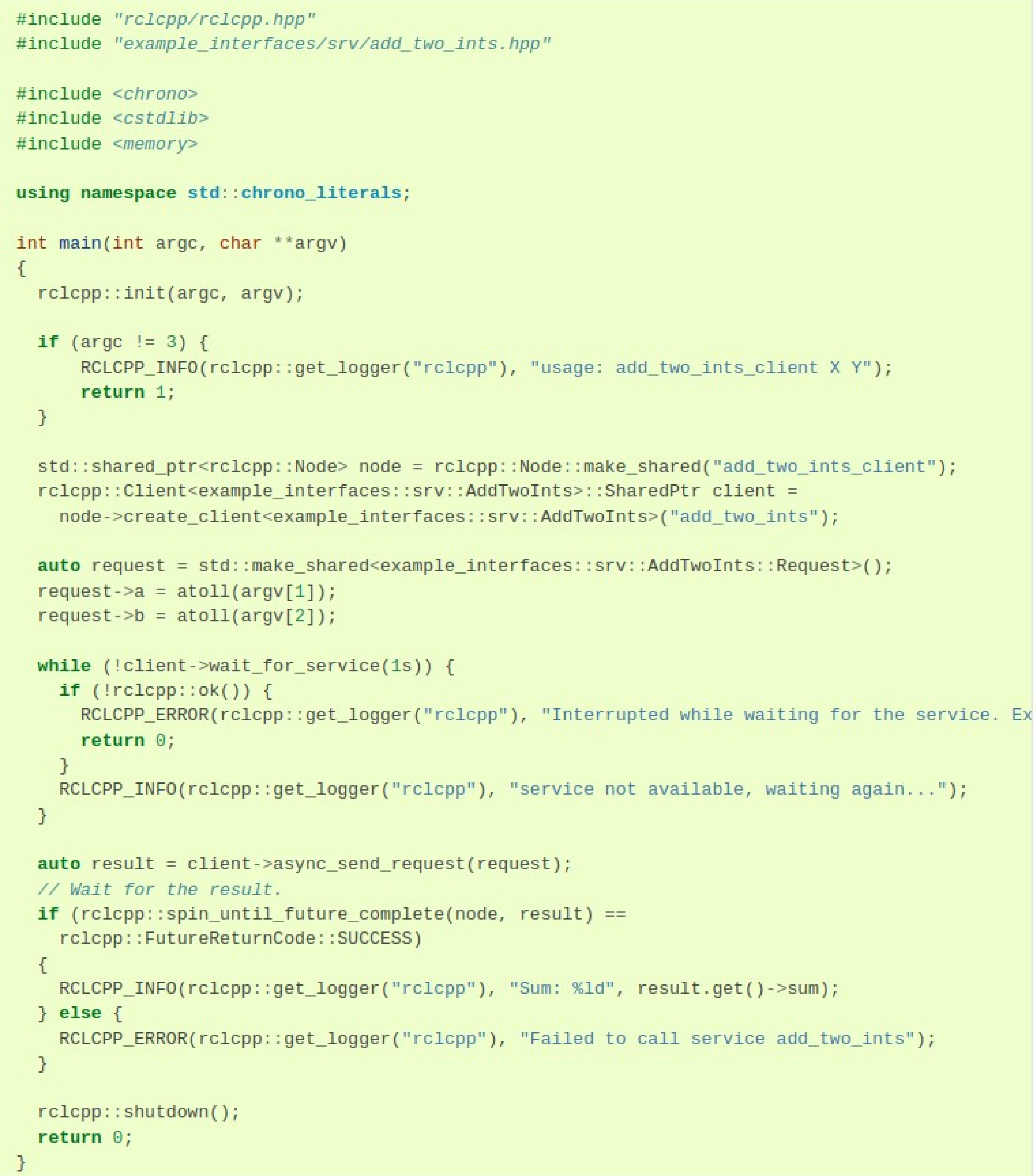
https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Service-And-Client.html



```
#include "rclcpp/rclcpp.hpp"
#include "example_interfaces/srv/add_two_ints.hpp"

#include <memory>

void add(const std::shared_ptr<example_interfaces::srv::AddTwoInts::Request> request,
          std::shared_ptr<example_interfaces::srv::AddTwoInts::Response>      response)
{
  response->sum = request->a + request->b;
  RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "Incoming request\na: %ld" " b: %ld",
                request->a, request->b);
  RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "sending back response: [%ld]", (long int)response-
}

int main(int argc, char **argv)
{
  rclcpp::init(argc, argv);

  std::shared_ptr<rclcpp::Node> node = rclcpp::Node::make_shared("add_two_ints_server");

  rclcpp::Service<example_interfaces::srv::AddTwoInts>::SharedPtr service =
    node->create_service<example_interfaces::srv::AddTwoInts>("add_two_ints", &add);

  RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "Ready to add two ints.");

  rclcpp::spin(node);
  rclcpp::shutdown();
}
```

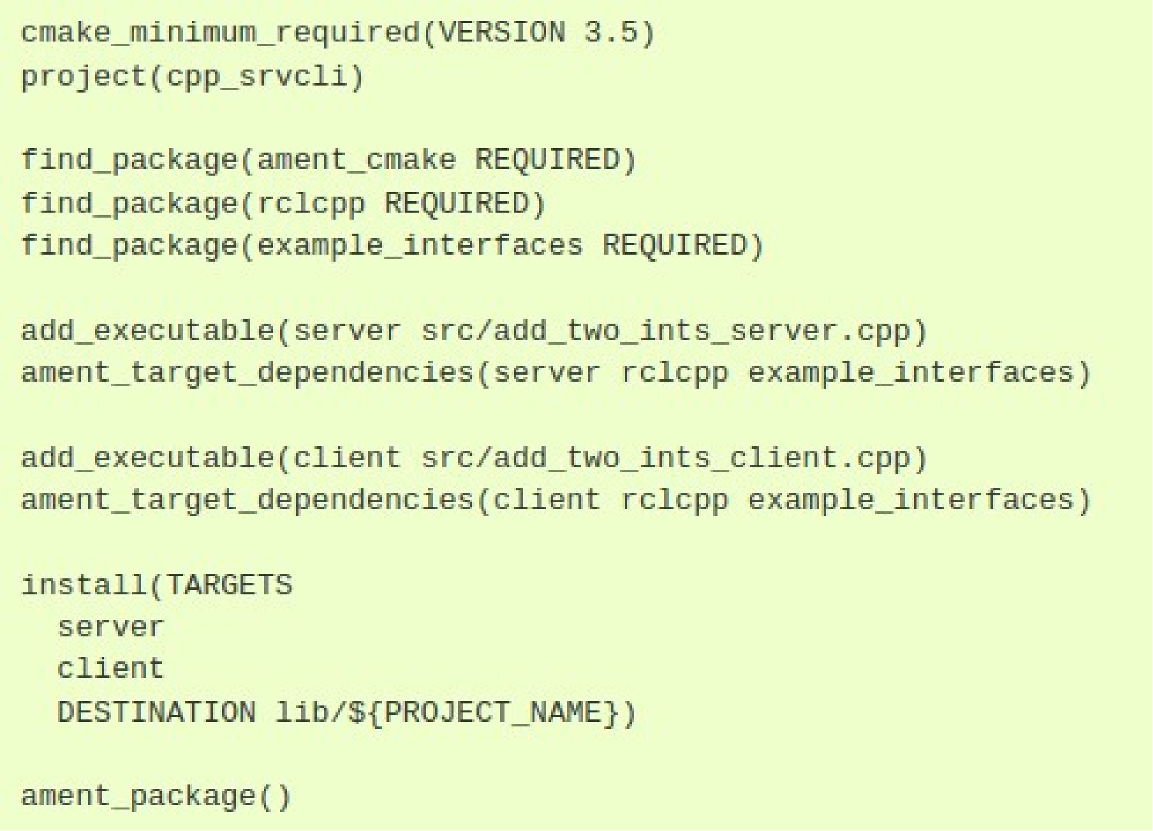
ROS Services

https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Service-And-Client.html



```cpp
#include "rclcpp/rclcpp.hpp"
#include "example_interfaces/srv/add_two_ints.hpp"

#include <chrono>
#include <cstdlib>
#include <memory>

using namespace std::chrono_literals;

int main(int argc, char **argv)
{
  rclcpp::init(argc, argv);

  if (argc != 3) {
      RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "usage: add_two_ints_client X Y");
      return 1;
  }

  std::shared_ptr<rclcpp::Node> node = rclcpp::Node::make_shared("add_two_ints_client");
  rclcpp::Client<example_interfaces::srv::AddTwoInts>::SharedPtr client =
    node->create_client<example_interfaces::srv::AddTwoInts>("add_two_ints");

  auto request = std::make_shared<example_interfaces::srv::AddTwoInts::Request>();
  request->a = atoll(argv[1]);
  request->b = atoll(argv[2]);

  while (!client->wait_for_service(1s)) {
    if (!rclcpp::ok()) {
      RCLCPP_ERROR(rclcpp::get_logger("rclcpp"), "Interrupted while waiting for the service. Ex
      return 0;
    }
    RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "service not available, waiting again...");
  }

  auto result = client->async_send_request(request);
  // Wait for the result.
  if (rclcpp::spin_until_future_complete(node, result) ==
    rclcpp::FutureReturnCode::SUCCESS)
  {
    RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "Sum: %ld", result.get()->sum);
  } else {
    RCLCPP_ERROR(rclcpp::get_logger("rclcpp"), "Failed to call service add_two_ints");
  }

  rclcpp::shutdown();
  return 0;
}
```

ROS Services



```
cmake_minimum_required(VERSION 3.5)
project(cpp_srvcli)

find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(example_interfaces REQUIRED)

add_executable(server src/add_two_ints_server.cpp)
ament_target_dependencies(server rclcpp example_interfaces)

add_executable(client src/add_two_ints_client.cpp)
ament_target_dependencies(client rclcpp example_interfaces)

install(TARGETS
  server
  client
  DESTINATION lib/${PROJECT_NAME})

ament_package()
```

Exercise

1. Kill the service and execute the client. What happens?
2. Modify your robot_controller package, including a service to switch between two controller modes: sending a linear velocity command vs sending a rotational velocity command

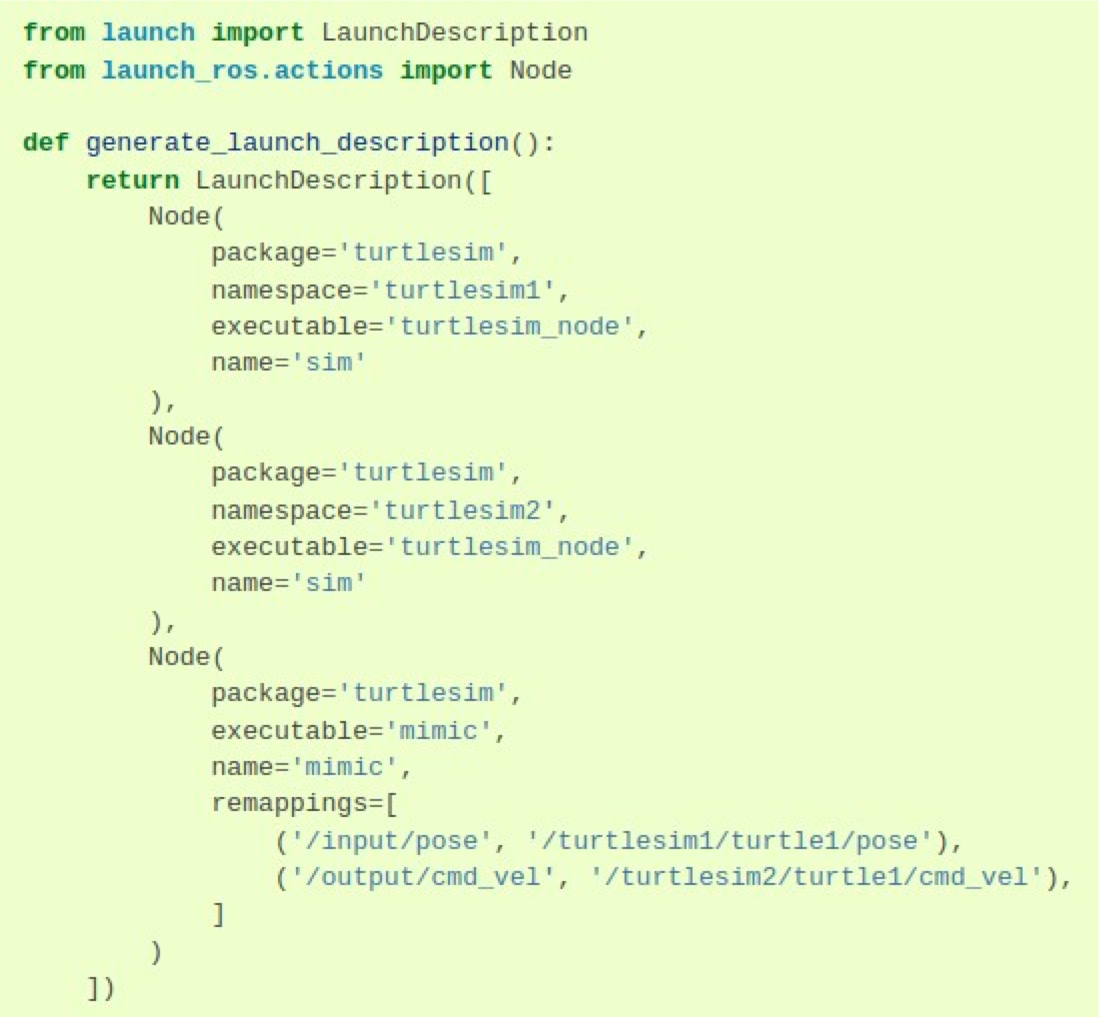
Launch files

- Three format alternatives
  - Python
  - XML
  - Yaml
- Functions
  - Start/stop different nodes
  - Parameters
  - ROS2 conventions and configurations

https://docs.ros.org/en/foxy/Tutorials/Intermediate/Launch/Creating-Launch-Files.html#

Launch files

- Example



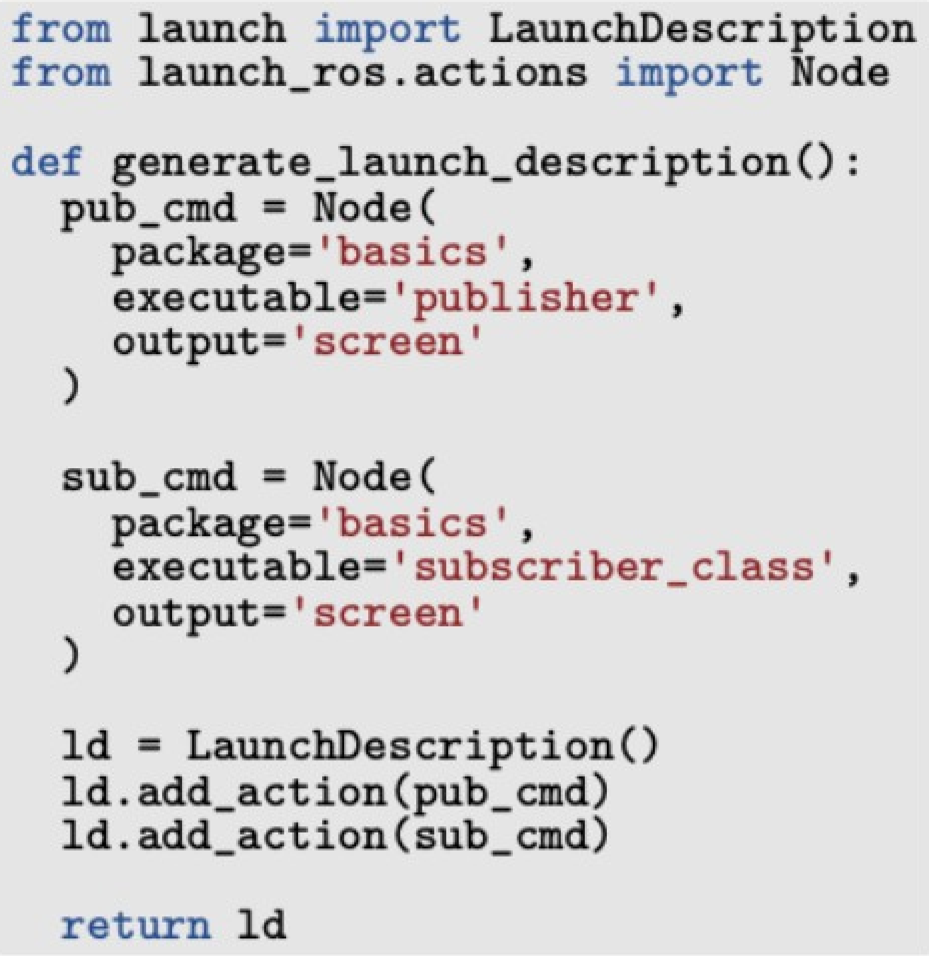
```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            namespace='turtlesim2',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        )
    ])
```

<exec_depend>ros2launch</exec_depend>

https://docs.ros.org/en/foxy/Tutorials/Intermediate/Launch/Creating-Launch-Files.html#

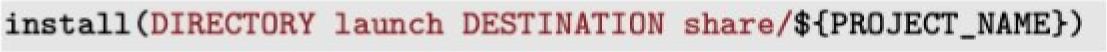
Launch files

- Example



```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
  pub_cmd = Node(
    package='basics',
    executable='publisher',
    output='screen'
  )

  sub_cmd = Node(
    package='basics',
    executable='subscriber_class',
    output='screen'
  )

  ld = LaunchDescription()
  ld.add_action(pub_cmd)
  ld.add_action(sub_cmd)

  return ld
```

By: Francisco Martín Rico, 2023



```
install(DIRECTORY launch DESTINATION share/${PROJECT_NAME})
```

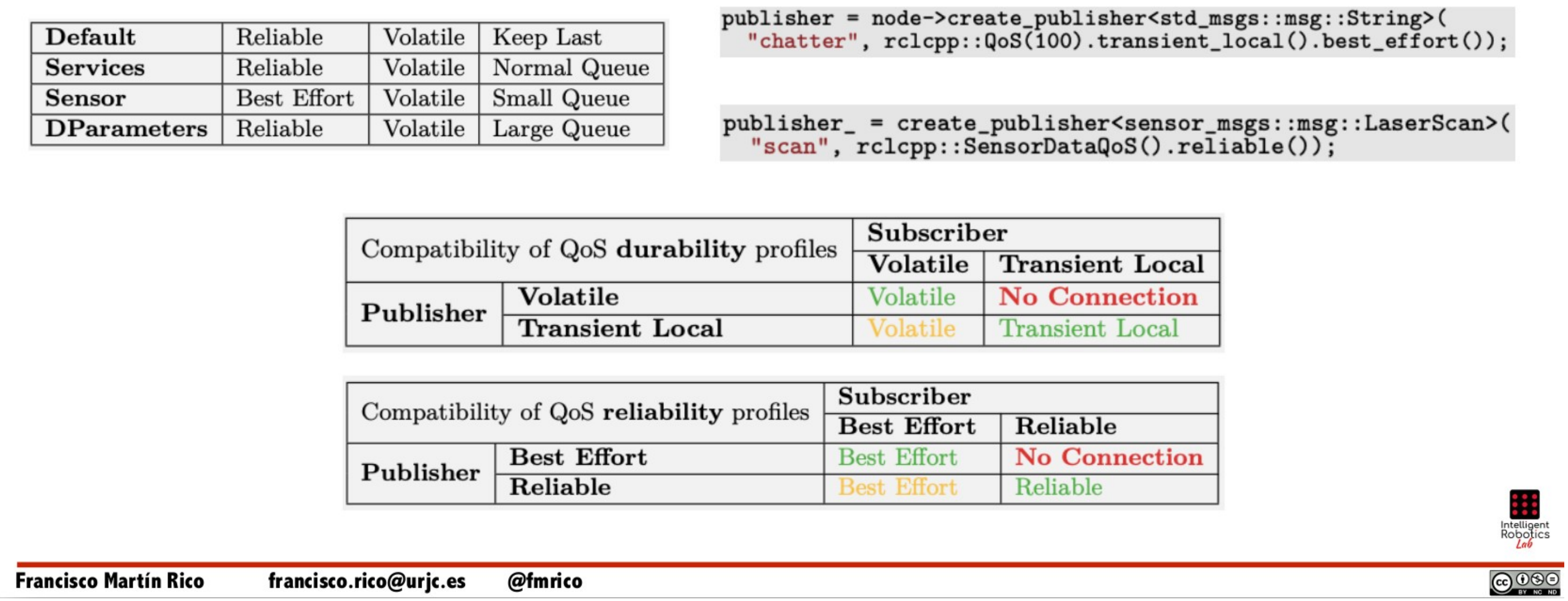
Exercise

1. Create a launch file to launch your robot_controller nodes

ROS2 QoS

- History
  - Keep last: only store up to N samples, configurable via the queue depth option
  - Keep all: store all samples
- Depth
  - Queue size: only honored if the “history” policy was set to “keep last”.
- Reliability
  - Best effort: attempt to deliver samples
  - Reliable: guarantee that samples are delivered, may retry multiple times.
- Durability
  - Transient local: the publisher becomes responsible for persisting samples
  - Volatile: no attempt is made to persist samples.
- Deadline
  - Duration: the expected maximum amount of time between subsequent messages
- Lifespan
  - Duration: the maximum amount of time between the publishing and the reception
- Liveliness
  - Automatic:
  - Manual by topic
- Lease Duration
  - Duration: the maximum period of time a publisher has to indicate that it is alive

ROS2 QoS



Default	Reliable	Volatile	Keep Last
Services	Reliable	Volatile	Normal Queue
Sensor	Best Effort	Volatile	Small Queue
DParameters	Reliable	Volatile	Large Queue

```
publisher = node->create_publisher<std_msgs::msg::String>(
  "chatter", rclcpp::QoS(100).transient_local().best_effort());
```

```
publisher_ = create_publisher<sensor_msgs::msg::LaserScan>(
  "scan", rclcpp::SensorDataQoS().reliable());
```

Compatibility of QoS **durability** profiles		**Subscriber**	
		Volatile	**Transient Local**
Publisher	**Volatile**	Volatile	**No Connection**
	Transient Local	Volatile	Transient Local

Compatibility of QoS **reliability** profiles		**Subscriber**	
		Best Effort	**Reliable**
Publisher	**Best Effort**	Best Effort	**No Connection**
	Reliable	Best Effort	Reliable

https://docs.ros.org/en/rolling/Concepts/Intermediate/About-Quality-of-Service-Settings.html

ROS PROGRAMMING

End of lesson