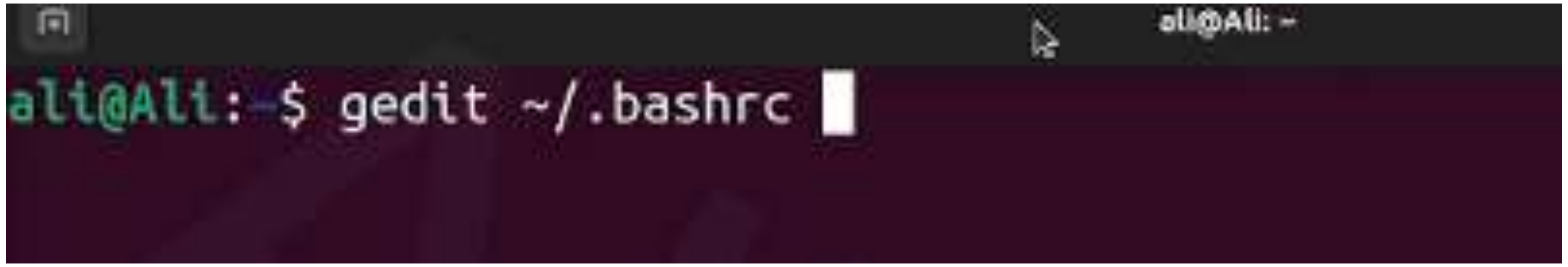
ROS2 _ Continue

Auto Sourcing your packages

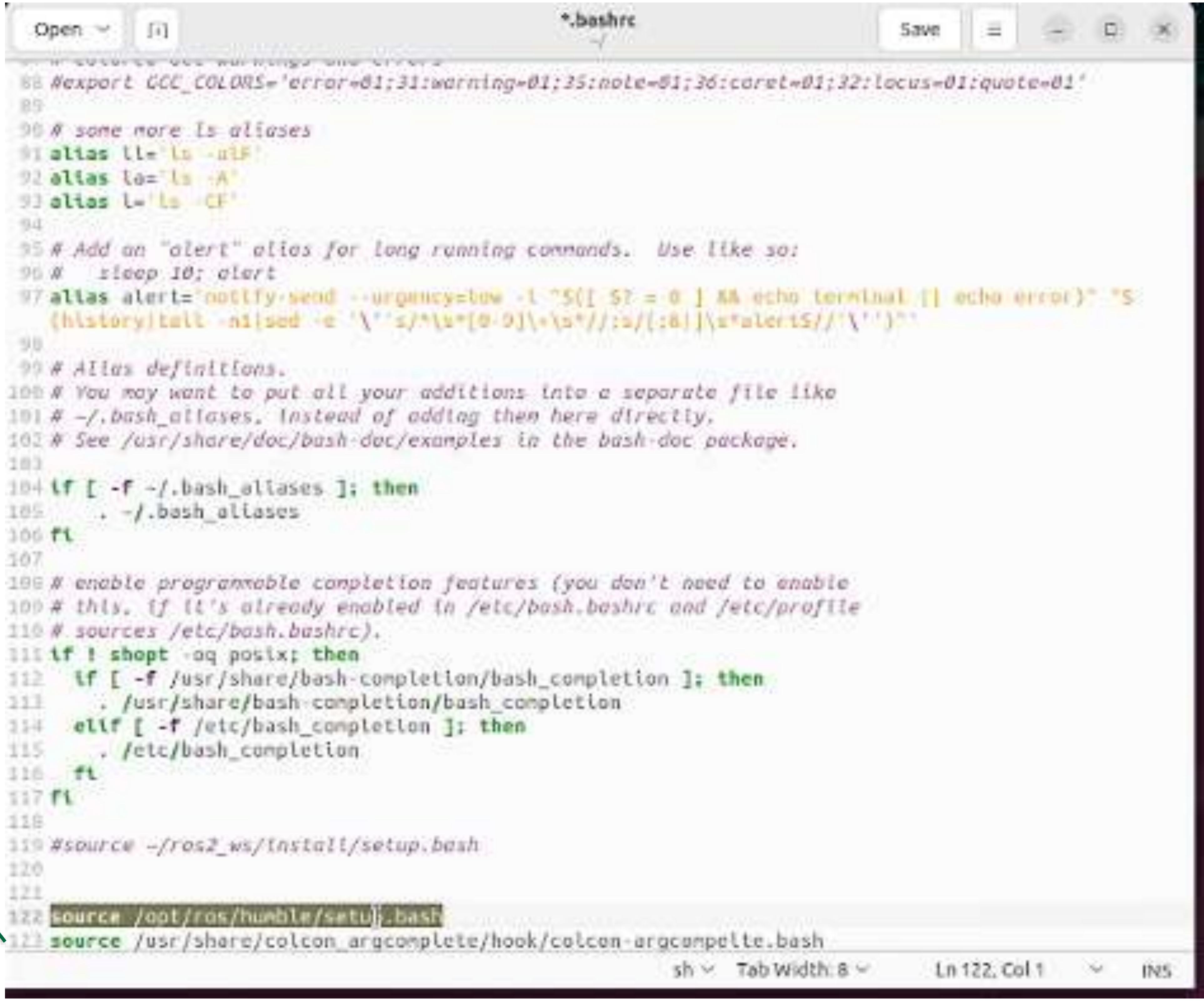
- We need to edit the bashrc file



```
ali@Ali:~$ gedit ~/.bashrc
```

Add line 122 +123 and save the file

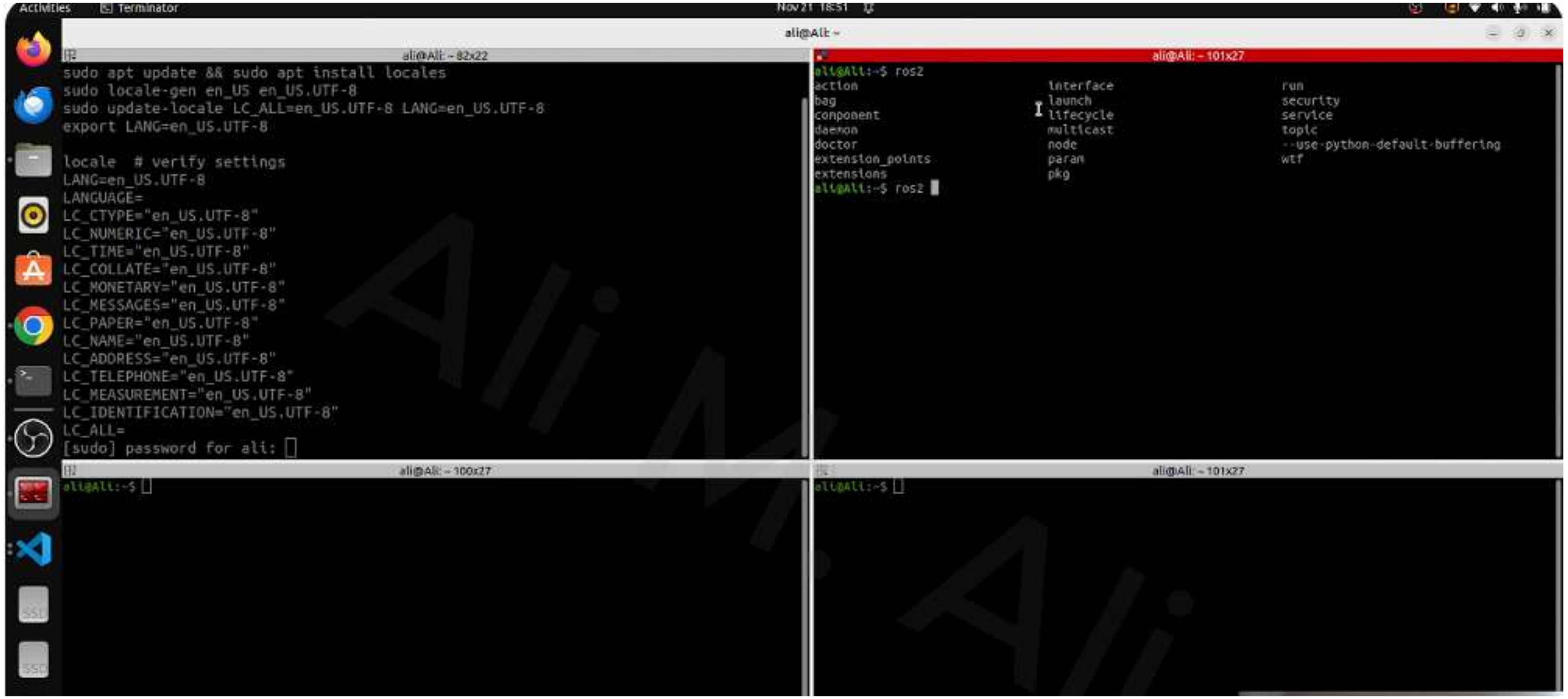
- Line 122 to auto source humble
- Line 123 for the autocomplete tool in Colcon (using double tab)



```
 87 # colored GCC warnings and errors
 88 #export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'
 89
 90 # some more ls aliases
 91 alias ll='ls -alF'
 92 alias la='ls -A'
 93 alias l='ls -CF'
 94
 95 # Add an "alert" alias for long running commands.  Use like so:
 96 #   sleep 10; alert
 97 alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$
    (history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
 98
 99 # Alias definitions.
100 # You may want to put all your additions into a separate file like
101 # ~/.bash_aliases, instead of adding them here directly.
102 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
103
104 if [ -f ~/.bash_aliases ]; then
105     . ~/.bash_aliases
106 fi
107
108 # enable programmable completion features (you don't need to enable
109 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
110 # sources /etc/bash.bashrc).
111 if ! shopt -oq posix; then
112   if [ -f /usr/share/bash-completion/bash_completion ]; then
113     . /usr/share/bash-completion/bash_completion
114   elif [ -f /etc/bash_completion ]; then
115     . /etc/bash_completion
116   fi
117 fi
118
119 #source ~/ros2_ws/install/setup.bash
120
121
122 source /opt/ros/humble/setup.bash
123 source /usr/share/colcon_argcomplete/hook/colcon-argcompelte.bash
```

Terminator

- Install it using sudo apt in terminator. Type terminator and split the terminals.
- This will allow you to work on more than one terminal in an easier setting.



```
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8

locale  # verify settings
LANG=en_US.UTF-8
LANGUAGE=
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
[sudo] password for ali:
```

```
ali@Ali:~$ ros2
action                  interface               run
bag                     launch                  security
component               lifecycle               service
daemon                  multicast               topic
doctor                  node                    --use-python-default-buffering
extension_points        param                   wtf
extensions              pkg
ali@Ali:~$ ros2
```

```
ali@Ali:~$
```

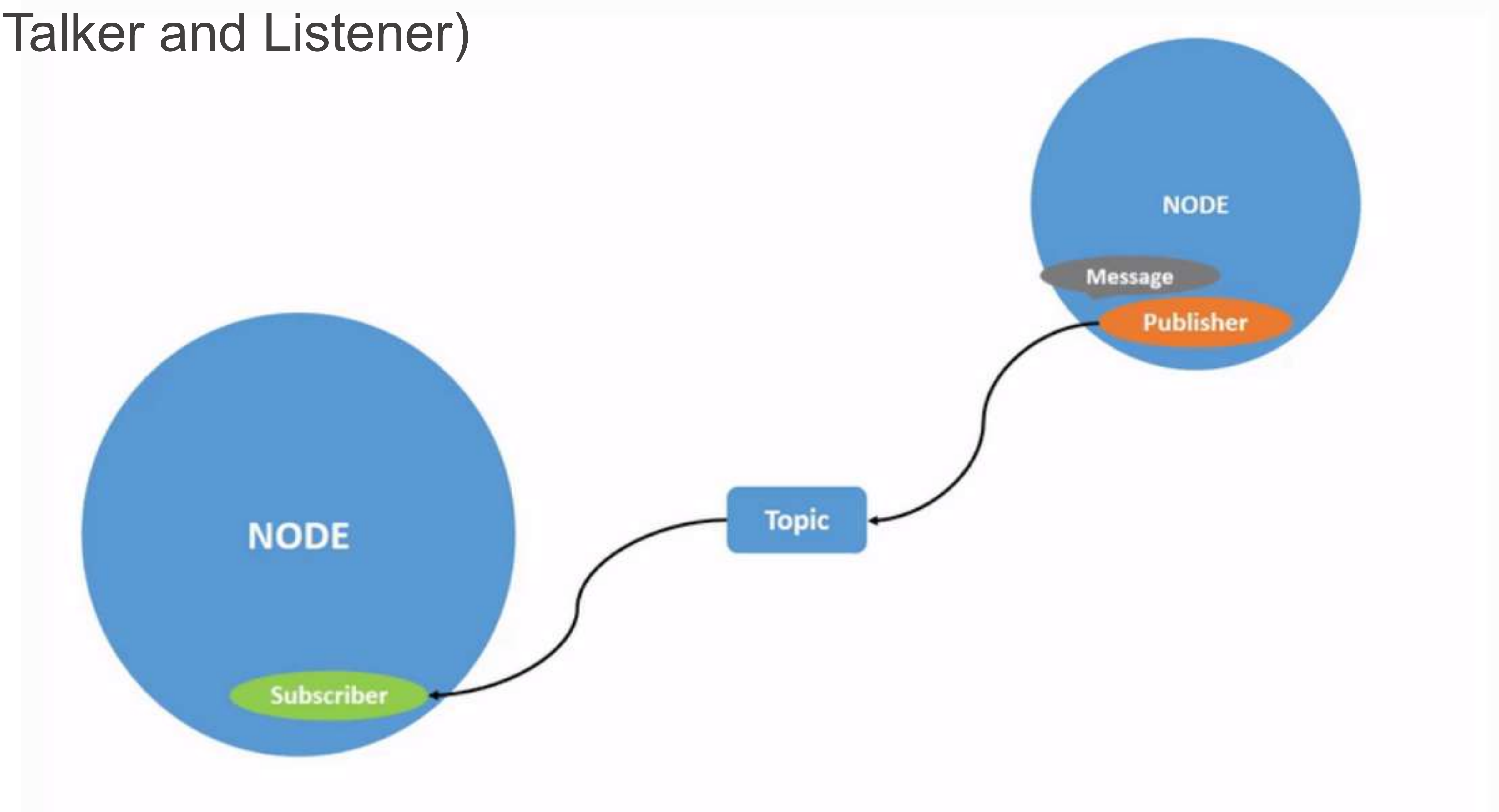
```
ali@Ali:~$
```

Nodes communication

- 1) Publish and subscribe with topics.
- 2) Service
- 3) Actions

1) Topics

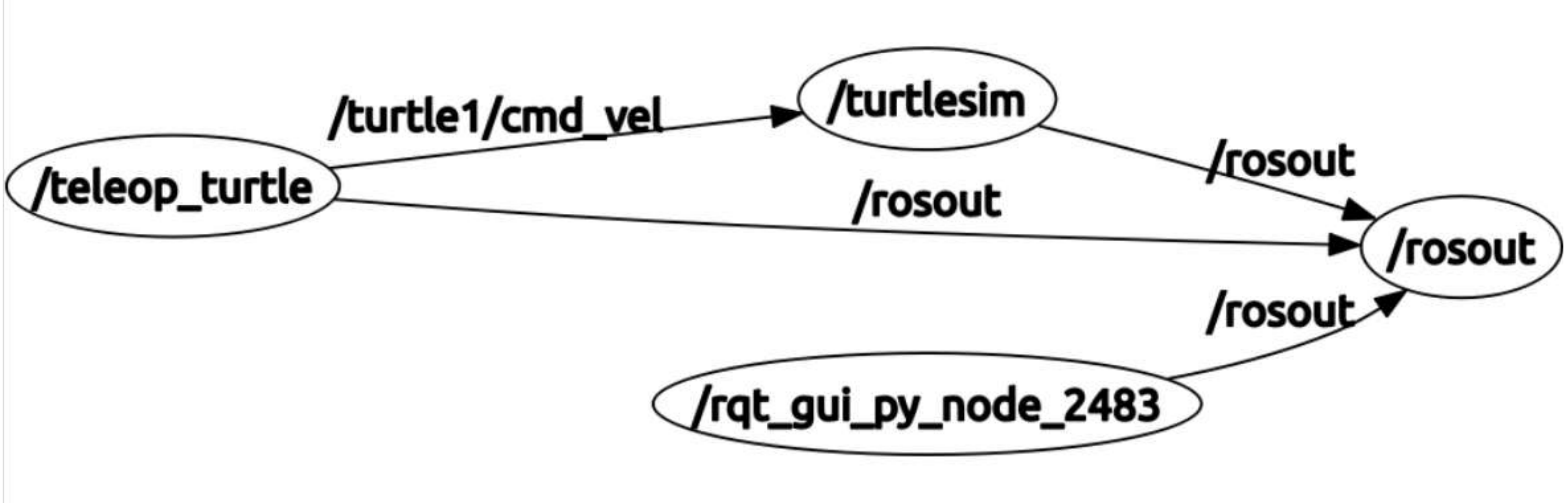
- Recall the demo nodes in the pervious slides that prints hello world everything second (Talker and Listener)



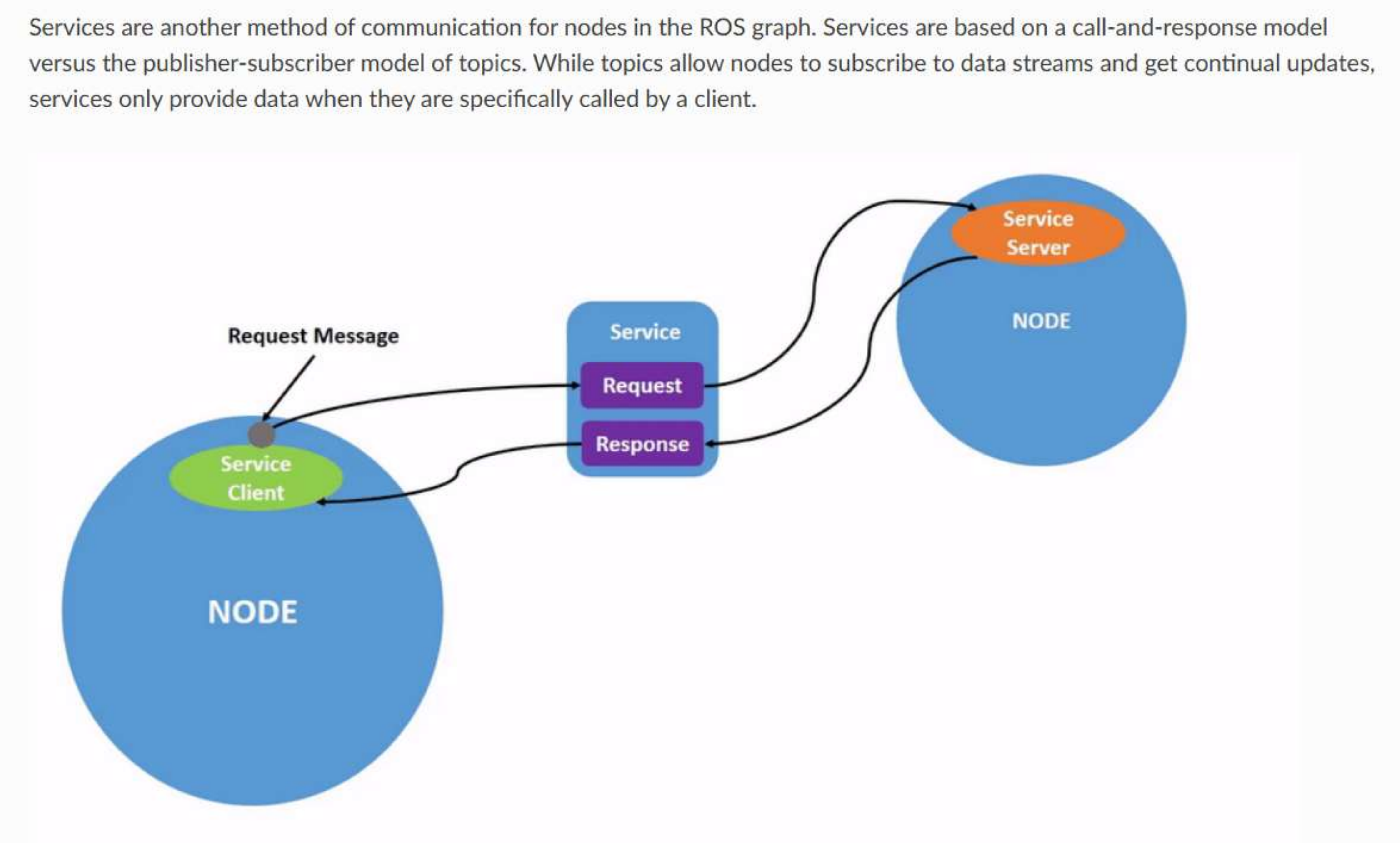
Rqt Graphs

- ros2 run rqt_graph rqt_graph

- It is used to visualize the communication between the nodes/packages

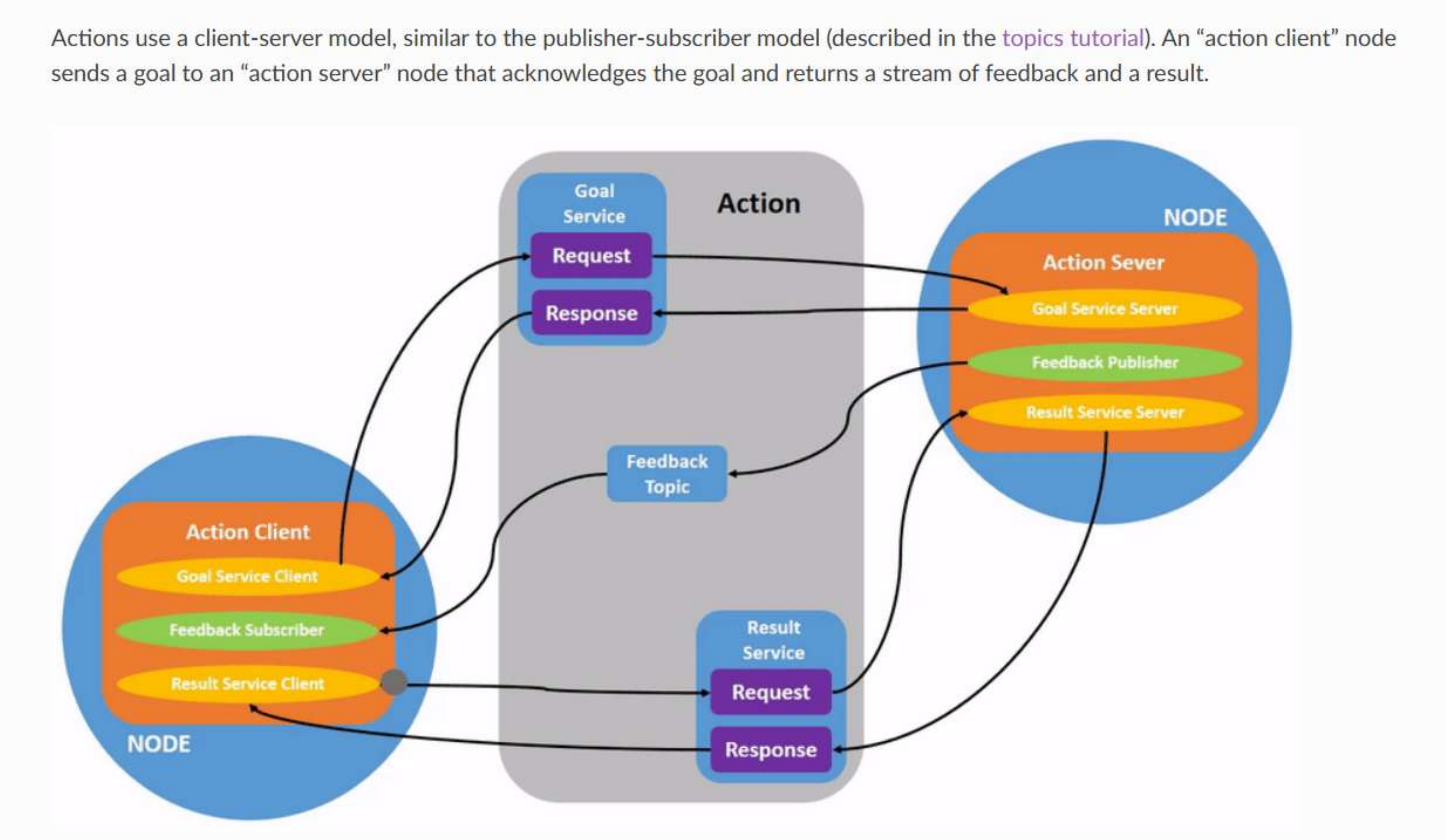


2) Service



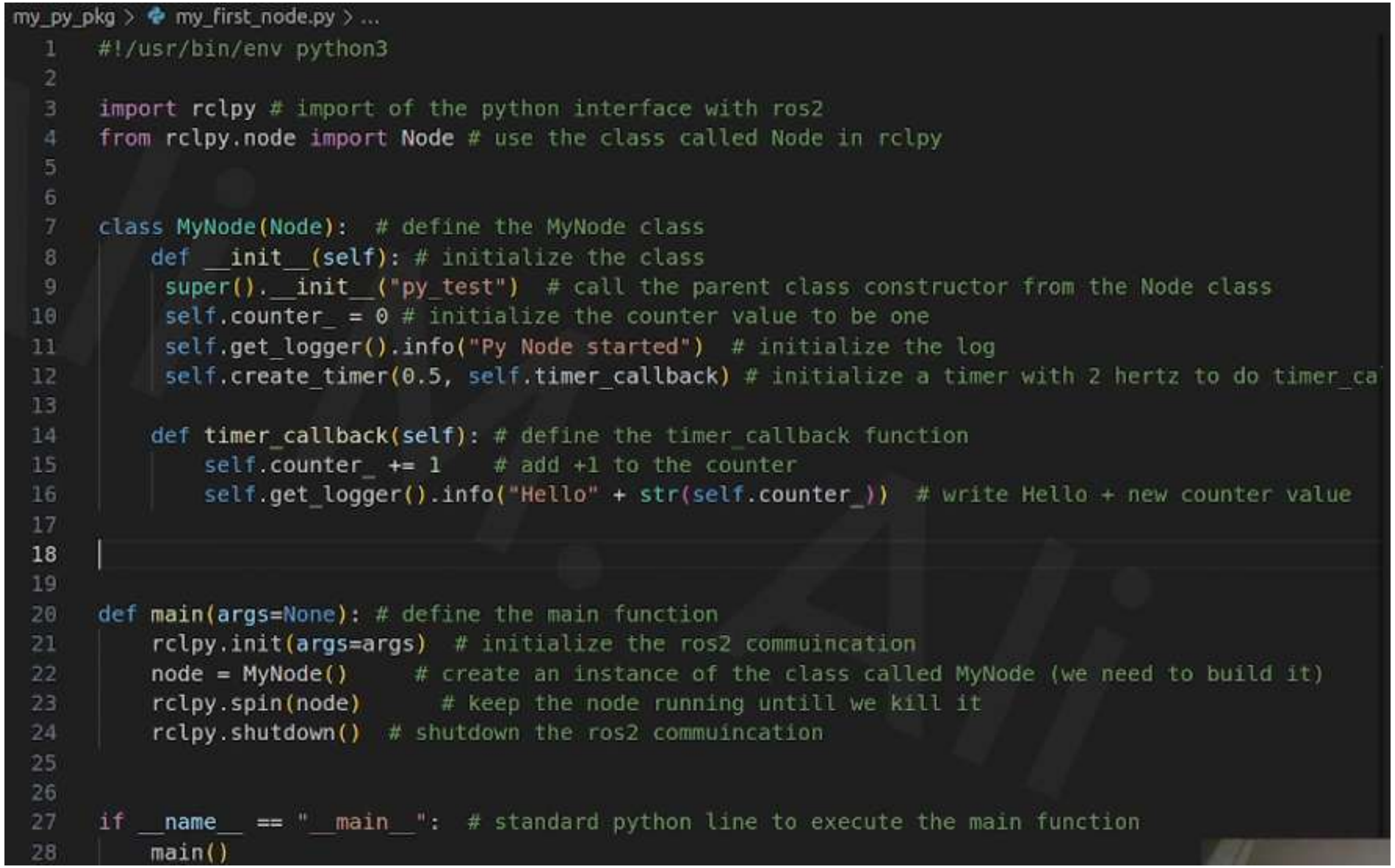
Services are another method of communication for nodes in the ROS graph. Services are based on a call-and-response model versus the publisher-subscriber model of topics. While topics allow nodes to subscribe to data streams and get continual updates, services only provide data when they are specifically called by a client.

3) Action



Actions use a client-server model, similar to the publisher-subscriber model (described in the topics tutorial). An “action client” node sends a goal to an “action server” node that acknowledges the goal and returns a stream of feedback and a result.

How to Install and Run your First Node ?

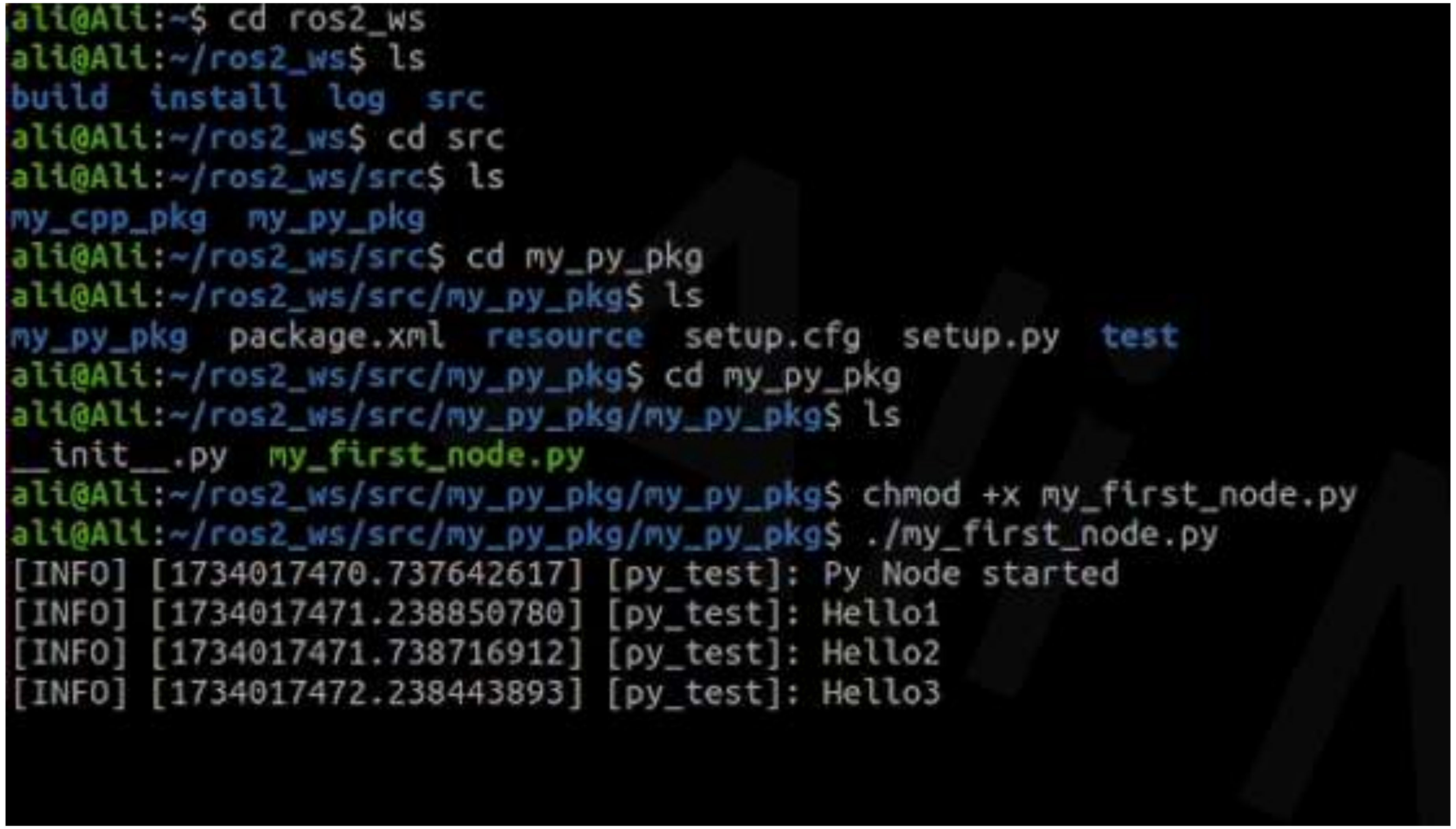


my_py_pkg > my_first_node.py > ...

```python
#!/usr/bin/env python3

import rclpy # import of the python interface with ros2
from rclpy.node import Node # use the class called Node in rclpy


class MyNode(Node):  # define the MyNode class
    def __init__(self): # initialize the class
     super().__init__("py_test")  # call the parent class constructor from the Node class
     self.counter_ = 0 # initialize the counter value to be one
     self.get_logger().info("Py Node started")  # initialize the log
     self.create_timer(0.5, self.timer_callback) # initialize a timer with 2 hertz to do timer_ca

    def timer_callback(self): # define the timer_callback function
        self.counter_ += 1    # add +1 to the counter
        self.get_logger().info("Hello" + str(self.counter_))  # write Hello + new counter value



def main(args=None): # define the main function
    rclpy.init(args=args)  # initialize the ros2 communication
    node = MyNode()     # create an instance of the class called MyNode (we need to build it)
    rclpy.spin(node)      # keep the node running untill we kill it
    rclpy.shutdown()  # shutdown the ros2 communication


if __name__ == "__main__":  # standard python line to execute the main function
    main()
```

Running your First Node

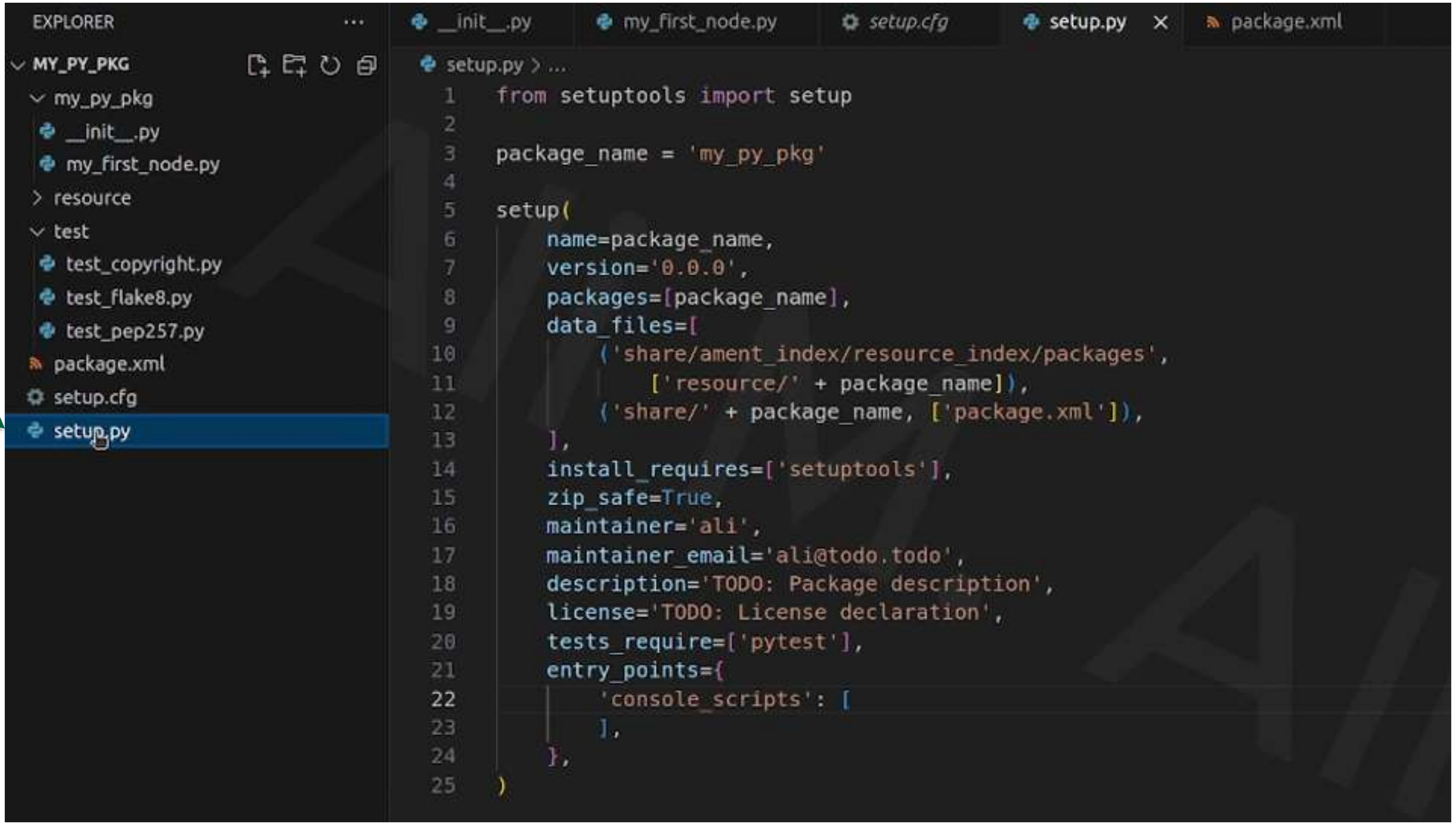
- Make sure that you saved the code first.
- Then mark the code as an executable file as follows:



```
ali@Ali:~$ cd ros2_ws
ali@Ali:~/ros2_ws$ ls
build  install  log  src
ali@Ali:~/ros2_ws$ cd src
ali@Ali:~/ros2_ws/src$ ls
my_cpp_pkg  my_py_pkg
ali@Ali:~/ros2_ws/src$ cd my_py_pkg
ali@Ali:~/ros2_ws/src/my_py_pkg$ ls
my_py_pkg  package.xml  resource  setup.cfg  setup.py  test
ali@Ali:~/ros2_ws/src/my_py_pkg$ cd my_py_pkg
ali@Ali:~/ros2_ws/src/my_py_pkg/my_py_pkg$ ls
__init__.py  my_first_node.py
ali@Ali:~/ros2_ws/src/my_py_pkg/my_py_pkg$ chmod +x my_first_node.py
ali@Ali:~/ros2_ws/src/my_py_pkg/my_py_pkg$ ./my_first_node.py
[INFO] [1734017470.737642617] [py_test]: Py Node started
[INFO] [1734017471.238850780] [py_test]: Hello1
[INFO] [1734017471.738716912] [py_test]: Hello2
[INFO] [1734017472.238443893] [py_test]: Hello3
```

Installing your First Node

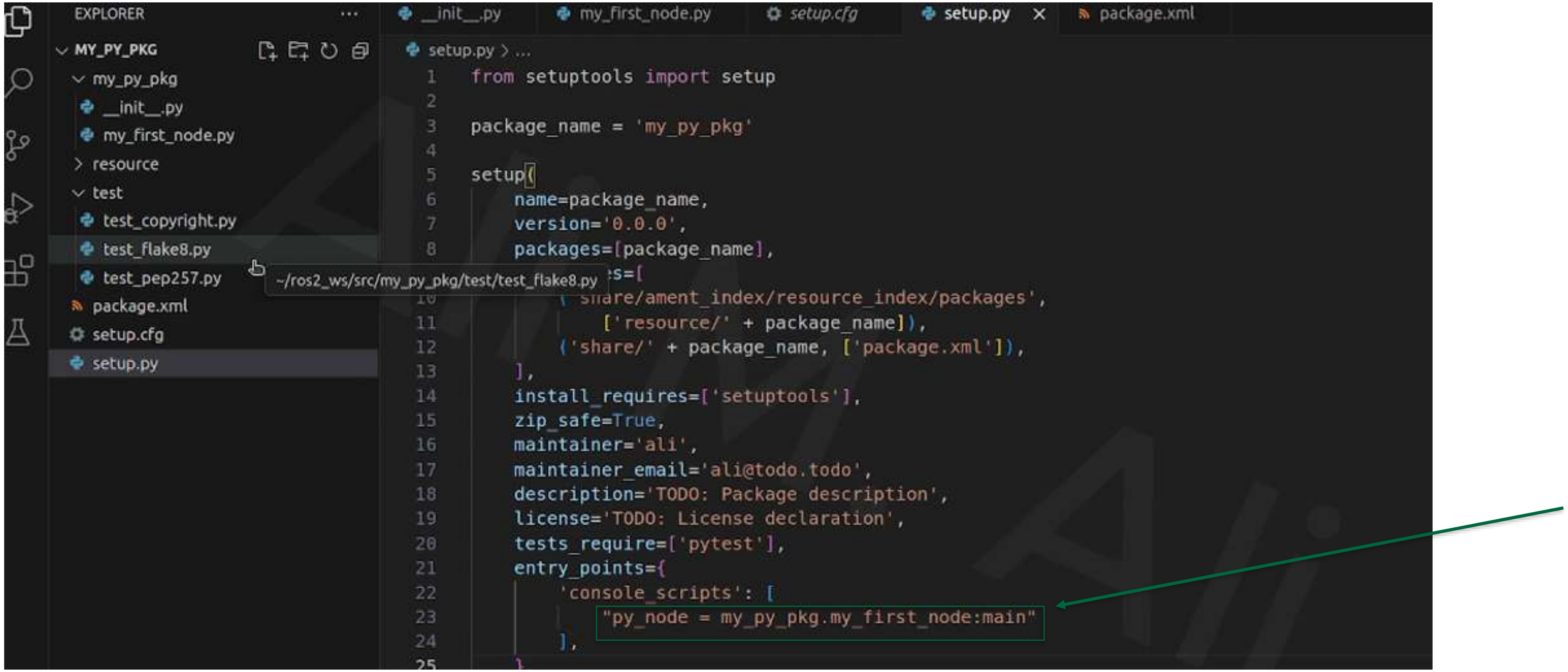
- Go to the setup code inside your IDE



```python
from setuptools import setup

package_name = 'my_py_pkg'

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

Installing your First Node

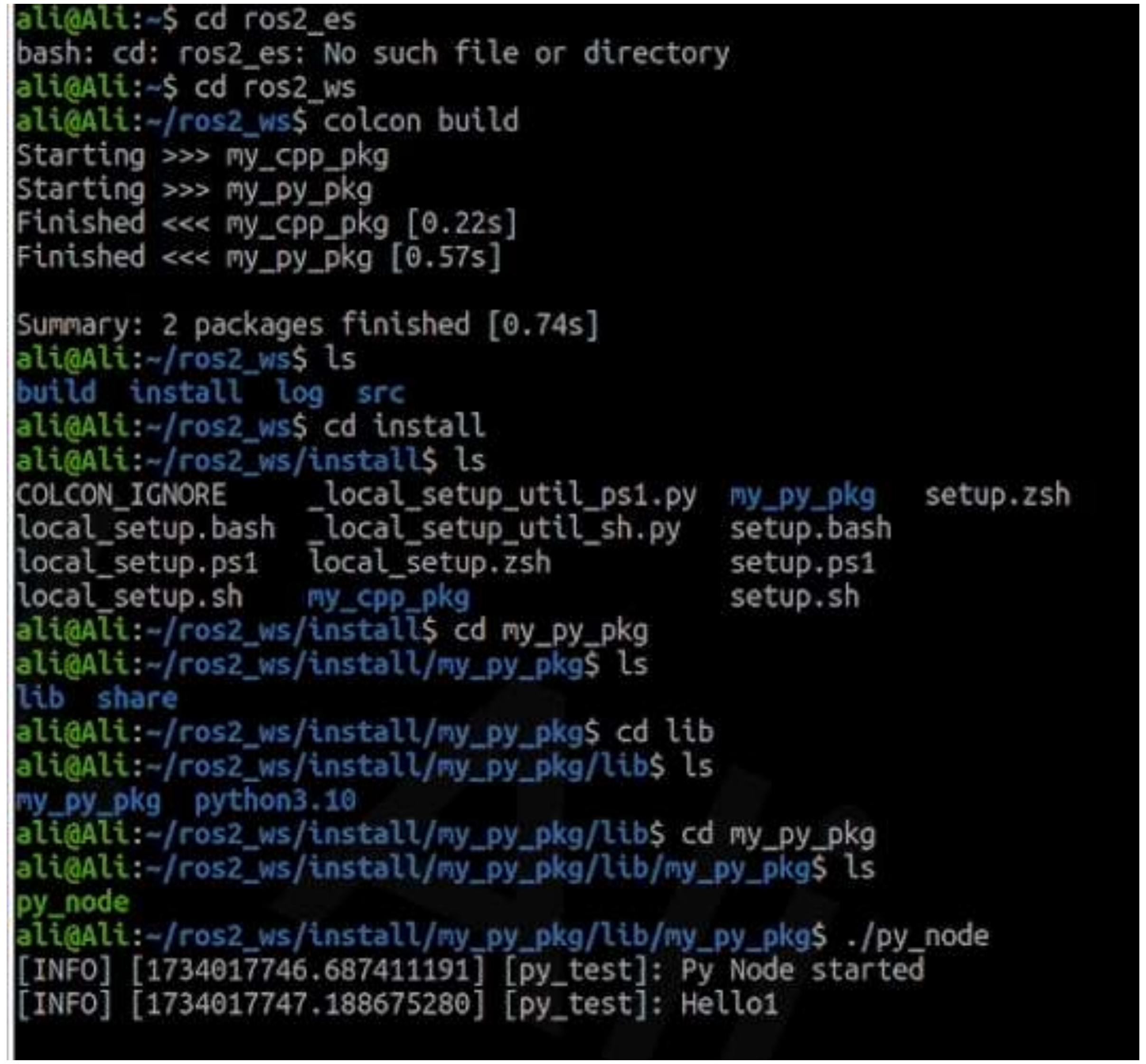
Add this line (see the green arrow). Don't forget to save the setup code.



```
from setuptools import setup

package_name = 'my_py_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
            s=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='ali',
    maintainer_email='ali@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            "py_node = my_py_pkg.my_first_node:main"
        ],
```

Installing your First Node

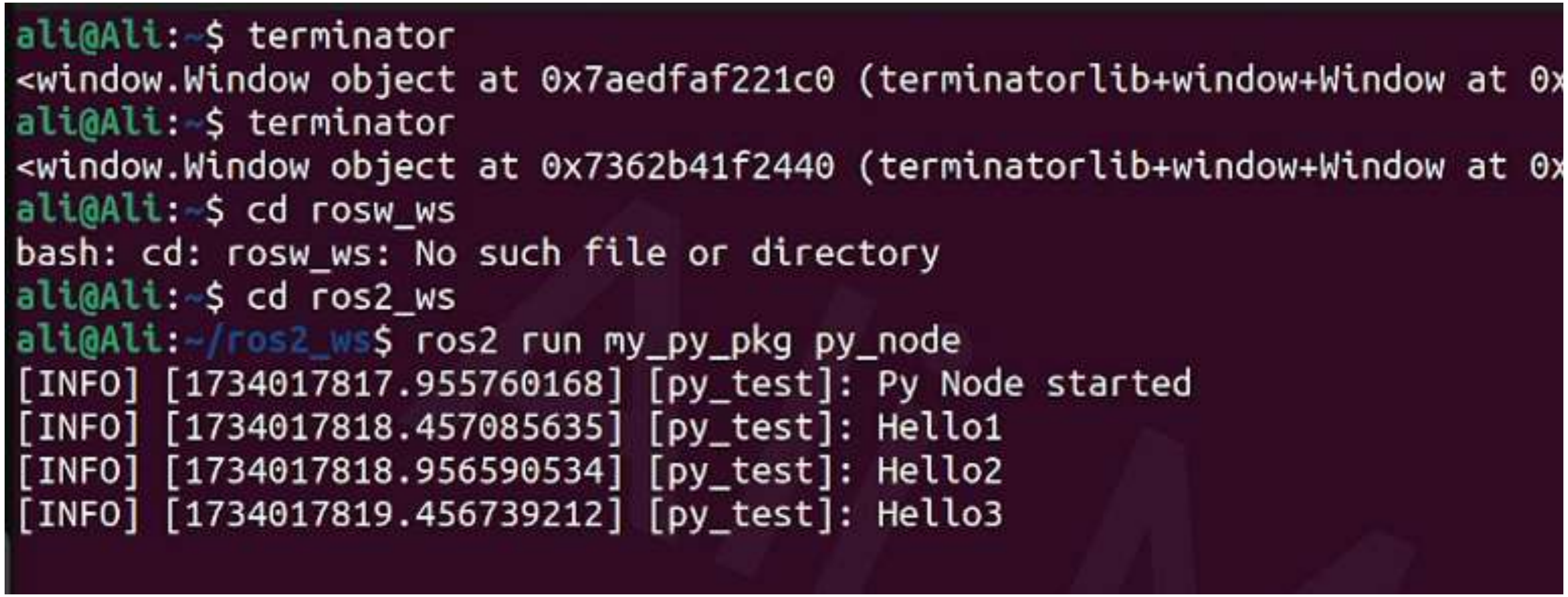
- We need to rebuild the package again after these modifications using Colcon Build



```
ali@Ali:~$ cd ros2_es
bash: cd: ros2_es: No such file or directory
ali@Ali:~$ cd ros2_ws
ali@Ali:~/ros2_ws$ colcon build
Starting >>> my_cpp_pkg
Starting >>> my_py_pkg
Finished <<< my_cpp_pkg [0.22s]
Finished <<< my_py_pkg [0.57s]

Summary: 2 packages finished [0.74s]
ali@Ali:~/ros2_ws$ ls
build  install  log  src
ali@Ali:~/ros2_ws$ cd install
ali@Ali:~/ros2_ws/install$ ls
COLCON_IGNORE      _local_setup_util_ps1.py  my_py_pkg   setup.zsh
local_setup.bash   _local_setup_util_sh.py   setup.bash
local_setup.ps1    local_setup.zsh           setup.ps1
local_setup.sh     my_cpp_pkg                setup.sh
ali@Ali:~/ros2_ws/install$ cd my_py_pkg
ali@Ali:~/ros2_ws/install/my_py_pkg$ ls
lib  share
ali@Ali:~/ros2_ws/install/my_py_pkg$ cd lib
ali@Ali:~/ros2_ws/install/my_py_pkg/lib$ ls
my_py_pkg  python3.10
ali@Ali:~/ros2_ws/install/my_py_pkg/lib$ cd my_py_pkg
ali@Ali:~/ros2_ws/install/my_py_pkg/lib/my_py_pkg$ ls
py_node
ali@Ali:~/ros2_ws/install/my_py_pkg/lib/my_py_pkg$ ./py_node
[INFO] [1734017746.687411191] [py_test]: Py Node started
[INFO] [1734017747.188675280] [py_test]: Hello1
```

Installing your First Node

- Now Finally, we can run our custom build Node



```
ali@Ali:~$ terminator
<window.Window object at 0x7aedfaf221c0 (terminatorlib+window+Window at 0x
ali@Ali:~$ terminator
<window.Window object at 0x7362b41f2440 (terminatorlib+window+Window at 0x
ali@Ali:~$ cd rosw_ws
bash: cd: rosw_ws: No such file or directory
ali@Ali:~$ cd ros2_ws
ali@Ali:~/ros2_ws$ ros2 run my_py_pkg py_node
[INFO] [1734017817.955760168] [py_test]: Py Node started
[INFO] [1734017818.457085635] [py_test]: Hello1
[INFO] [1734017818.956590534] [py_test]: Hello2
[INFO] [1734017819.456739212] [py_test]: Hello3
```