Lab3

Content

Reading the official doc is highly recommended!

ROS 2 Documentation: Humble

Search docs

Installation
Distributions
Tutorials
Beginner: CLI tools
Configuring environment
Using turtlesim, ros2, and rqt
Understanding nodes
Understanding topics
Understanding services
Understanding parameters
Understanding actions
Using rqt_console to view logs
Launching nodes
Recording and playing back data

Tutorials / Beginner: CLI tools Edit on GitHub

You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at Jazzy.

Beginner: CLI tools

- Configuring environment
- Using `turtlesim`, `ros2`, and `rqt`
- Understanding nodes
- Understanding topics
- Understanding services
- Understanding parameters
- Understanding actions
- Using `rqt_console` to view logs
- Launching nodes
- Recording and playing back data

Previous Next

optional

ROS 2 Documentation: Humble

Search docs

Installation
Distributions
Tutorials
Beginner: CLI tools
Beginner: Client libraries
Using colcon to build packages
Creating a workspace
Creating a package
Writing a simple publisher and subscriber (C++)
Writing a simple publisher and subscriber (Python)
Writing a simple service and client (C++)
Writing a simple service and client (Python)
Creating custom msg and srv files
Implementing custom interfaces
Using parameters in a class (C++)
Using parameters in a class (Python)
Using ros2doctor to identify issues
Creating and using plugins (C++)

Tutorials / Beginner: Client libraries
Edit on GitHub

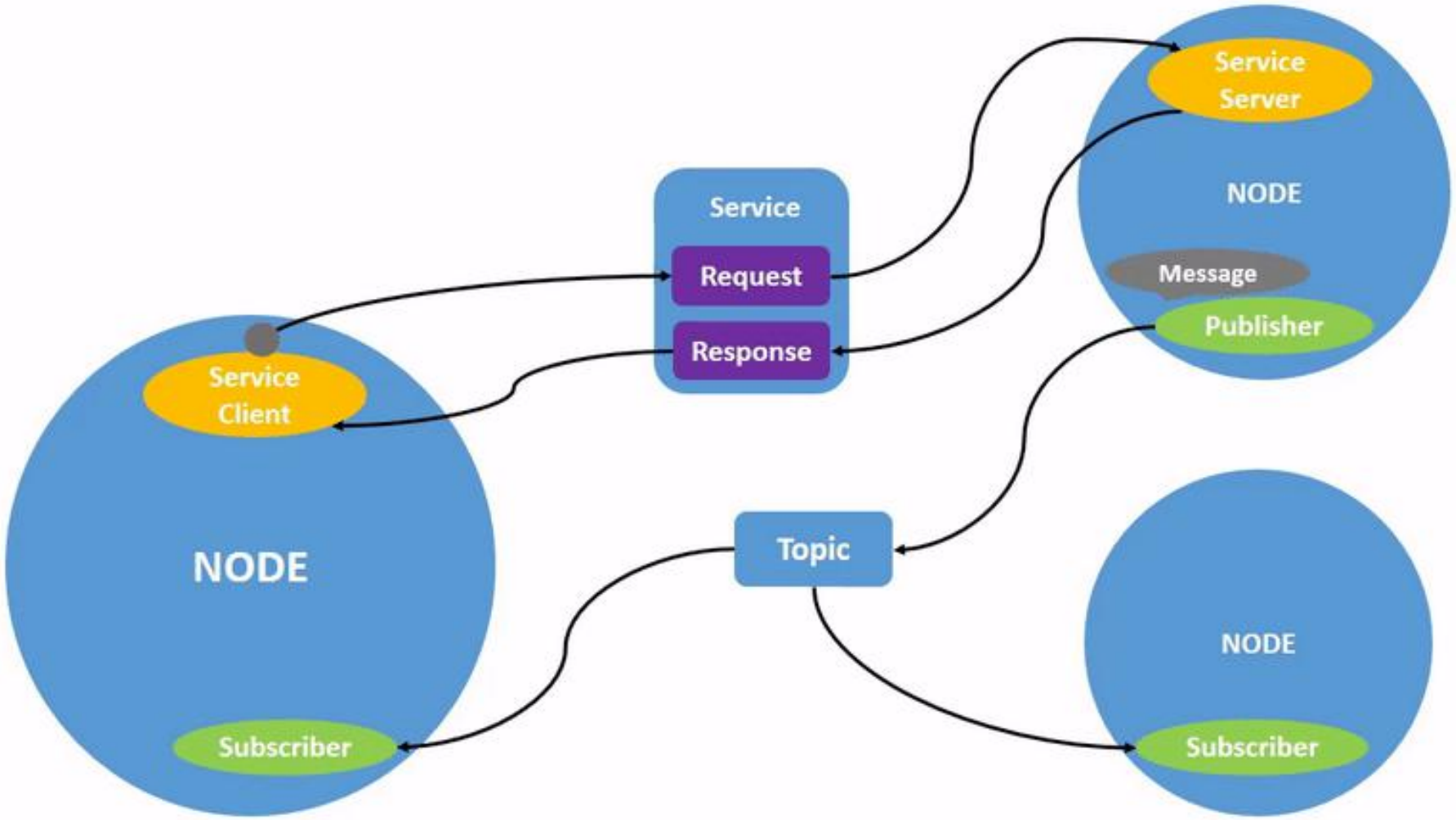
You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at Jazzy.

Beginner: Client libraries

- Using `colcon` to build packages
- Creating a workspace
- Creating a package
- Writing a simple publisher and subscriber (C++)
- Writing a simple publisher and subscriber (Python)
- Writing a simple service and client (C++)
- Writing a simple service and client (Python)
- Creating custom msg and srv files
- Implementing custom interfaces
- Using parameters in a class (C++)
- Using parameters in a class (Python)
- Using `ros2doctor` to identify issues
- Creating and using plugins (C++)

Previous Next

1 Node


Service
Request
Response
Service Server
NODE
Message
Publisher
Service Client
NODE
Subscriber
Topic
NODE
Subscriber

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
```

Open litsener.cpp ~/ros2_ws/src/c_pkg/src

```cpp
#include <cstdio>

int main(int argc, char ** argv)
{
  (void) argc;
  (void) argv;

  printf("hello world c_pkg package\n");
  return 0;
}
```

1 Node

Open | *CMakeLists.txt ~/ros2_ws/src/c_pkg | Save

```cmake
cmake_minimum_required(VERSION 3.8)
project(c_pkg)

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)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)

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

install(TARGETS litsener
  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()
```

Open | package.xml ~/ros2_ws/src/c_pkg | Save

```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>c_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="pi@todo.todo">pi</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>
```

1 Node

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

Open

CMakeLists.txt
~/ros2_ws/src/c_pkg

```
cmake_minimum_required(VERSION 3.8)
project(c_pkg)

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)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)

add_executable(listener src/listener.cpp)
add_executable(talker src/talker.cpp)
target_include_directories(listener PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)
target_include_directories(talker PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)
target_compile_features(listener PUBLIC c_std_99 cxx_std_17)  # Require C99 and C++17
ament_target_dependencies(
  listener
  rclcpp
  std_msgs
)
ament_target_dependencies(
  talker
  rclcpp
  std_msgs
)
install(TARGETS listener
  DESTINATION lib/${PROJECT_NAME})
install(TARGETS talker
  DESTINATION lib/${PROJECT_NAME})


if(BUILD_TESTING)
```

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

```python
def main():
    print('Hi from py_pkg.')


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

1 Node

Open setup.py ~/ros2_ws/src/py_pkg Sav

```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'
        ],
    },
)
```

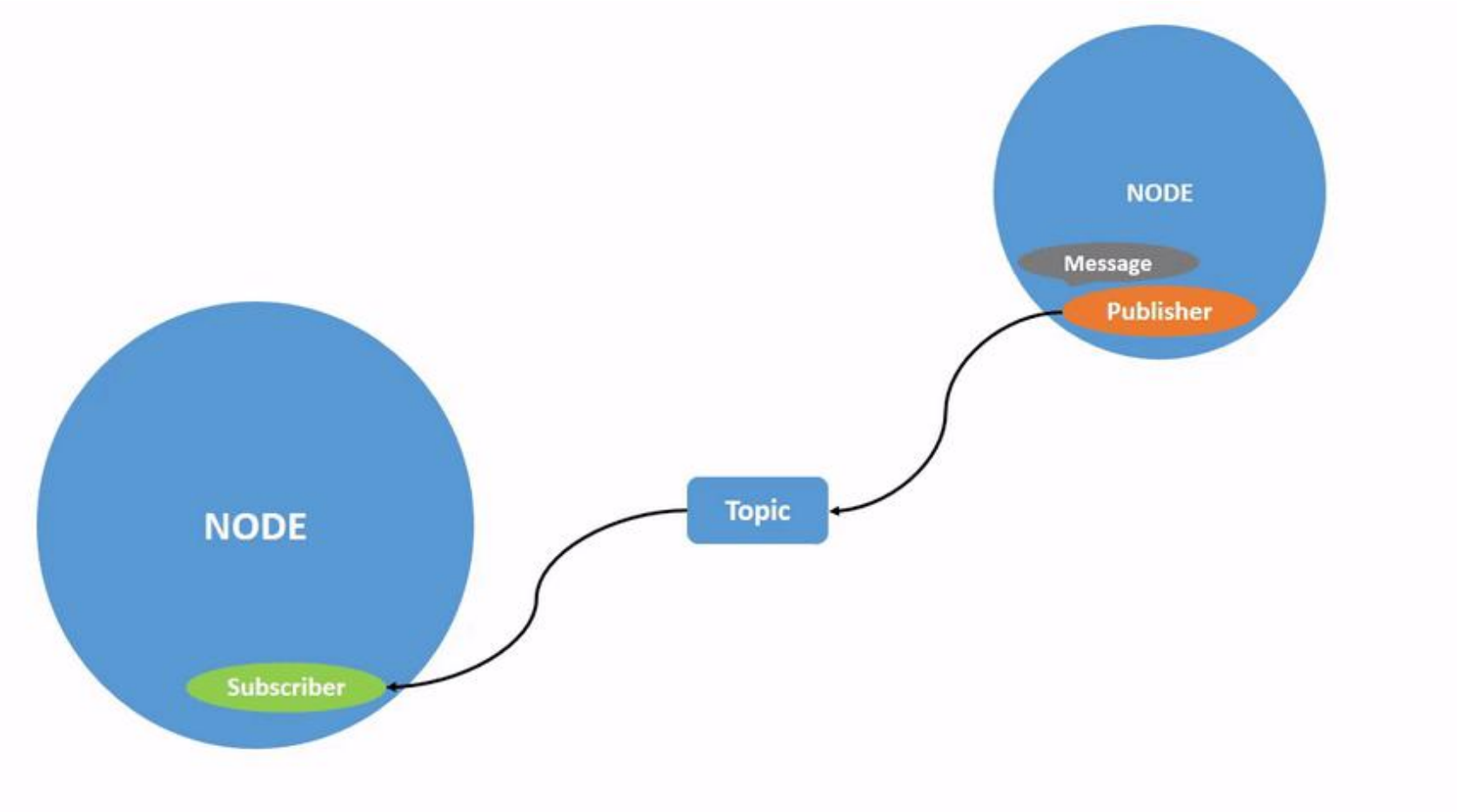
1 Node

```
$ cd ~/ros2_ws/src/py_pkg/py_pkg/
$ touch listener.py
```

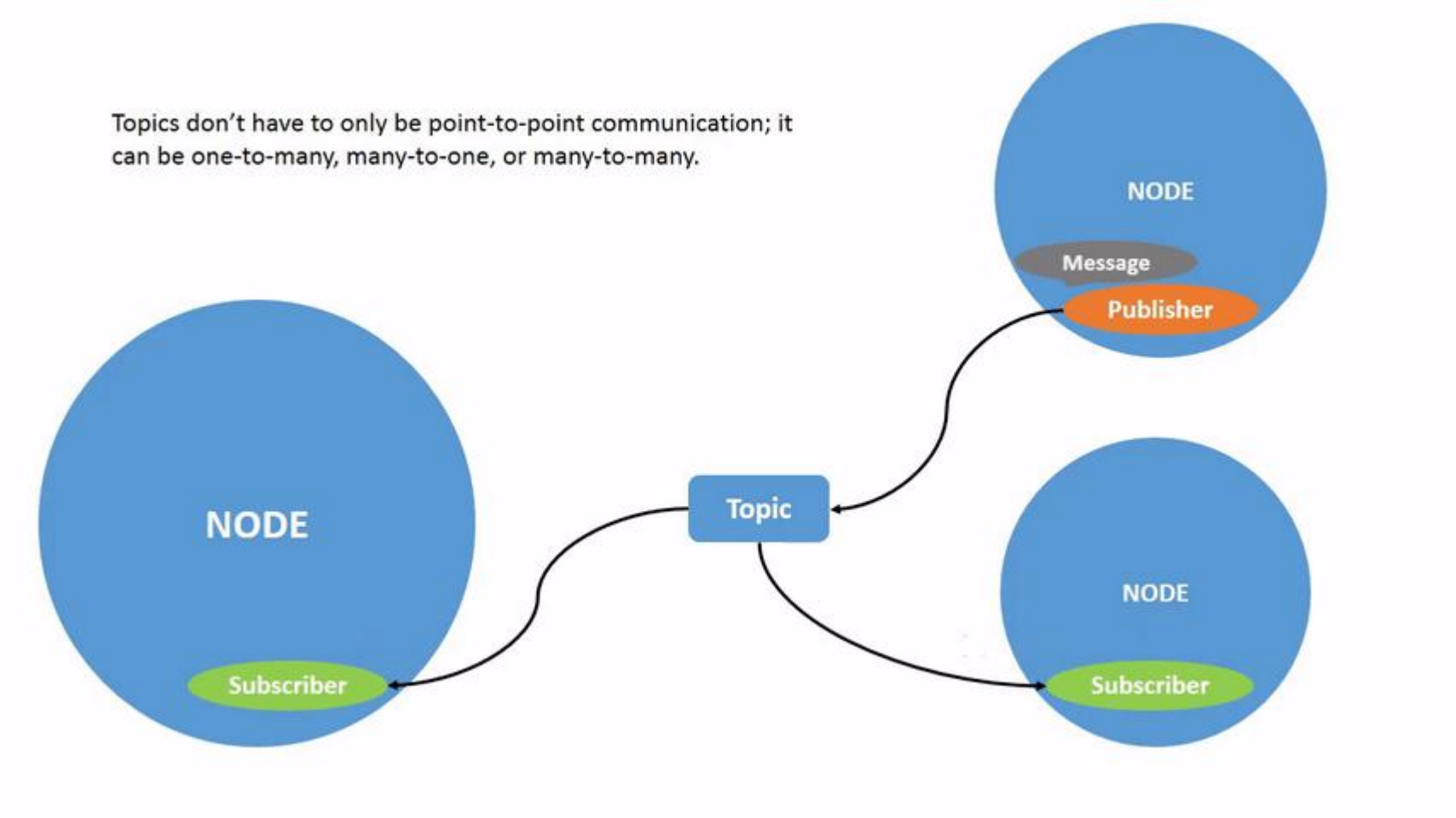
Open setup.py ~/ros2_ws/src/py_pkg Save

```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'
        ],
    },
)
```

2 Topic



2 Topic


Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.
NODE
Message
Publisher
NODE
Subscriber
Topic
NODE
Subscriber

Create first publisher (C++)

```
#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;
}
```

```
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++)

```
#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;
}
```

```
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)

```
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 Subcriber (Python)

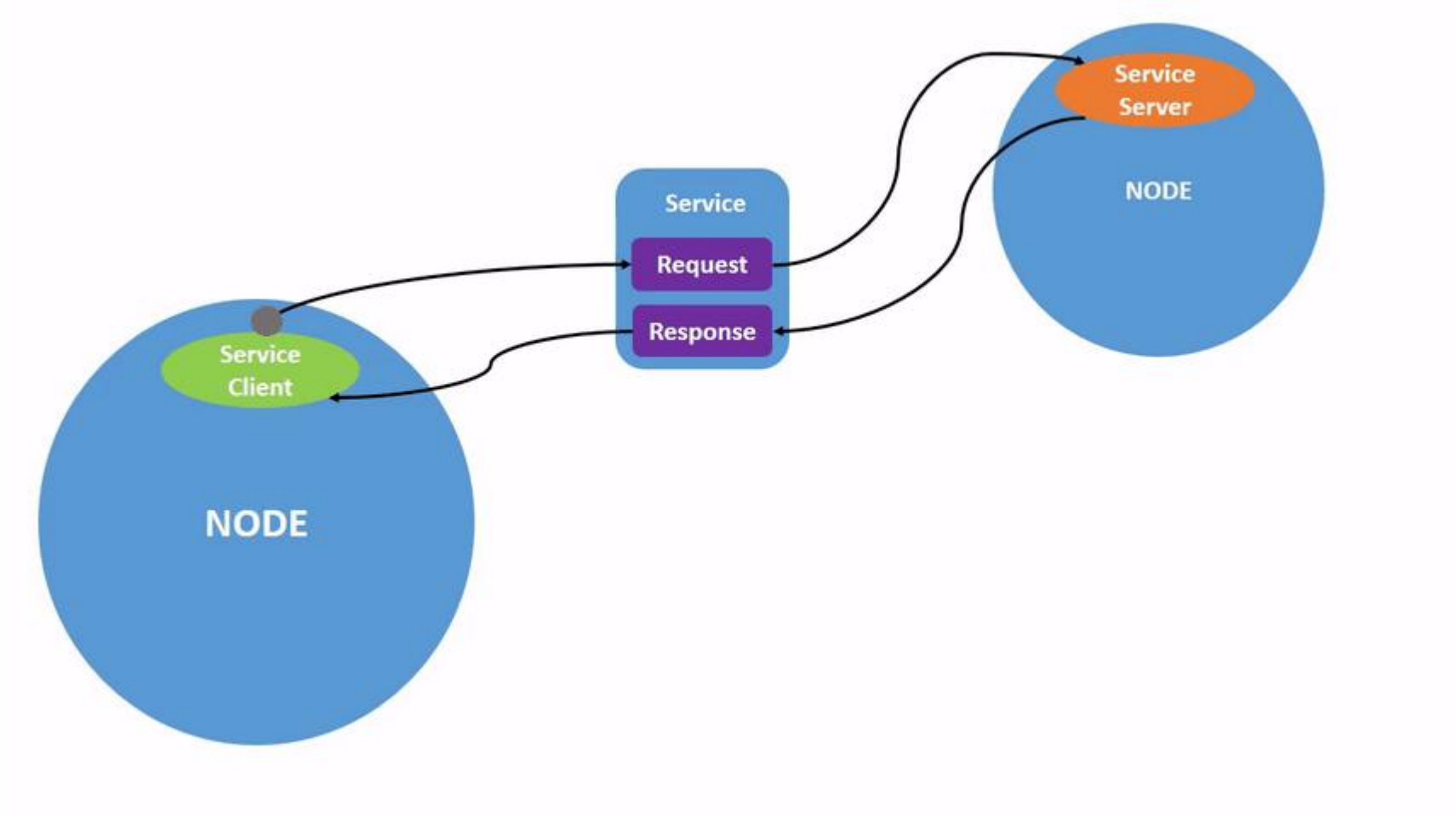
```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String


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


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

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

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

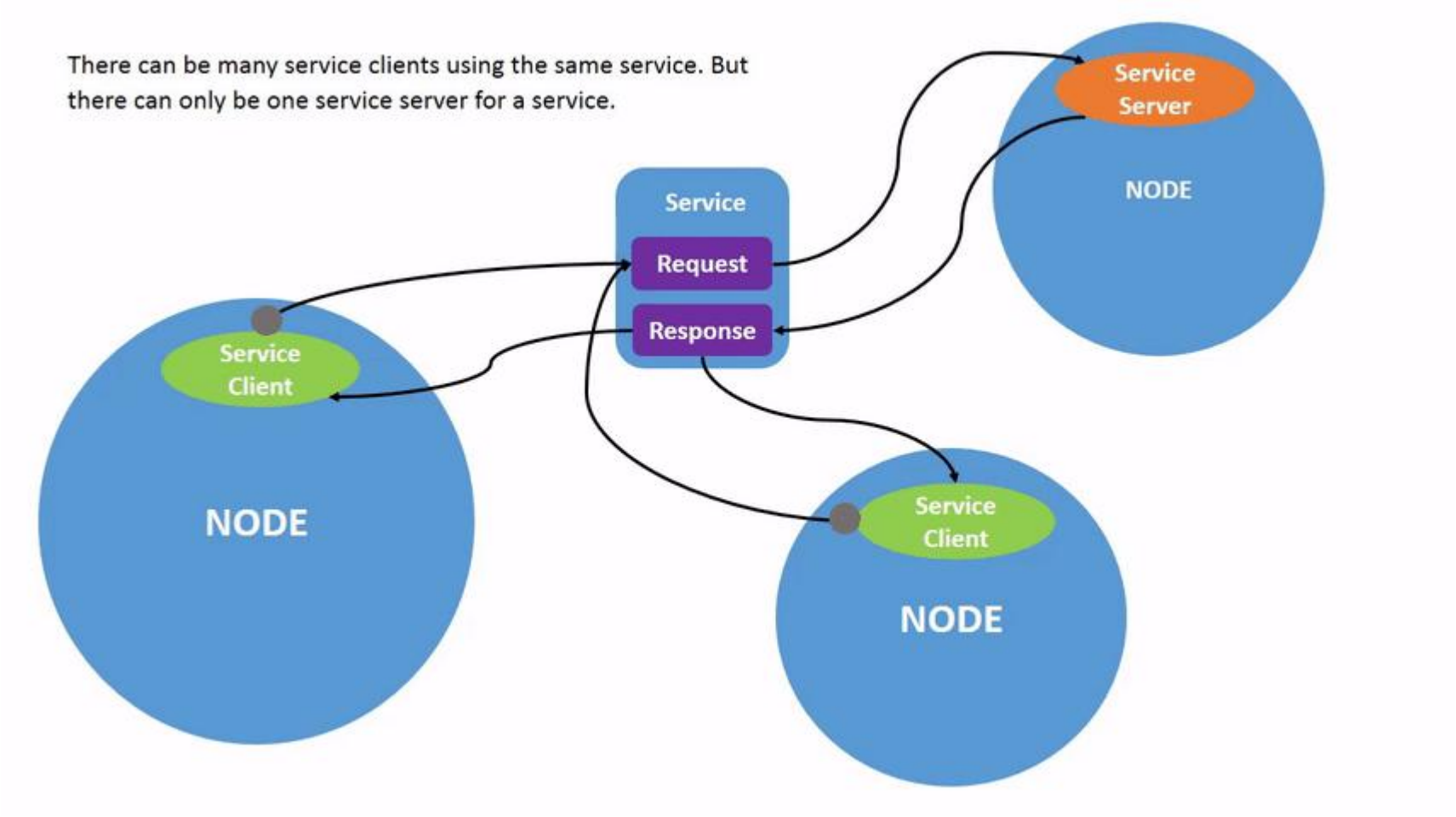
2 Topic

```
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.
```

3 Service


Service
Server
NODE
Service
Request
Response
Service
Client
NODE

3 Service


There can be many service clients using the same service. But there can only be one service server for a service.
Service
Request
Response
Service Server
NODE
Service Client
NODE
Service Client
NODE

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

```
int64 a
int64 b
---
int64 sum
```

3 Service

Open

CMakeLists.txt
~/ros2_ws/src/data_type

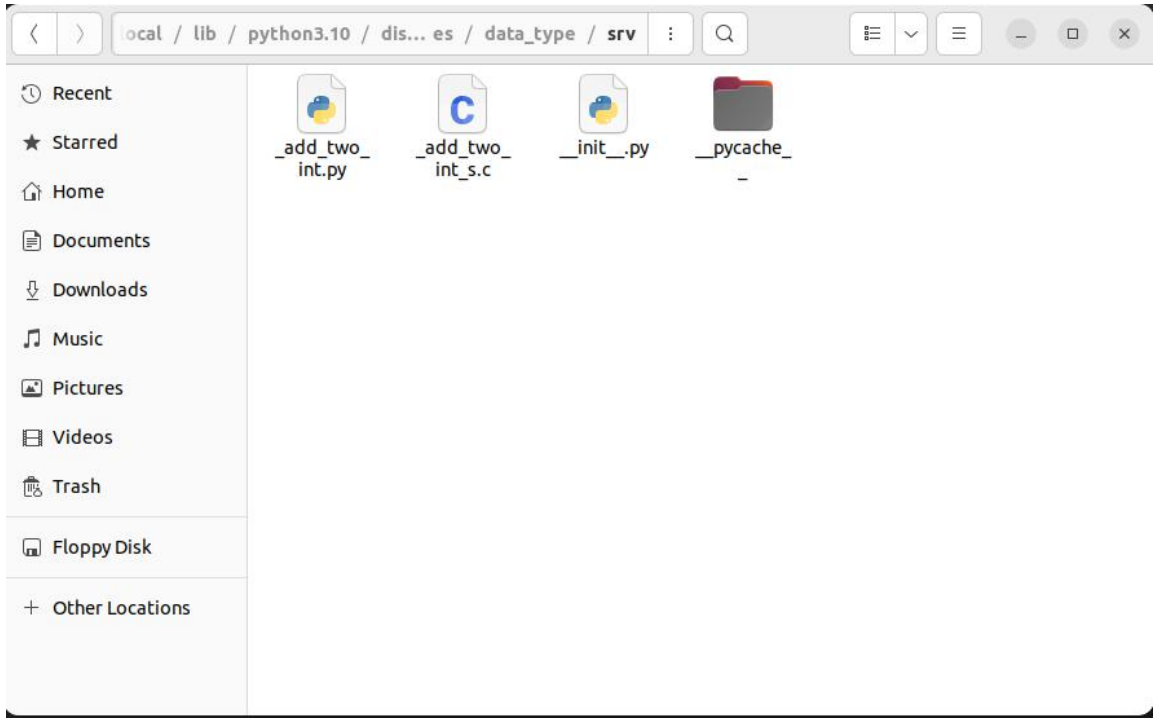
```
cmake_minimum_required(VERSION 3.8)
project(data_type)

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)
find_package(rosidl_default_generators REQUIRED)
rosidl_generate_interfaces(${PROJECT_NAME}
"srv/AddTwoInt.srv"
)
ament_export_dependencies(rosidl_default_runtime)
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()
```

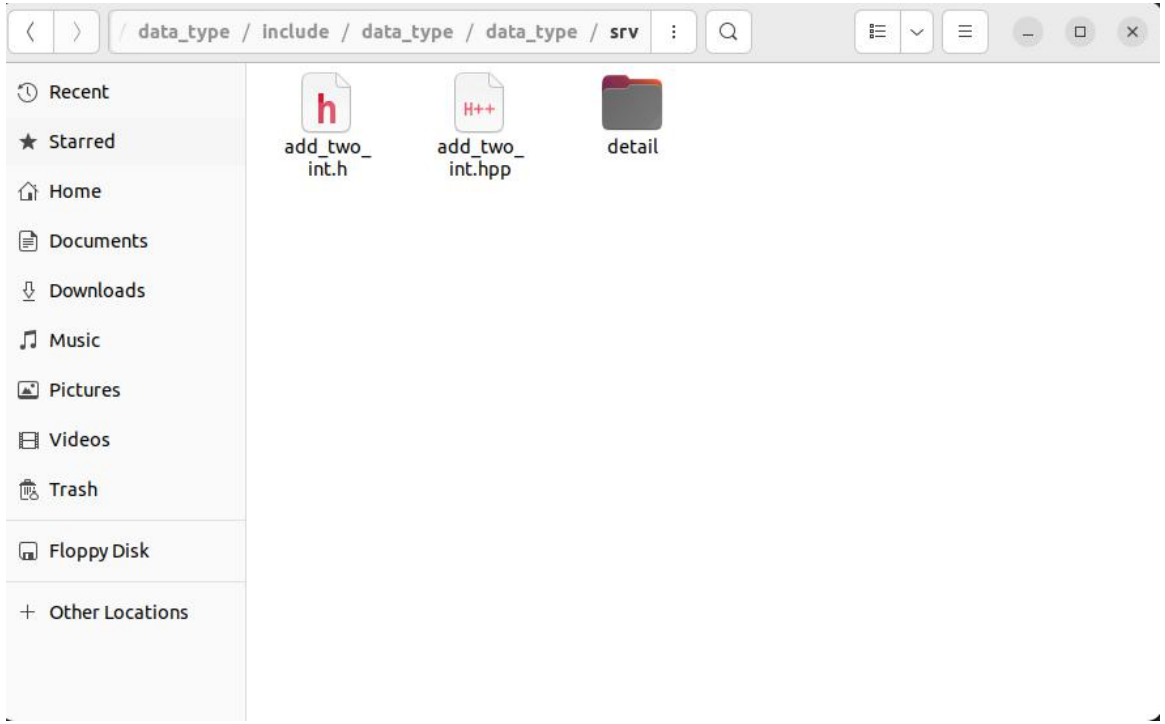
3 Service

Open | package.xml ~/ros2_ws/src/data_type | Save

```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>data_type</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="pi@todo.todo">pi</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rosidl_default_generators</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

3 Service


ocal / lib / python3.10 / dis... es / data_type / srv
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Floppy Disk
Other Locations
_add_two_int.py
_add_two_int_s.c
__init__.py
__pycache__


data_type / include / data_type / data_type / srv
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Floppy Disk
Other Locations
add_two_int.h
add_two_int.hpp
detail

Create server (C++)

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

Open

CMakeLists.txt
~/ros2_ws/src/c_pkg

```
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(data_type REQUIRED)

add_executable(listener src/listener.cpp)
add_executable(talker src/talker.cpp)
add_executable(srv_server src/srv_server.cpp)


target_include_directories(listener PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)
target_include_directories(talker PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)

target_compile_features(listener PUBLIC c_std_99 cxx_std_17)  # Require C99 and C++17
ament_target_dependencies(
  listener
  rclcpp
  std_msgs
)
ament_target_dependencies(
  talker
  rclcpp
  std_msgs
)
ament_target_dependencies(srv_server rclcpp data_type)
install(TARGETS listener
  DESTINATION lib/${PROJECT_NAME})
install(TARGETS talker
  DESTINATION lib/${PROJECT_NAME})
install(TARGETS
srv_server
DESTINATION lib/${PROJECT_NAME})
```

3 Service

```
#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();
}
```

3 Service

```
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
```

Open

setup.py
~/ros2_ws/src/py_pkg

```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',
        ],
    },
)
```

3 Service

```
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()
```

```
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
```

```
#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;
}
```

3 Service

```
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_;
}  ;
```

3 Service

Open | CMakeLists.txt ~/ros2_ws/src/c_pkg | Save

```
find_package(std_msgs REQUIRED)
find_package(data_type REQUIRED)

add_executable(listener src/listener.cpp)
add_executable(talker src/talker.cpp)
add_executable(srv_server src/srv_server.cpp)
add_executable(srv_client src/srv_client.cpp)

target_include_directories(listener PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)
target_include_directories(talker PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<INSTALL_INTERFACE:include>)

target_compile_features(listener PUBLIC c_std_99 cxx_std_17)  # Require C99 and C++17
ament_target_dependencies(
  listener
  rclcpp
  std_msgs
)
ament_target_dependencies(
  talker
  rclcpp
  std_msgs
)
ament_target_dependencies(srv_server rclcpp data_type)
ament_target_dependencies(srv_client rclcpp data_type)
install(TARGETS listener
  DESTINATION lib/${PROJECT_NAME})
install(TARGETS talker
  DESTINATION lib/${PROJECT_NAME})
install(TARGETS
srv_server
DESTINATION lib/${PROJECT_NAME})
install(TARGETS
srv_client
DESTINATION lib/${PROJECT_NAME})

```

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',
        ],
    },
)
```

3 Service

```
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()
```

```
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))
```

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Thanks for Listening