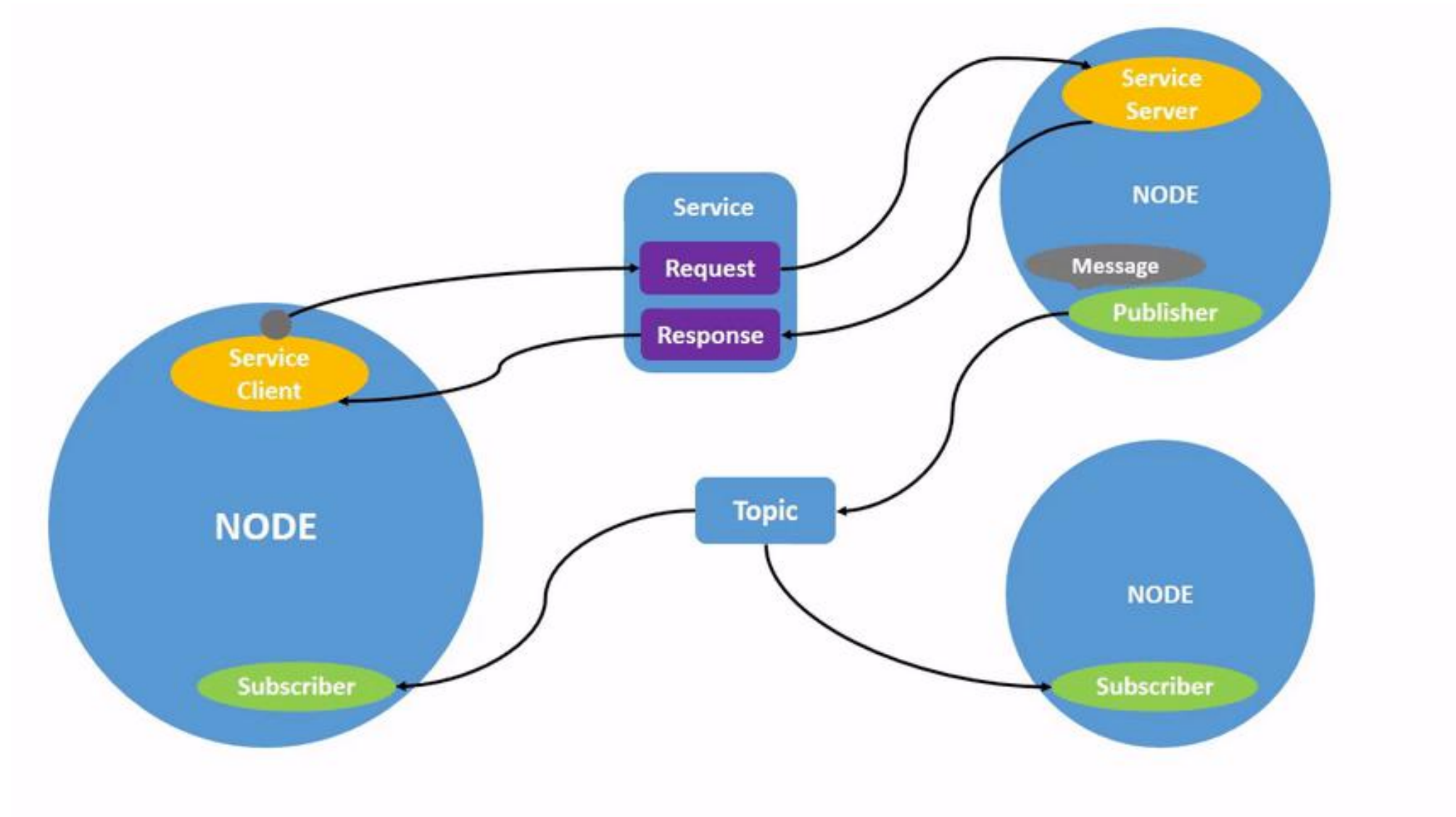# Lab3

# Content

**1** **ROS2 Node**

**2** **ROS2 Topic**

**3** **ROS2 Service**

## Create first node (C++)

```
$ cd ~/ros2_ws/src/
$ ros2 pkg create c_pkg --node-name listener --dependencies rclcpp std_msgs --build-type ament_cmake
```



```
pi@pi-virtual-machine:~/ros2_ws/src/c_pkg$ tree
├── CMakeLists.txt
├── include
│   └── c_pkg
├── package.xml
└── src
    └── litsener.cpp

3 directories, 3 files
```



```cpp
1 #include <cstdio>
2
3 int main(int argc, char ** argv)
4 {
5   (void) argc;
6   (void) argv;
7
8   printf("hello world c_pkg package\n");
9   return 0;
10 }
```

```cmake
1 cmake_minimum_required(VERSION 3.8)
2 project(c_pkg)
3
4 if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
5   add_compile_options(-Wall -Wextra -Wpedantic)
6 endif()
7
8 # find dependencies
9 find_package(ament_cmake REQUIRED)
10 find_package(rclcpp REQUIRED)
11 find_package(std_msgs REQUIRED)
12
13 add_executable(litsener src/litsener.cpp)
14 target_include_directories(litsener PUBLIC
15   $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
16   $<INSTALL_INTERFACE:include>)
17 target_compile_features(litsener PUBLIC c_std_99 cxx_std_17)  # Require C99 and C++17
18 ament_target_dependencies(
19   litsener
20   "rclcpp"
21   "std_msgs"
22 )
23
24 install(TARGETS litsener
25   DESTINATION lib/${PROJECT_NAME})
26
27 if(BUILD_TESTING)
28   find_package(ament_lint_auto REQUIRED)
29   # the following line skips the linter which checks for copyrights
30   # comment the line when a copyright and license is added to all source files
31   set(ament_cmake_copyright_FOUND TRUE)
32   # the following line skips cpplint (only works in a git repo)
33   # comment the line when this package is in a git repo and when
34   # a copyright and license is added to all source files
35   set(ament_cmake_cpplint_FOUND TRUE)
36   ament_lint_auto_find_test_dependencies()
37 endif()
```

*CMakeLists.txt — ~/ros2_ws/src/c_pkg*

```xml
1 <?xml version="1.0"?>
2 <?xml-model href="http://download.ros.org/schema/package_format3.xsd"
  schematypens="http://www.w3.org/2001/XMLSchema"?>
3 <package format="3">
4   <name>c_pkg</name>
5   <version>0.0.0</version>
6   <description>TODO: Package description</description>
7   <maintainer email="pi@todo.todo">pi</maintainer>
8   <license>TODO: License declaration</license>
9
10   <buildtool_depend>ament_cmake</buildtool_depend>
11
12   <depend>rclcpp</depend>
13   <depend>std_msgs</depend>
14
15   <test_depend>ament_lint_auto</test_depend>
16   <test_depend>ament_lint_common</test_depend>
17
18   <export>
19     <build_type>ament_cmake</build_type>
20   </export>
21 </package>
```

*package.xml — ~/ros2_ws/src/c_pkg*

```
$ cd ~/ros2_ws/src/c_pkg/src/
$ touch talker.cpp
```



```
                                            CMakeLists.txt
 Open          [+]                         ~/ros2_ws/src/c_pkg
 1 cmake_minimum_required(VERSION 3.8)
 2 project(c_pkg)
 3
 4 if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
 5   add_compile_options(-Wall -Wextra -Wpedantic)
 6 endif()
 7
 8 # find dependencies
 9 find_package(ament_cmake REQUIRED)
 0 find_package(rclcpp REQUIRED)
 1 find_package(std_msgs REQUIRED)
 2
 3 add_executable(listener src/listener.cpp)
 4 add_executable(talker src/talker.cpp)
 5 target_include_directories(listener PUBLIC
 6   $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
 7   $<INSTALL_INTERFACE:include>)
 8 target_include_directories(talker PUBLIC
 9   $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
 0   $<INSTALL_INTERFACE:include>)
 1 target_compile_features(listener PUBLIC c_std_99 cxx_std_17)  # Require C99 and C++17
 2 ament_target_dependencies(
 3   listener
 4   rclcpp
 5   std_msgs
 6 )
 7 ament_target_dependencies(
 8   talker
 9   rclcpp
 0   std_msgs
 1 )
 2 install(TARGETS listener
 3   DESTINATION lib/${PROJECT_NAME})
 4 install(TARGETS talker
 5   DESTINATION lib/${PROJECT_NAME})
 6
 7
 8 if(BUILD_TESTING)
 9   find_package(ament_lint_auto REQUIRED)
```

## Create first node (Python)

$ cd ~/ros2_ws/src/
$ ros2 pkg create py_pkg --node-name talker --dependencies rclpy std_msgs --build-type ament_python



```
pi@pi-virtual-machine:~/ros2_ws/src/py_pkg$ tree

├── package.xml
├── py_pkg
│   ├── __init__.py
│   └── talker.py
├── resource
│   └── py_pkg
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py

3 directories, 9 files
```
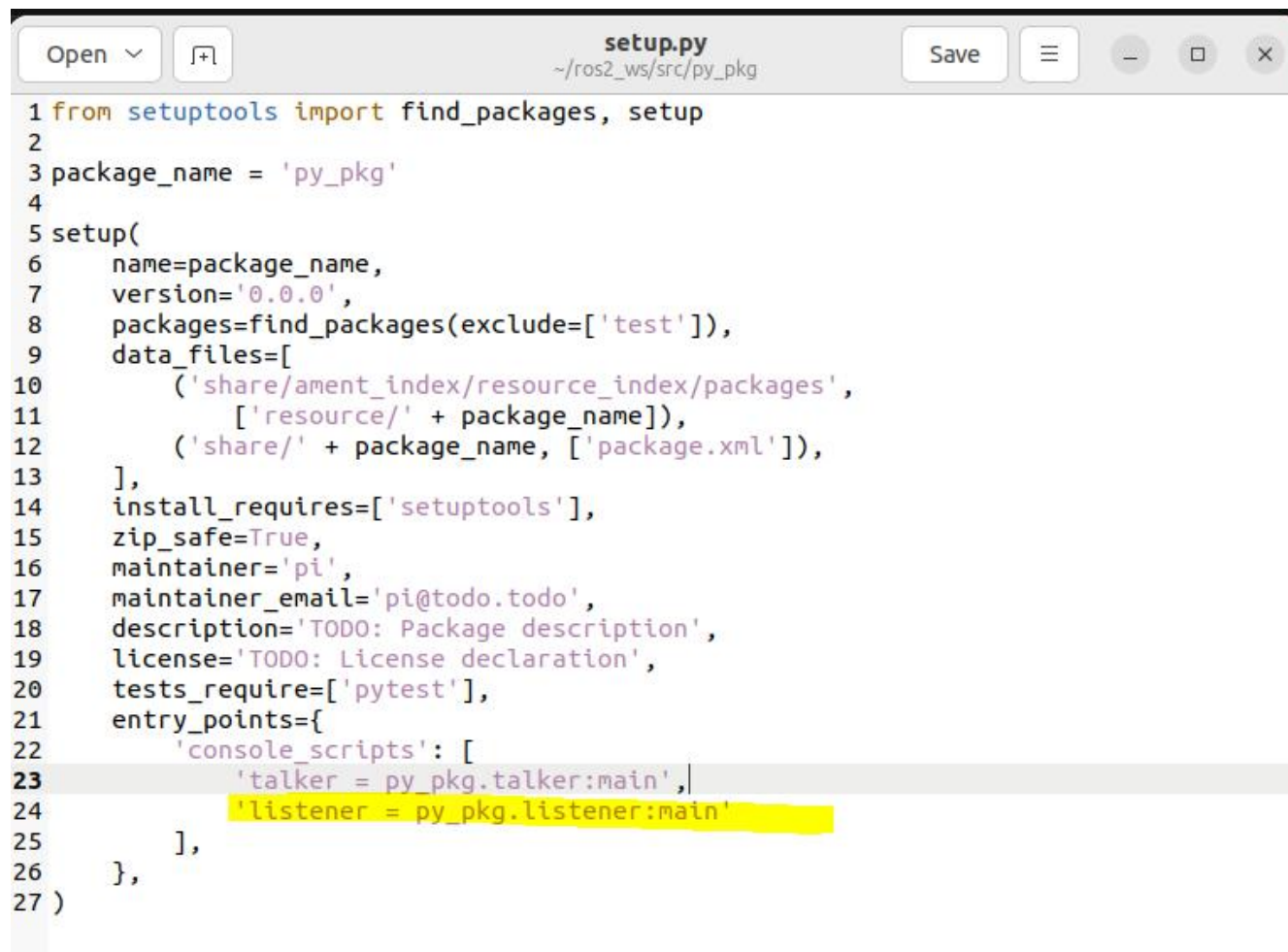


```
Open ∨  ⊞                    talker.py
                             ~/ros2_ws/src/py_pkg/py_pkg
1 def main():
2     print('Hi from py_pkg.')
3
4
5 if __name__ == '__main__':
6     main()
```

```
Open  ∨    ⌐+⌐                    setup.py                    Sav
                              ~/ros2_ws/src/py_pkg
 1 from setuptools import find_packages, setup
 2
 3 package_name = 'py_pkg'
 4
 5 setup(
 6     name=package_name,
 7     version='0.0.0',
 8     packages=find_packages(exclude=['test']),
 9     data_files=[
10         ('share/ament_index/resource_index/packages',
11             ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='pi',
17     maintainer_email='pi@todo.todo',
18     description='TODO: Package description',
19     license='TODO: License declaration',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'talker = py_pkg.talker:main'
24         ],
25     },
26 )
```
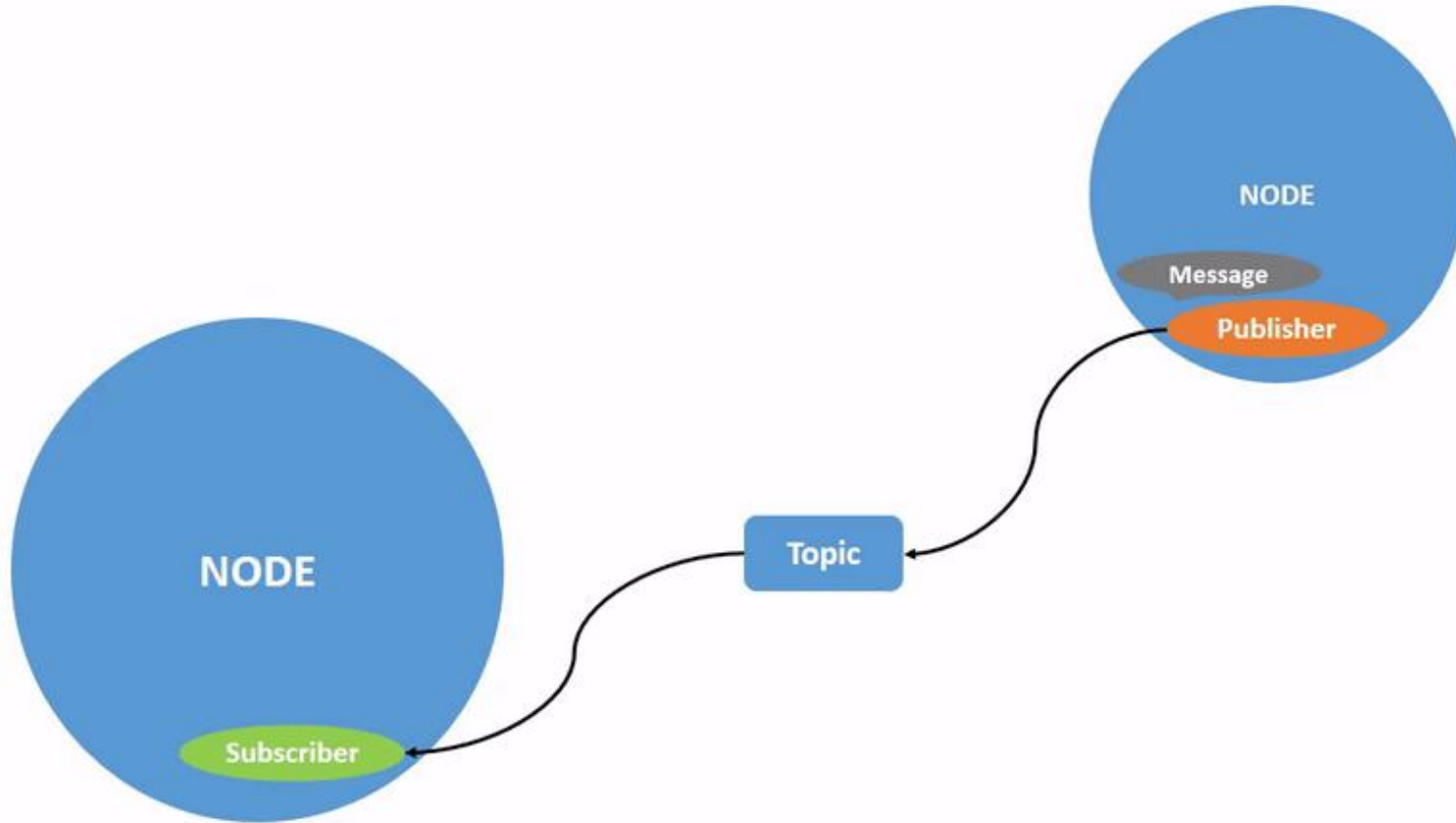
```
$ cd ~/ros2_ws/src/py_pkg/py_pkg/
$ touch listener.py
```



```python
from setuptools import find_packages, setup

package_name = 'py_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='pi',
    maintainer_email='pi@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'talker = py_pkg.talker:main',
            'listener = py_pkg.listener:main'
        ],
    },
)
```

9

Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.
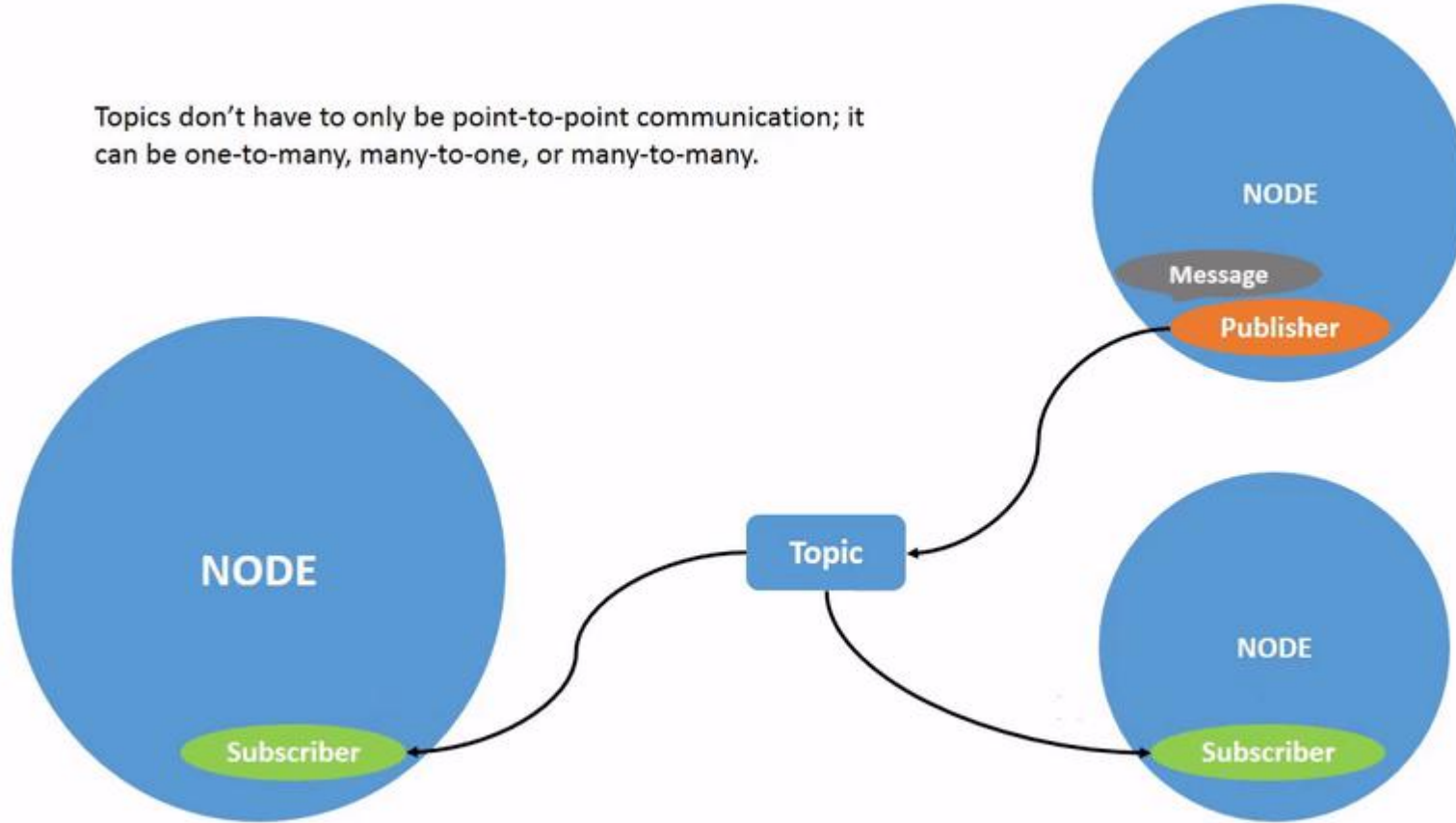
## Create first publisher (C++)

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

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
using namespace std::chrono_literals;

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

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

## Create first publisher (Python)

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

def main():
    rclpy.init(args=None)
    node = Talker()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

```
class Talker(Node):
    def __init__(self):
        super().__init__("Talker")
        self.publisher_ =
self.create_publisher(String,"topic",10)
        self.count = 0
        self.timer_ =
self.create_timer(0.5,self.timer_cb)
    def timer_cb(self):
        message = String()
        message.data = "Hello,
world!"+str(self.count)

self.get_logger().info("Publishing"+message.data)
        self.publisher_.publish(message)
        self.count += 1
```

## Create first subscriber (C++)

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

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

```cpp
class Listener : public rclcpp::Node
{
public:
    Listener() : Node("listener")
    {
        subscription_ = this->create_subscription<std_msgs::msg::String>(
            "topic", 10, std::bind(&Listener::callback, this,
std::placeholders::_1));
    }

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

    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr
subscription_;
};
```

## Create first publisher (Python)

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

def main():
    rclpy.init(args=None)
    node = Talker()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

```python
class Talker(Node):
    def __init__(self):
        super().__init__("Talker")
        self.publisher_ =
self.create_publisher(String,"topic",10)
        self.count = 0
        self.timer_ =
self.create_timer(0.5,self.timer_cb)
    def timer_cb(self):
        message = String()
        message.data = "Hello,
world!"+str(self.count)

self.get_logger().info("Publishing:"+message.data)
        self.publisher_.publish(message)
        self.count += 1
```

## Create first Subcriber (Python)

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String


def main():
    rclpy.init(args=None)
    node = Listener()
    rclpy.spin(node)


if __name__ == '__main__':
    main()
```

```python
class Listener(Node):
    def __init__(self):
        super().__init__('listener')
        self.subscriber_ =
self.create_subscription(String,'topic',self.subscriber_cb,10)

    def subscriber_cb(self,msg):
        self.get_logger().info("I heard:"+msg.data)
```

```
pi@pi-virtual-machine:~$ ros2 topic --help
usage: ros2 topic [-h] [--include-hidden-topics]
                   Call `ros2 topic <command> -h` for more detailed usage. ...

Various topic related sub-commands

options:
  -h, --help              show this help message and exit
  --include-hidden-topics
                          Consider hidden topics as well

Commands:
  bw     Display bandwidth used by topic
  delay  Display delay of topic from timestamp in header
  echo   Output messages from a topic
  find   Output a list of available topics of a given type
  hz     Print the average publishing rate to screen
  info   Print information about a topic
  list   Output a list of available topics
  pub    Publish a message to a topic
  type   Print a topic's type

  Call `ros2 topic <command> -h` for more detailed usage.
```
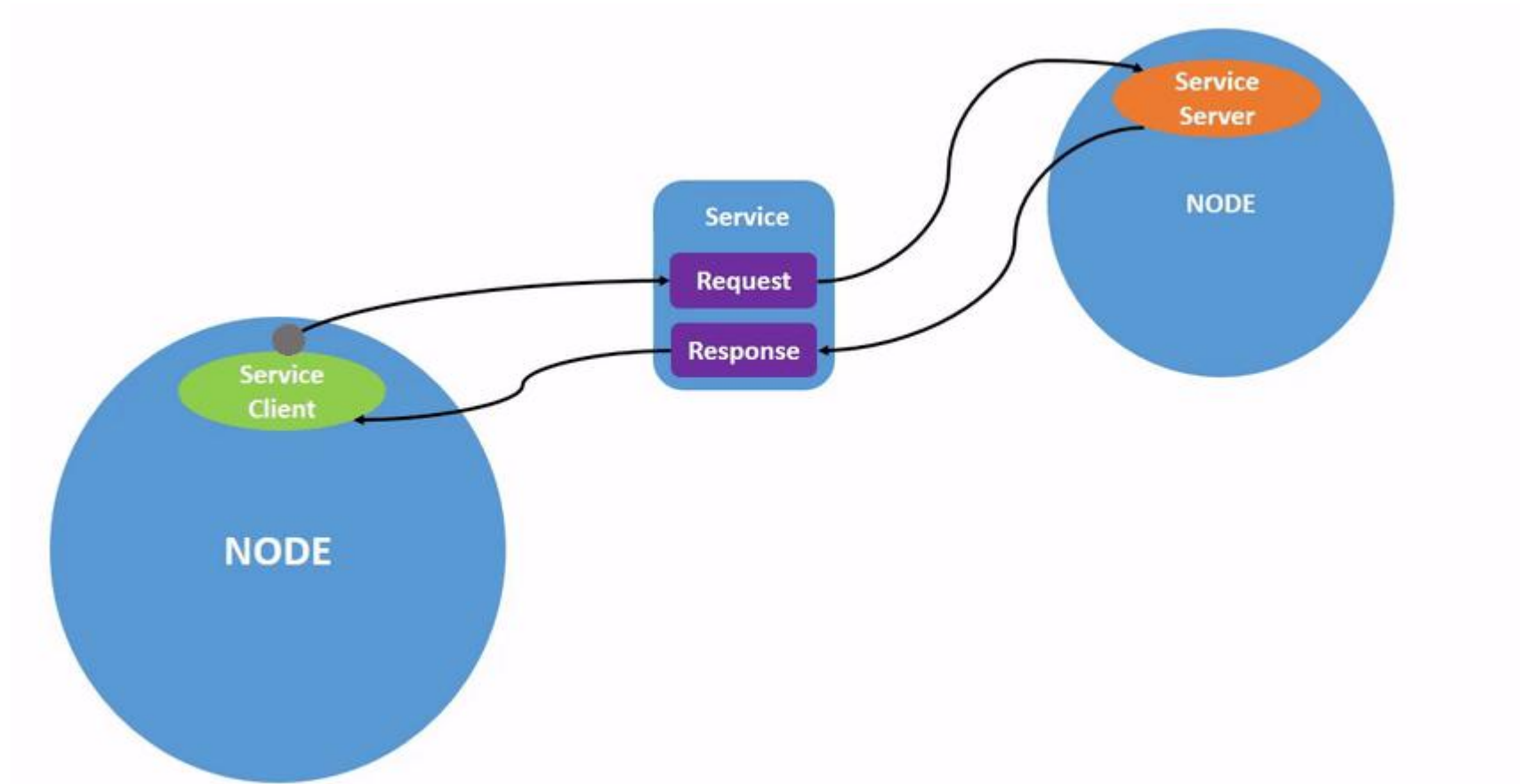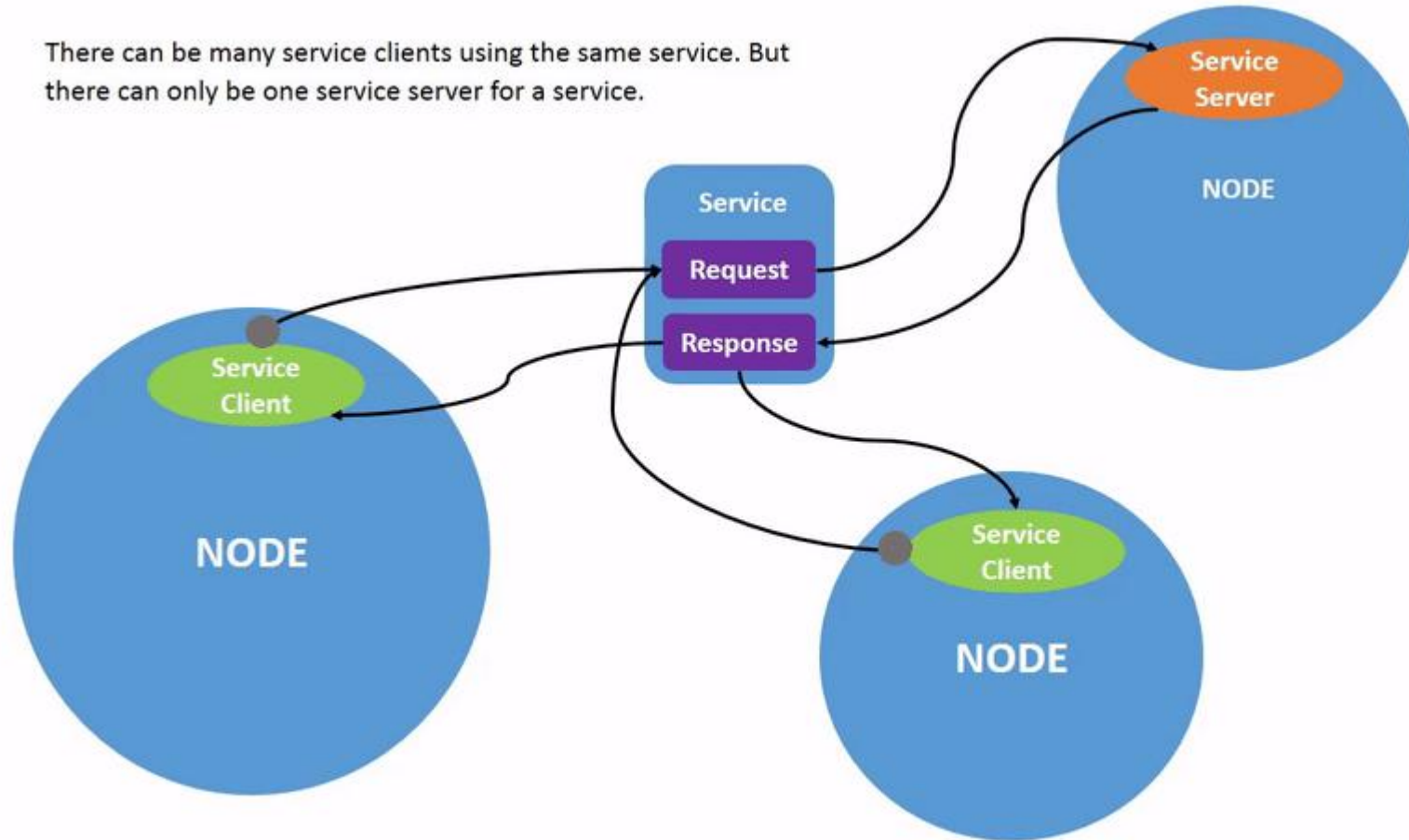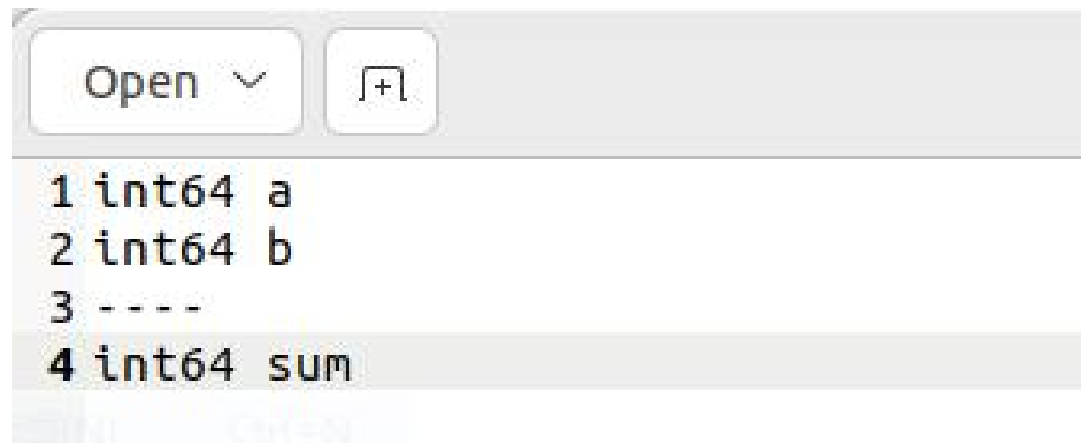
There can be many service clients using the same service. But there can only be one service server for a service.

# Create service message

```
$ cd ~/ros2_ws/src
$ ros2 pkg create data_type --build-type ament_cmake --dependencies
rosidl_default_generators
$ cd ~/ros2_ws/src/data_type/
$ mkdir srv
$ cd srv
$ touch AddTwoInt.srv
```

```
Open ∨    ⊞

1 int64 a
2 int64 b
3 ----
4 int64 sum
```

```
                                              CMakeLists.txt
 Open ∨    ⊡                                ~/ros2_ws/src/data_type

 1 cmake_minimum_required(VERSION 3.8)
 2 project(data_type)
 3
 4 if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
 5   add_compile_options(-Wall -Wextra -Wpedantic)
 6 endif()
 7
 8 # find dependencies
 9 find_package(ament_cmake REQUIRED)
10 find_package(rosidl_default_generators REQUIRED)
11 rosidl_generate_interfaces(${PROJECT_NAME}
12 "srv/AddTwoInt.srv"
13 )
14 ament_export_dependencies(rosidl_default_runtime)
15 if(BUILD_TESTING)
16   find_package(ament_lint_auto REQUIRED)
17   # the following line skips the linter which checks for copyrights
18   # comment the line when a copyright and license is added to all source files
19   set(ament_cmake_copyright_FOUND TRUE)
20   # the following line skips cpplint (only works in a git repo)
21   # comment the line when this package is in a git repo and when
22   # a copyright and license is added to all source files
23   set(ament_cmake_cpplint_FOUND TRUE)
24   ament_lint_auto_find_test_dependencies()
25 endif()
26
27 ament_package()
```

```xml
package.xml
~/ros2_ws/src/data_type

 1 <?xml version="1.0"?>
 2 <?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
 3 <package format="3">
 4   <name>data_type</name>
 5   <version>0.0.0</version>
 6   <description>TODO: Package description</description>
 7   <maintainer email="pi@todo.todo">pi</maintainer>
 8   <license>TODO: License declaration</license>
 9
10   <buildtool_depend>ament_cmake</buildtool_depend>
11
12   <depend>rosidl_default_generators</depend>
13
14   <test_depend>ament_lint_auto</test_depend>
15   <test_depend>ament_lint_common</test_depend>
16 <member_of_group>rosidl_interface_packages</member_of_group>
17   <export>
18     <build_type>ament_cmake</build_type>
19   </export>
20 </package>
```
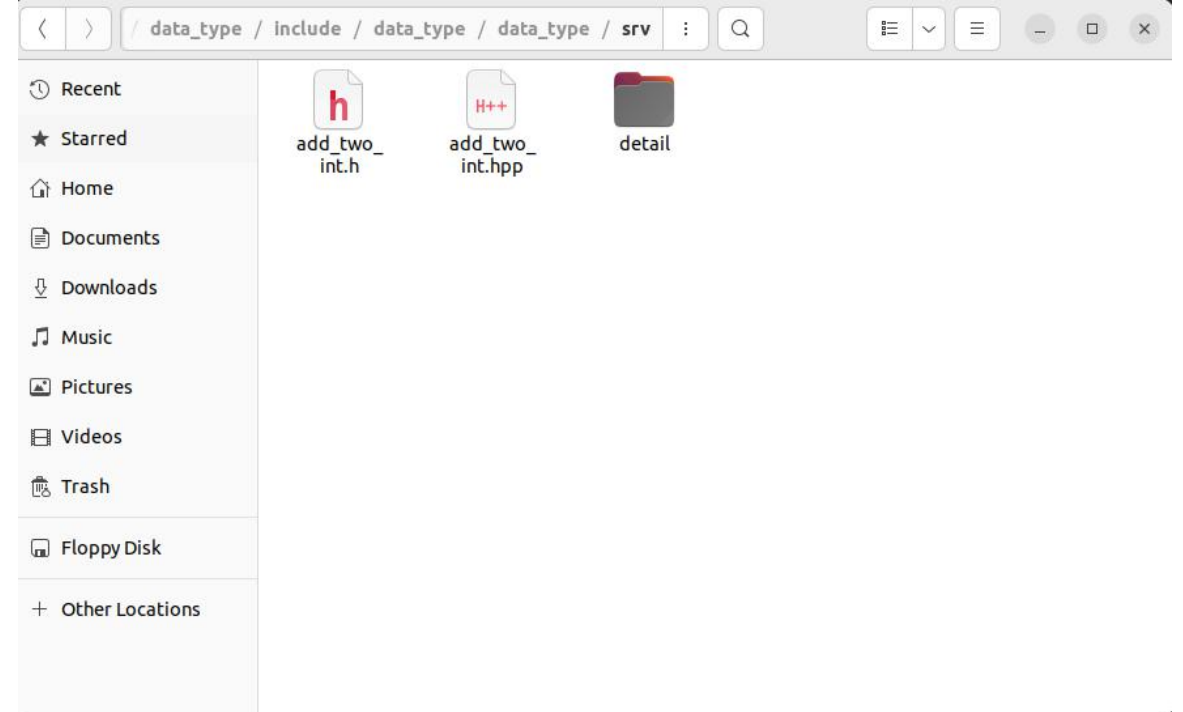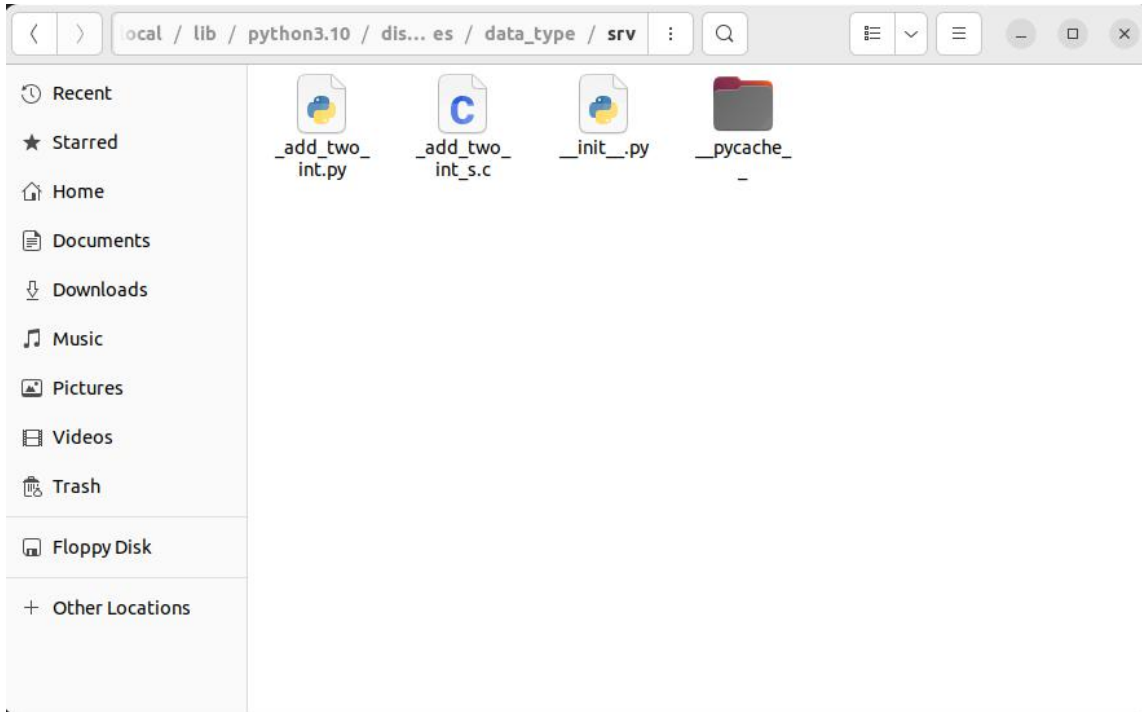
# Create server (C++)

$ cd ~/ros2_ws/src/c_pkg/src/
$ touch srv_server.cpp

```
                                           CMakeLists.txt
 Open  ⌄   ⊞                                ~/ros2_ws/src/c_pkg

10 find_package(rclcpp REQUIRED)
11 find_package(std_msgs REQUIRED)
12 find_package(data_type REQUIRED)

13
14 add_executable(listener src/listener.cpp)
15 add_executable(talker src/talker.cpp)
16 add_executable(srv_server src/srv_server.cpp)
17
18
19 target_include_directories(listener PUBLIC
20   $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
21   $<INSTALL_INTERFACE:include>)
22 target_include_directories(talker PUBLIC
23   $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
24   $<INSTALL_INTERFACE:include>)
25
26 target_compile_features(listener PUBLIC c_std_99 cxx_std_17)  # Require C99 and C++17
27 ament_target_dependencies(
28   listener
29   rclcpp
30   std_msgs
31 )
32 ament_target_dependencies(
33   talker
34   rclcpp
35   std_msgs
36 )
37 ament_target_dependencies(srv_server rclcpp data_type)
38 install(TARGETS listener
39   DESTINATION lib/${PROJECT_NAME})
40 install(TARGETS talker
41   DESTINATION lib/${PROJECT_NAME})
42 install(TARGETS
43 srv_server
44 DESTINATION lib/${PROJECT_NAME})
```

```cpp
#include "rclcpp/rclcpp.hpp"
#include "data_type/srv/add_two_int.hpp"

#include <memory>

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_INFO(rclcpp::get_logger("rclcpp"), "Ready
to add two ints.");
  rclcpp::spin(std::make_shared<SrvServer>());

  rclcpp::shutdown();
}
```

```cpp
class SrvServer : public rclcpp::Node
{

    public :
    SrvServer() : Node("srv_server")
    {
        service = this->create_service<data_type::srv::AddTwoInt>("add_two_int",
std::bind(&SrvServer::add, this, std::placeholders::_1, std::placeholders::_2));
    }
    private:
    void add(const std::shared_ptr<data_type::srv::AddTwoInt::Request> request,
            std::shared_ptr<data_type::srv::AddTwoInt::Response>      response)
    {
        response->sum = request->a + request->b;
        RCLCPP_INFO(this->get_logger(), "Incoming request\na: %ld" " b: %ld",
                        request->a, request->b);
        RCLCPP_INFO(this->get_logger(), "sending back response: [%ld]", (long int)response->sum);
    }

    rclcpp::Service <data_type::srv::AddTwoInt>::SharedPtr  service;
};
```

# Create server (Python)

$ cd ~/ros2_ws/src/py_pkg/py_pkg/
$ touch srv_server.py



```python
from setuptools import find_packages, setup

package_name = 'py_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='pi',
    maintainer_email='pi@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'talker = py_pkg.talker:main',
            'listener = py_pkg.listener:main',
            'srv_server = py_pkg.srv_server:main',
        ],
    },
)
```

setup.py
~/ros2_ws/src/py_pkg

```python
import rclpy
from rclpy.node import Node
from data_type.srv import AddTwoInt

def main():
    rclpy.init(args=None)
    node = SrvServer()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

```python
class SrvServer(Node):
    def __init__(self):
        super().__init__("srv_server")
        self.service_ =self.create_service(AddTwoInt,'add_two_int',self.add)
    def add(self,request,response):
        response.sum = request.a+request.b
        self.get_logger().info(f"Incoming requst:{request.a}"+f":{request.b}")
        self.get_logger().info(f"sending back response:{response.sum}")
        return response
```

## Create client (C++)

```
$ cd ~/ros2_ws/src/c_pkg/src
$ touch srv_client.cpp
```

```cpp
#include "rclcpp/rclcpp.hpp"
#include "data_type/srv/add_two_int.hpp"
#include <chrono>
#include <cstdlib>
#include <memory>
using namespace std::chrono_literals;

int main(int argc, char **argv)
{
  rclcpp::init(argc, argv);
  auto node = std::make_shared<SrvClient>();
  node->send_request();
  rclcpp::shutdown();
  return 0;
}
```

```cpp
class SrvClient : rclcpp::Node
{

    public:
    SrvClient():Node("srv_client")
    {

        client_ = this->create_client<data_type::srv::AddTwoInt>("add_two_int");
    }
    void send_request()
  {

    auto request = std::make_shared<data_type::srv::AddTwoInt::Request>();
    request->a = 2;
    request->b = 3;
auto result_future = client_->async_send_request(request);
if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), result_future) ==
        rclcpp::FutureReturnCode::SUCCESS)
    {

      RCLCPP_INFO(this->get_logger(), "Result: %ld", result_future.get()->sum);
    }
    else
    {

      RCLCPP_ERROR(this->get_logger(), "Failed to call service");
    }
  }
    rclcpp::Client<data_type::srv::AddTwoInt>::SharedPtr client_;
} ;
```

## Create client (Python)

$ cd ~/ros2_ws/src/py_pkg/py_pkg
$ touch srv_client.py

```python
from setuptools import find_packages, setup

package_name = 'py_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='pi',
    maintainer_email='pi@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'talker = py_pkg.talker:main',
            'listener = py_pkg.listener:main',
            'srv_server = py_pkg.srv_server:main',
            'srv_client = py_pkg.srv_client:main',
        ],
    },
)
```

```python
import rclpy
from rclpy.node import Node
from data_type.srv import AddTwoInt

def main(args=None):
    rclpy.init(args=args)

    client = SrvClient()
    client.send_request()

    rclpy.spin(client)

    client.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

```python
class SrvClient(Node):
    def __init__(self):
        super().__init__("srv_client")
        self.client_ = self.create_client(AddTwoInt,"add_two_int")
        while not self.client_.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('service not available, waiting
again...')
        self.req = AddTwoInt.Request()
    def send_request(self):
        self.req.a = 41
        self.req.b = 1
        self.future = self.client_.call_async(self.req)
        self.future.add_done_callback(self.callback)

    def callback(self, future):
        try:
            response = future.result()
        except Exception as e:
            self.get_logger().info('Service call failed %r' % (e,))
        else:
            self.get_logger().info(
                'Result of add_two_ints: for %d + %d = %d' %
                (self.req.a, self.req.b, response.sum))
```

Thanks for Listening