22.Python implementation of ROS2 launch

1. Launch introduction

So far, every time we run a ROS node, we need to open a new terminal and run a command. There are many nodes in the robot system, and it is troublesome to start them like this every time. Is there a way to start all nodes at once? The answer is of course yes, that is the Launch file, which is a script for multi-node startup and configuration in the ROS system.

In ROS2, launch is used for functions such as multi-node startup and configuring program running parameters. The launch file formats of ROS2 include xml, yaml and python formats. This course takes the launch file in python format as an example. Compared with the other two formats, the python format is more flexible:

- Python has numerous function libraries that can be used in startup files;
- ROS2 general launch features and specific launch features are written in Python, so XML and YAML are accessible for launch features that may not be exposed;

Using python language to write ROS2 launch files, the most important thing is to abstract each node, file, script, etc. into an action, and use a unified interface to launch it. The main structure is:

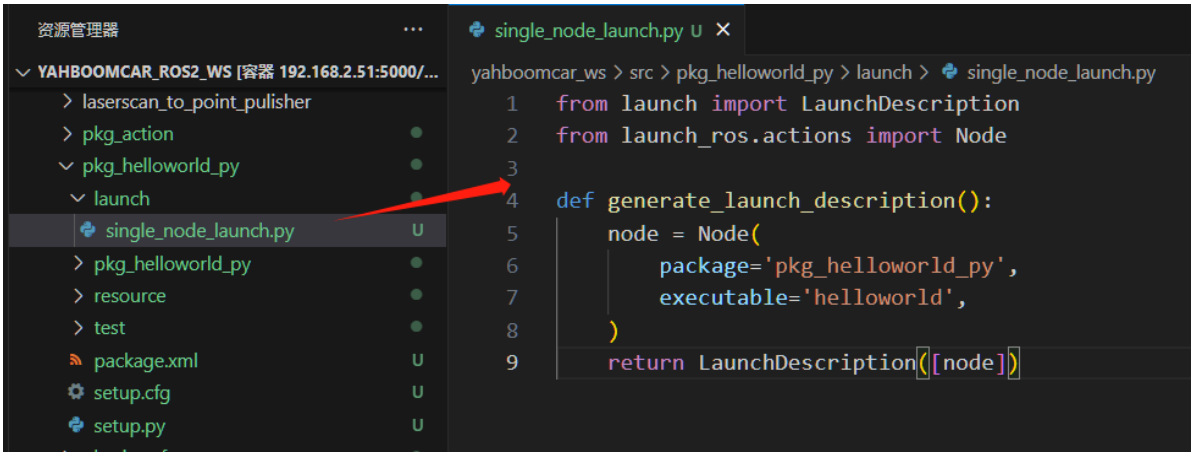
```
def generate_launch_description():
    return LaunchDescription([
        action_1,
        action_2,
        ...
        action_n
    ])
```

2. Write the launch of a single Node node

2.1. New launch file

We create a new launch folder under the path of the pkg_helloworld_py function package created before, then create a new [single_node_launch.py] file in the launch folder, and copy the following content into the file:

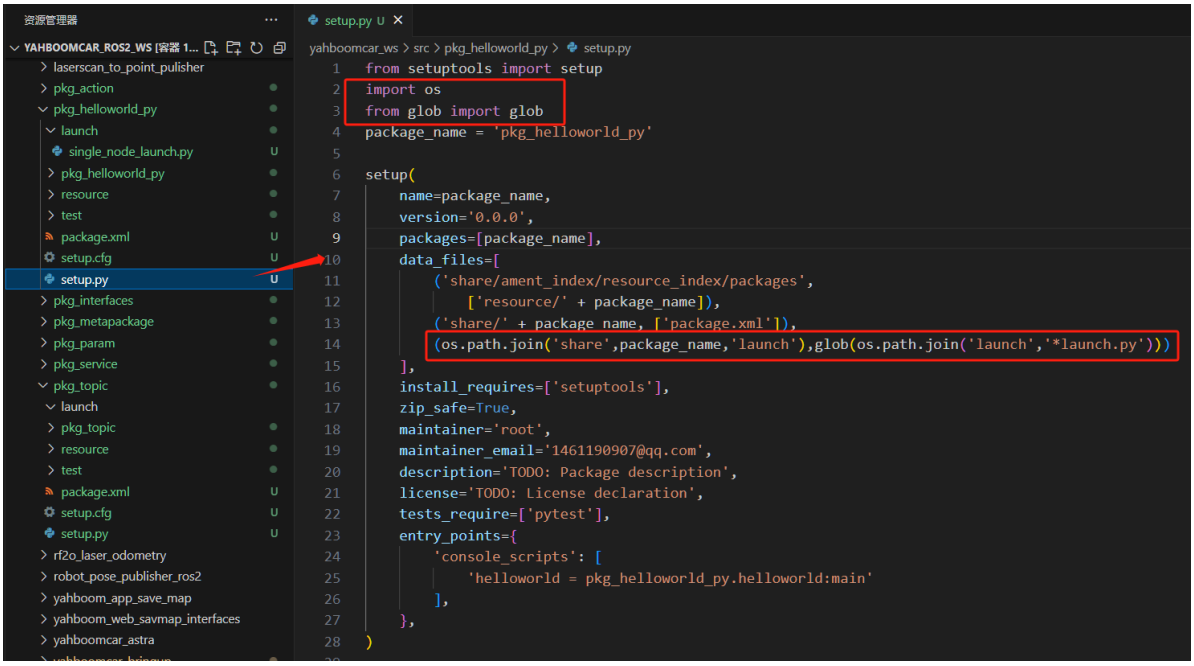
```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    node = Node(
        package='pkg_helloworld_py',
        executable='helloworld',
    )
    return LaunchDescription([node])
```



2.2. Configuration launch file

Launch files are often named with LaunchName_launch.py, where LaunchName is customized and _launch.py is often considered fixed. You need to modify the setup.py file under the function package. The modification content is to add the file under the launch path. Compilation can generate an executed .py file.

```
#1. Import relevant header files
import os
from glob import glob
#2. In the list of data_files, add the launch path and the launch.py file under the path
(os.path.join('share',package_name,'launch'),glob(os.path.join('launch','*launch.py')))
```



2.3. Compile workspace

```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws
colcon build --packages-select pkg_helloworld_py
source install/setup.bash
```

2.4. Run the program

```
ros2 launch pkg_helloworld_py single_node_launch.py
```

```
root@unbutu:~/yahboomcar_ros2_ws/yahboomcar_ws# ros2 launch pkg_helloworld_py single_node_launch.py
[INFO] [launch]: All log files can be found below /root/.ros/log/2023-10-31-03-39-32-011970-unbutu-93543
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [helloworld-1]: process started with pid [93558]
[helloworld-1] [INFO] [1698723572.633101303] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723573.134772801] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723573.636495435] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723574.138219061] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723574.640050048] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723575.141956206] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723575.644302731] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723576.146973557] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723576.649667774] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723577.152369606] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723577.655086381] [helloworld]: Hello World
[helloworld-1] [INFO] [1698723578.157735375] [helloworld]: Hello World
```

2.5. Source code analysis

1. Import related libraries

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

2. Define a function generate_launch_description and return a launch_description

```
def generate_launch_description():
    node = Node(
        package='pkg_helloworld_py',
        executable='helloworld',
    )
    return LaunchDescription([node])
```

A variable node is defined as the return value of a node startup, and the Node function is called to start two important parameters, package and executable.

- package: indicates the function package and represents the name of the function package.
- Executable: Indicates the executed program, the name of the executable program.

Finally, call the LaunchDescription function and pass in the node parameter to execute and return.

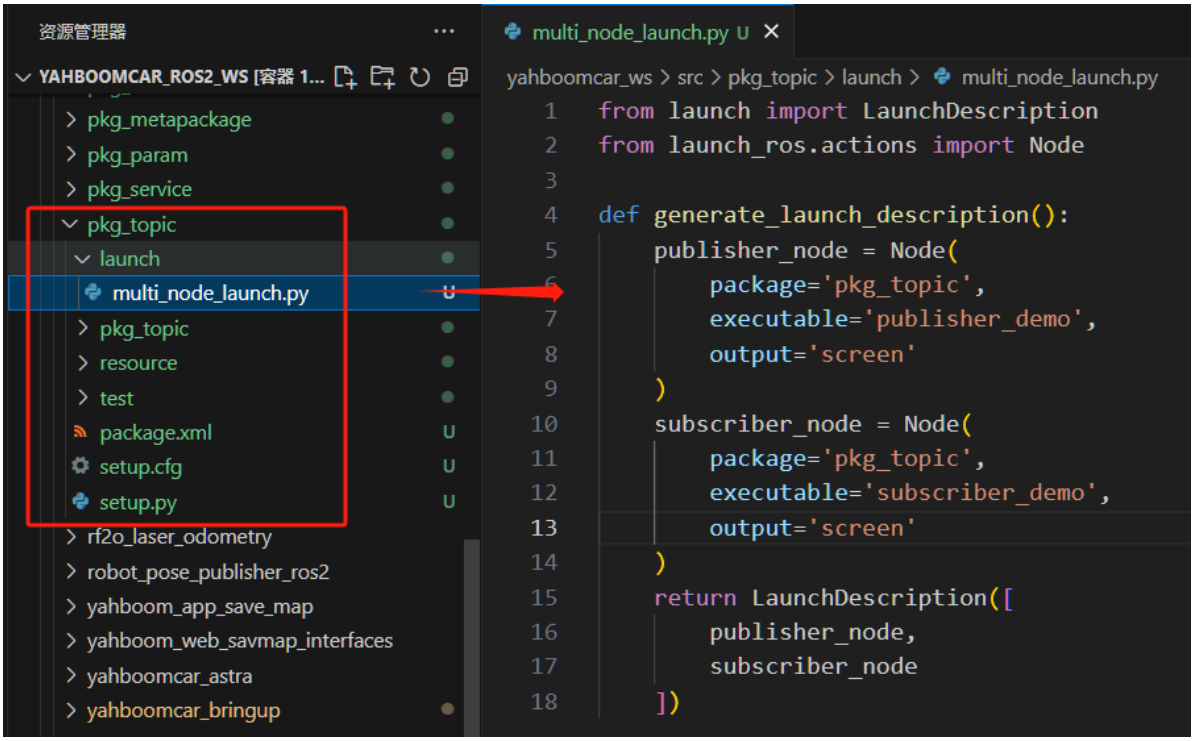
```
return LaunchDescription([node])
```

3. Write the launch of multiple Node nodes

3.1. Create a new launch file

Create a new launch folder under the pkg_topic function package, then create a new [multi_node_launch.py] file in the launch folder and add the following content:

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

def generate_launch_description():
    publisher_node = Node(
        package='pkg_topic',
        executable='publisher_demo',
        output='screen'
    )
    subscriber_node = Node(
        package='pkg_topic',
        executable='subscriber_demo',
        output='screen'
    )
    return LaunchDescription([
        publisher_node,
        subscriber_node
    ])
```



3.2. Compile workspace

```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws
colcon build --packages-select pkg_topic
source install/setup.bash
```

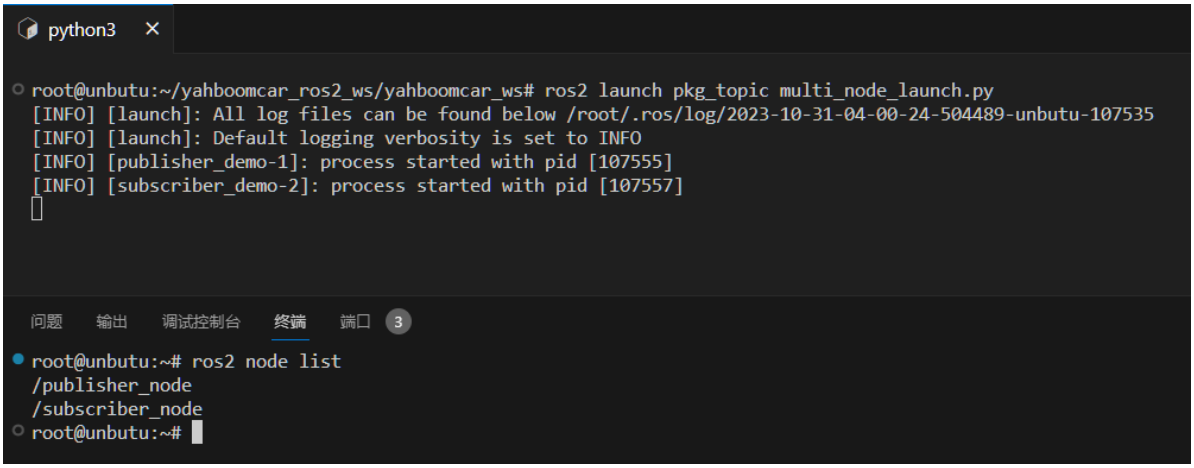
3.3. Run the program

Terminal input

```
ros2 launch pkg_topic multi_node_launch.py
```

The terminal does not print anything. We can check which nodes are started to verify whether the startup is successful. Enter the terminal:

```
ros2 node list
```



3.4. Source code analysis

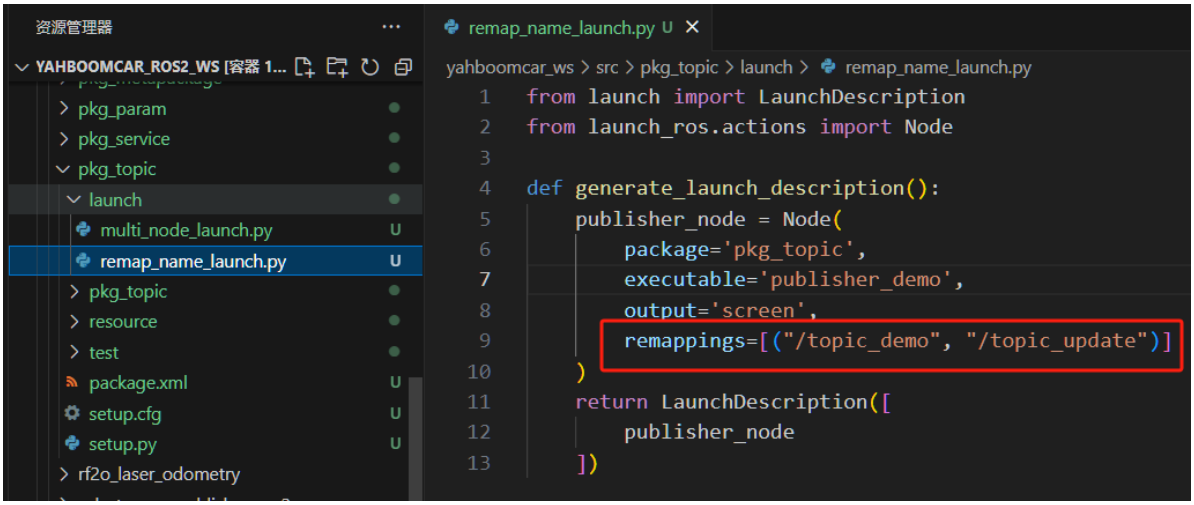
It is roughly the same as simple_node_launch.py, but with one more node.

4. Topic remapping case

4.1. Create a new launch file

Create a new file [remap_name_launch.py] in the same directory as multi_node_launch.py and add the following content:

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

def generate_launch_description():
    publisher_node = Node(
        package='pkg_topic',
        executable='publisher_demo',
        output='screen',
        remappings=[("/topic_demo", "/topic_update")]
    )
    return LaunchDescription([
        publisher_node
    ])
```



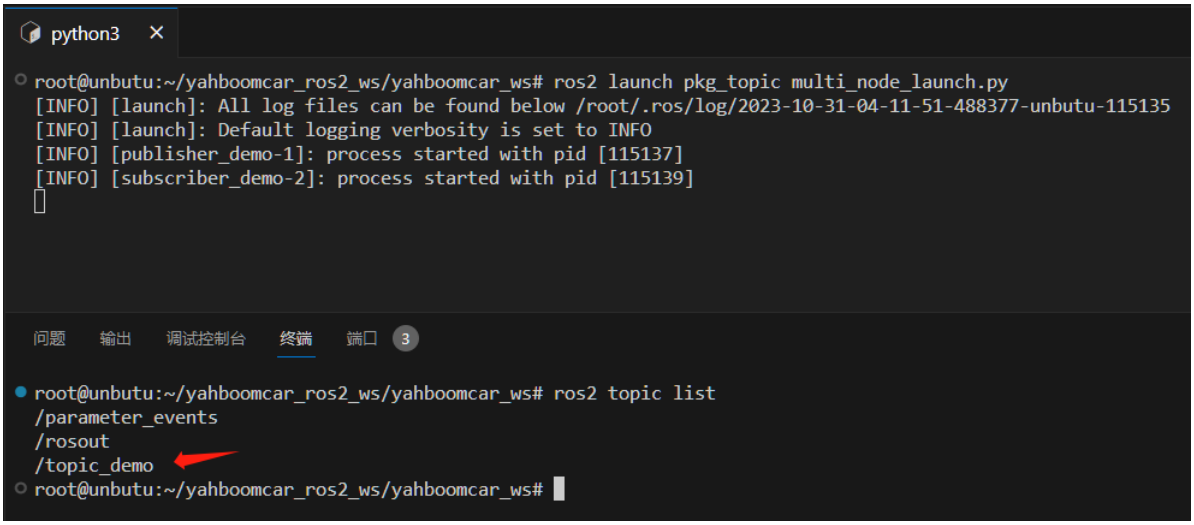
4.2. Compile workspace

```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws
colcon build --packages-select pkg_topic
source install/setup.bash
```

4.3. Run the program

Let's first take a look at what topics the publisher_demo node publishes before there is no remapping topic:

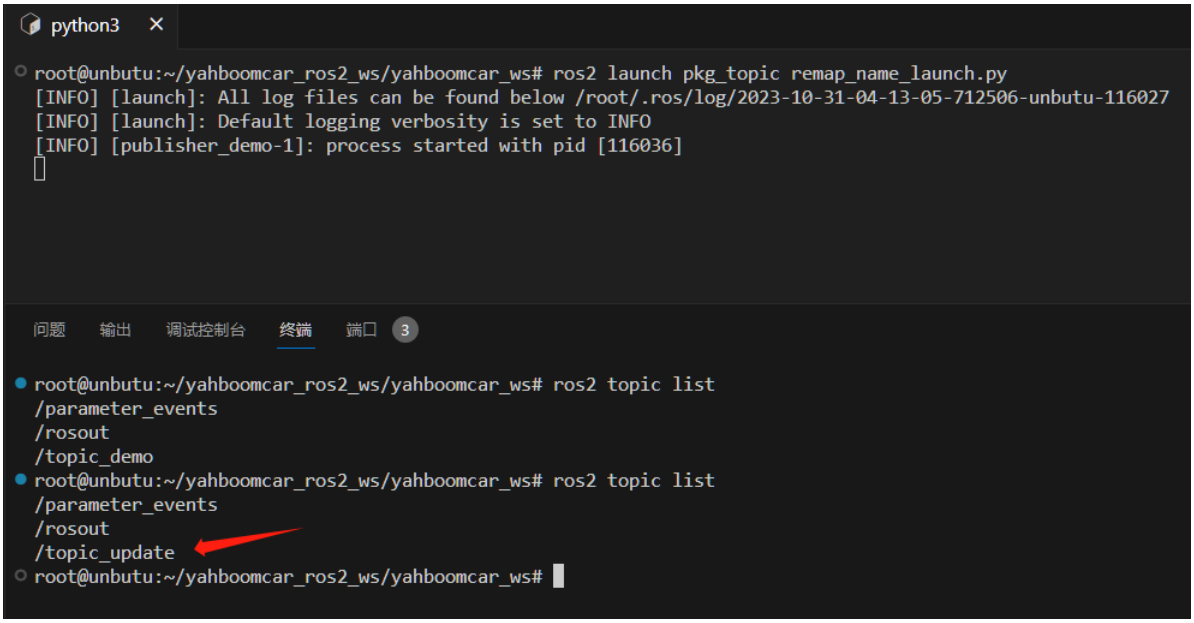
```
ros2 launch pkg_topic multi_node_launch.py
ros2 topic list
```



The topic here is【/topic_demo】

Run the program after remapping the topic again and see the changes:

```
ros2 launch pkg_topic remap_name_launch.py
ros2 topic list
```



As can be seen from the above figure, the remapped topic name is [/topic_update]

4.4. Source code analysis

Mainly adding the following parts:

```
remappings=[("/topic_demo", "/topic_update")]
```

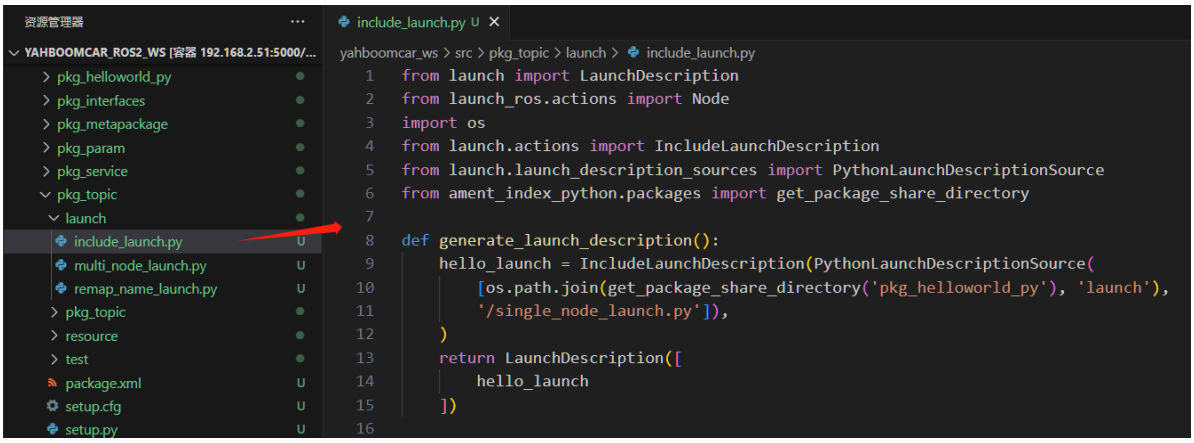
This is to remap the original /topic_demo topic to /topic_update

5. Launch file starts launch file case

5.1. Create a new launch file

Create a new file [include_launch.py] in the same directory as multi_node_launch.py and add the following content:

```
from launch import LaunchDescription
from launch_ros.actions import Node
import os
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():
    hello_launch = IncludeLaunchDescription(PythonLaunchDescriptionSource(
        [os.path.join(get_package_share_directory('pkg_helloworld_py'),
'launch'),
        '/single_node_launch.py']),
    )
    return LaunchDescription([
        hello_launch
    ])
```



5.2. Compile workspace

```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws
colcon build --packages-select pkg_topic
source install/setup.bash
```

5.3. Run the program

```
ros2 launch pkg_topic include_launch.py
```

```
root@unbutu:~/yahboomcar_ros2_ws/yahboomcar_ws# ros2 launch pkg_topic include_launch.py
[INFO] [launch]: All log files can be found below /root/.ros/log/2023-10-31-06-06-21-772686-unbutu-183550
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [helloworld-1]: process started with pid [183561]
[helloworld-1] [INFO] [1698732382.667452115] [helloworld]: Hello World
[helloworld-1] [INFO] [1698732383.169843271] [helloworld]: Hello World
[helloworld-1] [INFO] [1698732383.671496257] [helloworld]: Hello World
[helloworld-1] [INFO] [1698732384.173235837] [helloworld]: Hello World
[helloworld-1] [INFO] [1698732384.674837845] [helloworld]: Hello World
```

You can run the program in the pkg_helloworld_py package in the pkg_topic package.

5.4. Source code analysis

```
from launch import LaunchDescription
from launch_ros.actions import Node
import os
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():
    hello_launch = IncludeLaunchDescription(PythonLaunchDescriptionSource(
        [os.path.join(get_package_share_directory('pkg_helloworld_py'),
'launch'),
        '/single_node_launch.py']),
    )
    return LaunchDescription([
        hello_launch
    ])
```

- os.path.join(get_package_share_directory('pkg_helloworld_py'): Get the location of the function package, where 'pkg_helloworld_py' is the name of the function package;
- launch'): indicates the folder where the launch file is stored under the function package;
- /single_node_launch.py' : Indicates that the name of the launch file under the launch file in the function package folder is /single_node_launch.py in the example.

6. Slightly complex launch file example

This case mainly shows how to write a complex launch file, and the function of the program can be ignored.

6.1. Create a new launch file

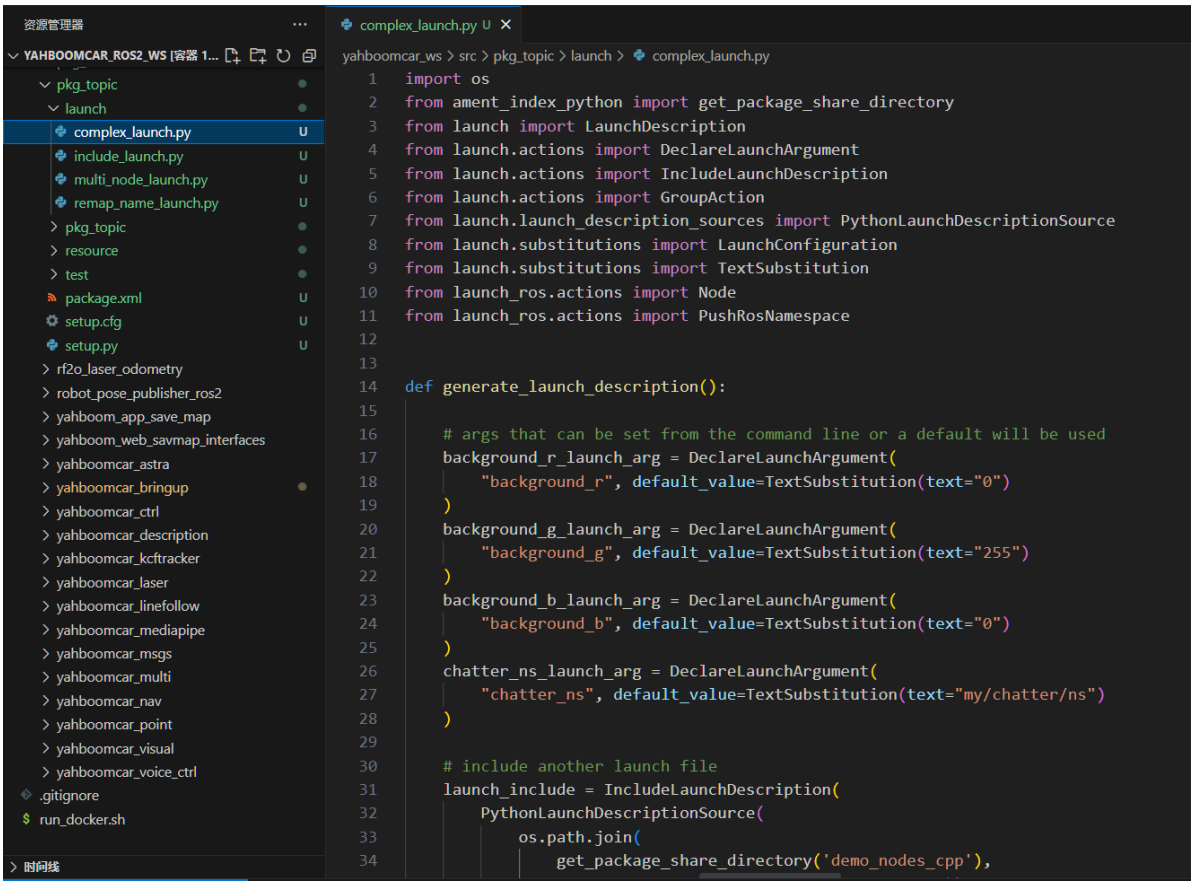
Create a new file [complex_launch.py] in the same directory as multi_node_launch.py and add the following content:

Note: This case will display the little turtle window. Before starting the program, please make sure that the docker GUI display is turned on, otherwise the little turtle window cannot be displayed

```
import os
from ament_index_python import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.actions import IncludeLaunchDescription
from launch.actions import GroupAction
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration
from launch.substitutions import TextSubstitution
from launch_ros.actions import Node
from launch_ros.actions import PushRosNamespace
```

```
def generate_launch_description():

    # args that can be set from the command line or a default will be used
    background_r_launch_arg = DeclareLaunchArgument(
        "background_r", default_value=TextSubstitution(text="0")
    )
    background_g_launch_arg = DeclareLaunchArgument(
        "background_g", default_value=TextSubstitution(text="255")
    )
    background_b_launch_arg = DeclareLaunchArgument(
        "background_b", default_value=TextSubstitution(text="0")
    )
    chatter_ns_launch_arg = DeclareLaunchArgument(
        "chatter_ns", default_value=TextSubstitution(text="my/chatter/ns")
    )

    # include another launch file
    launch_include = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            os.path.join(
                get_package_share_directory('demo_nodes_cpp'),
                'launch/topics/talker_listener.launch.py'))
    )
    # include another launch file in the chatter_ns namespace
    launch_include_with_namespace = GroupAction(
        actions=[
            # push-ros-namespace to set namespace of included nodes
            PushRosNamespace(LaunchConfiguration('chatter_ns')),
            IncludeLaunchDescription(
                PythonLaunchDescriptionSource(
                    os.path.join(
                        get_package_share_directory('demo_nodes_cpp'),
                        'launch/topics/talker_listener.launch.py'))
            ),
        ]
    )

    # start a turtlesim_node in the turtlesim1 namespace
    turtlesim_node = Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
        )

    # start another turtlesim_node in the turtlesim2 namespace
    # and use args to set parameters
    turtlesim_node_with_parameters = Node(
            package='turtlesim',
            namespace='turtlesim2',
            executable='turtlesim_node',
            name='sim',
            parameters=[{
```

```
                "background_r": LaunchConfiguration('background_r'),
                "background_g": LaunchConfiguration('background_g'),
                "background_b": LaunchConfiguration('background_b'),
            }]
        )

    # perform remap so both turtles listen to the same command topic
    forward_turtlesim_commands_to_second_turtlesim_node = Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        )

    return LaunchDescription([
        background_r_launch_arg,
        background_g_launch_arg,
        background_b_launch_arg,
        chatter_ns_launch_arg,
        launch_include,
        launch_include_with_namespace,
        turtlesim_node,
        turtlesim_node_with_parameters,
        forward_turtlesim_commands_to_second_turtlesim_node,
    ])
```



6.2. Compile workspace

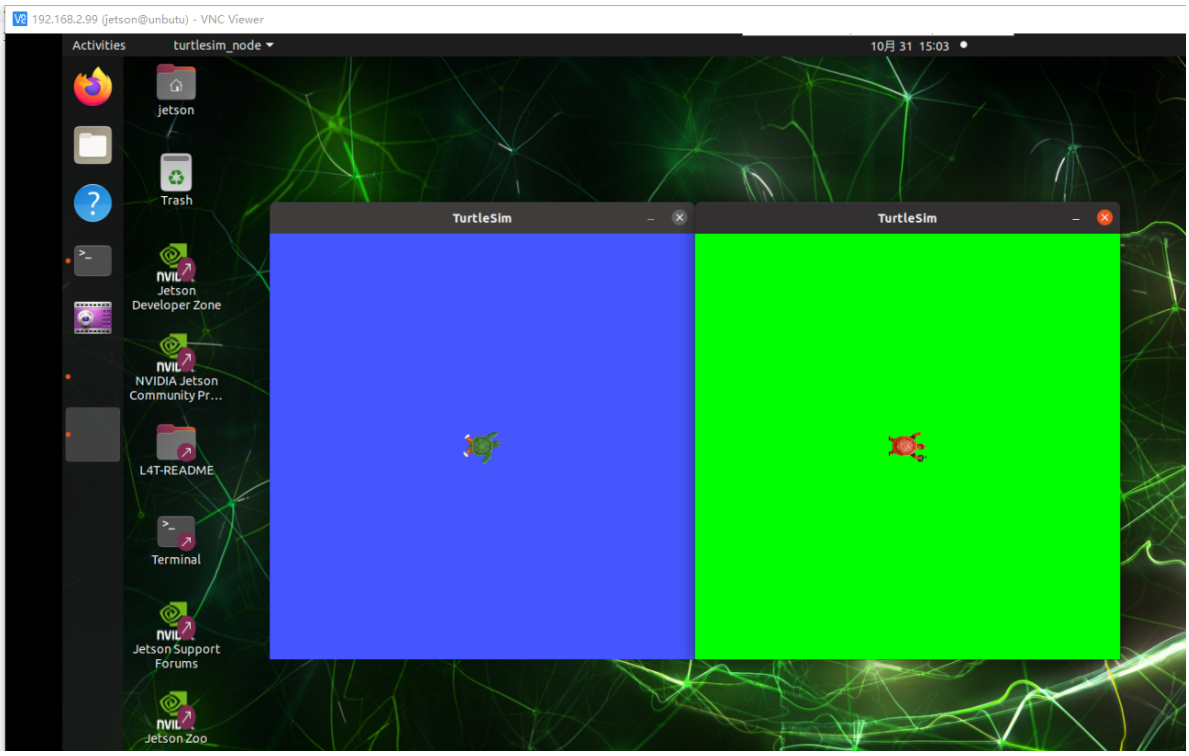
```
cd ~/yahboomcar_ros2_ws/yahboomcar_ws
colcon build --packages-select pkg_topic
source install/setup.bash
```

6.3. Run the program

Terminal input:

```
ros2 launch pkg_topic complex_launch.py
```

Two little turtles will be displayed on the host's vnc.



6.4. Program description

The program mainly starts:

1. The talker_listener node of demo_nodes_cpp,
2. talker_listener node with namespace
3. Little turtle 1 with turtlesim1 as namespace
4. Little turtle 2 with turtlesim2 as namespace
5. Perform remapping so that both turtles hear the same command theme