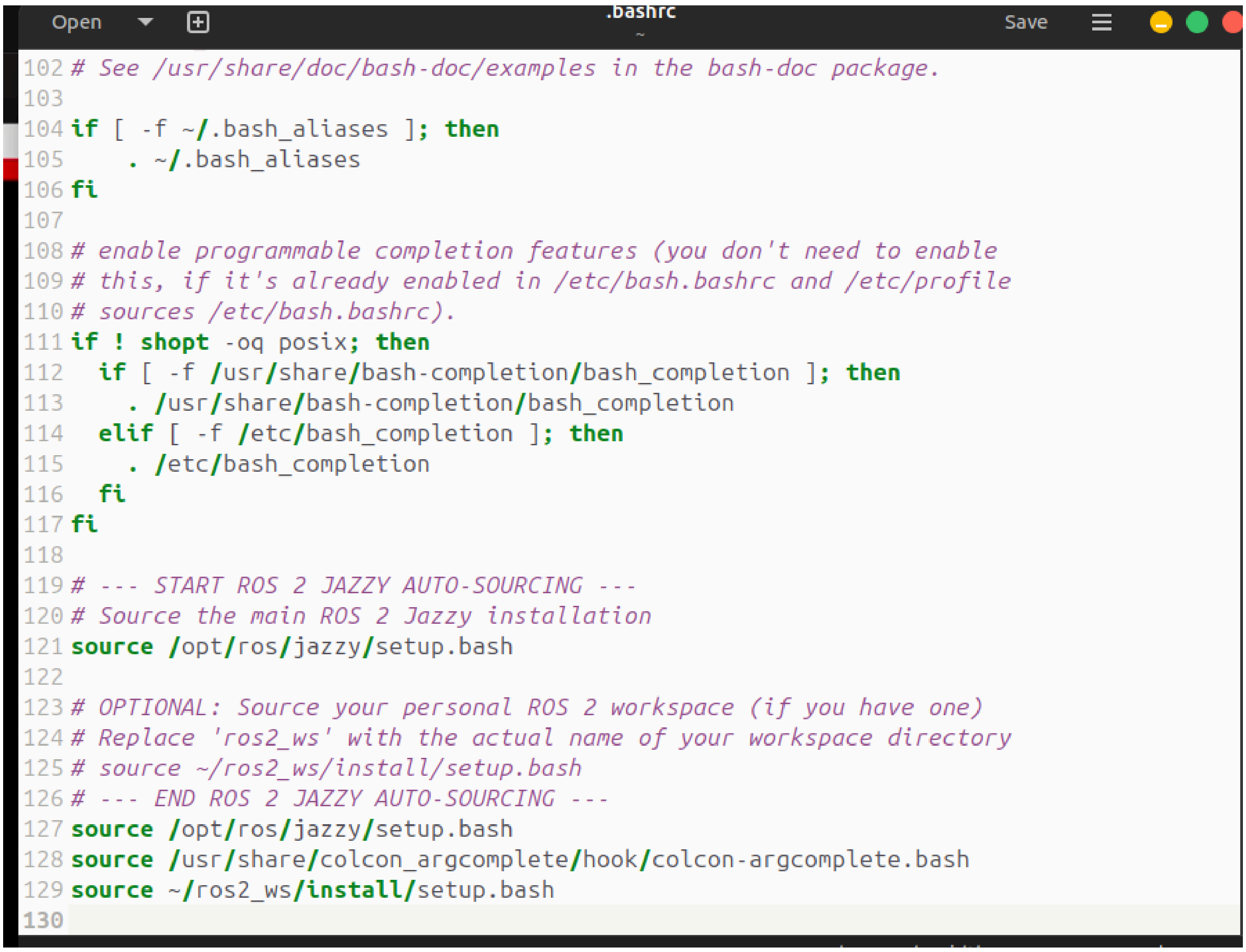
Workspace and packages preparations "steps before building nodes"

First: prepare the the workspace by initializing the directory

create a basic directory structure for the ROS2 Porject
1- mkdir ros2_ws
2- cd ros2_ws
3-mkdir src
4-use colcon build
4-cd install
5-ls
6-4-source the setup.bash inside the install file inside the bashrc file



```
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

# --- START ROS 2 JAZZY AUTO-SOURCING ---
# Source the main ROS 2 Jazzy installation
source /opt/ros/jazzy/setup.bash

# OPTIONAL: Source your personal ROS 2 workspace (if you have one)
# Replace 'ros2_ws' with the actual name of your workspace directory
# source ~/ros2_ws/install/setup.bash
# --- END ROS 2 JAZZY AUTO-SOURCING ---
source /opt/ros/jazzy/setup.bash
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
source ~/ros2_ws/install/setup.bash

```

Second Step: create the custom Package

you need to create a new package inside the src directory to house the nodes source code
1-cd ~/ros2_ws/src

2- ros2 pkg create pkgs_name --build-type ament_python --dependencies rclpy

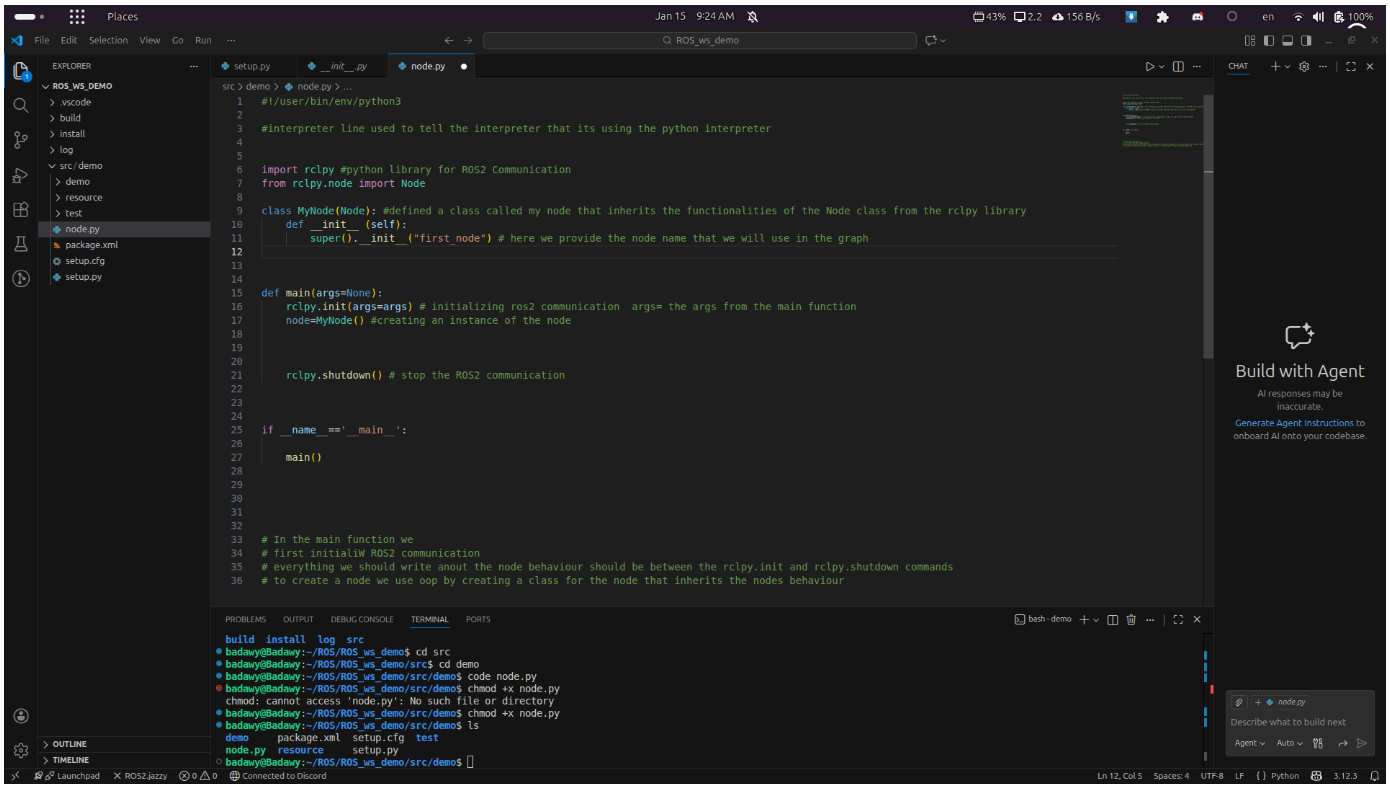
What is inside the src?
package.xml contains details about the authon of the package and the dependencies used in the package
setup.cfg contains setup configs for the ros2 system
Setup.py: contains main info about the package

We then get into the folder that has the name as the package and write the nodes code inside it

Third Step: Write the nodes code

1-get inside the directory that has the same name as the package
2- create a python file
touch node.py
3-make it executable
chmod +x node.py

inside the code we make a class for the node that inherits from the Node class inside the rclpy library inside this class we define the behaviour of the node we are doing

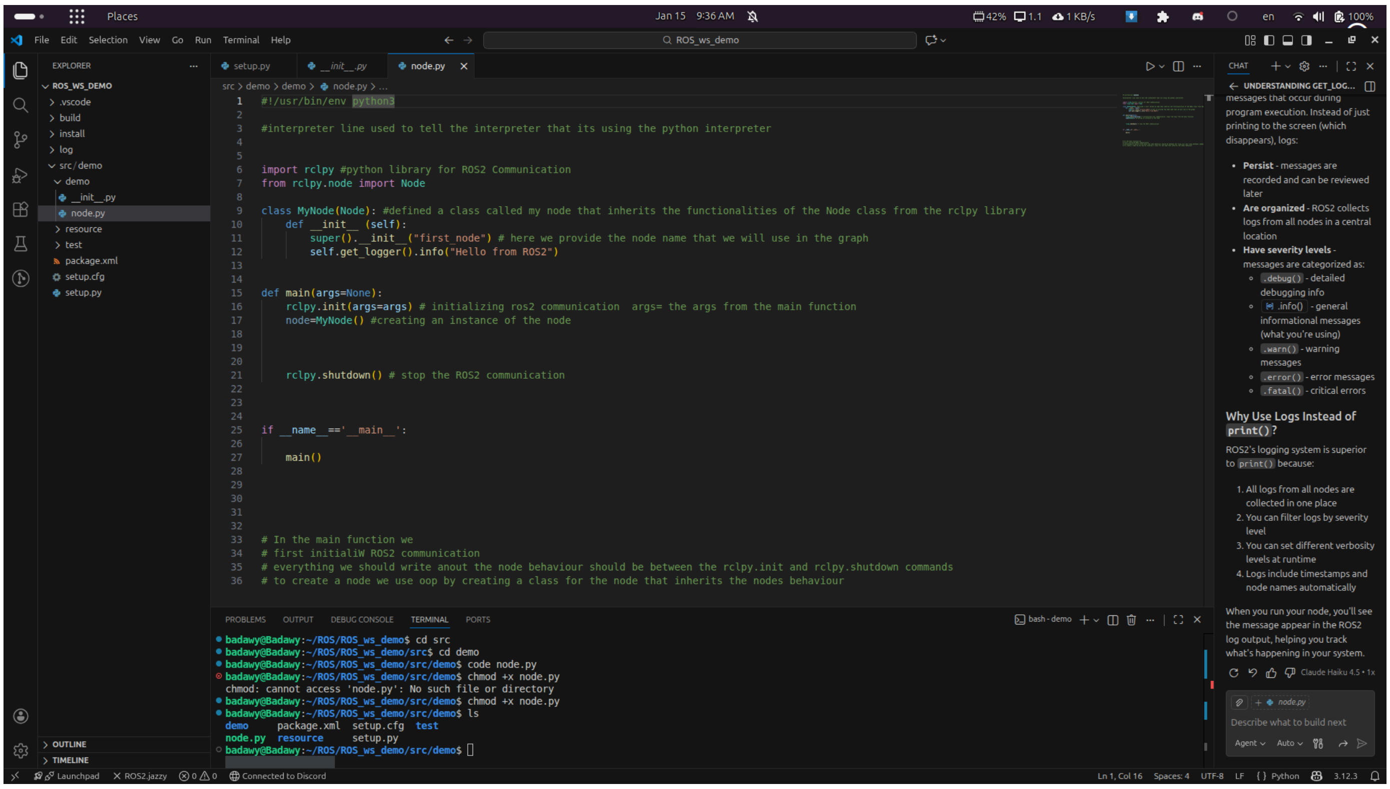


```python
#!/user/bin/env/python3

#interpreter line used to tell the interpreter that its using the python interpreter


import rclpy #python library for ROS2 Communication
from rclpy.node import Node

class MyNode(Node): #defined a class called my node that inherits the functionalities of the Node class from the rclpy library
    def __init__ (self):
        super().__init__("first_node") # here we provide the node name that we will use in the graph



def main(args=None):
    rclpy.init(args=args) # initializing ros2 communication  args= the args from the main function
    node=MyNode() #creating an instance of the node



    rclpy.shutdown() # stop the ROS2 communication



if __name__=='__main__':

    main()






# In the main function we
# first initialiW ROS2 communication
# everything we should write anout the node behaviour should be between the rclpy.init and rclpy.shutdown commands
# to create a node we use oop by creating a class for the node that inherits the nodes behaviour
```

```
build  install  log  src
badawy@Badawy:~/ROS/ROS_ws_demo$ cd src
badawy@Badawy:~/ROS/ROS_ws_demo/src$ cd demo
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ code node.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ chmod +x node.py
chmod: cannot access 'node.py': No such file or directory
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ chmod +x node.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ ls
demo     package.xml  setup.cfg  test
node.py  resource     setup.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ 
```

This is the basic tempelate for inializing a ROS2 node that does nothing.
after initializing the node we must then add some behaviours inside
node class

1-



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

#interpreter line used to tell the interpreter that its using the python interpreter


import rclpy #python library for ROS2 Communication
from rclpy.node import Node

class MyNode(Node): #defined a class called my node that inherits the functionalities of the Node class from the rclpy library
    def __init__ (self):
        super().__init__("first_node") # here we provide the node name that we will use in the graph
        self.get_logger().info("Hello from ROS2")


def main(args=None):
    rclpy.init(args=args) # initializing ros2 communication  args= the args from the main function
    node=MyNode() #creating an instance of the node



    rclpy.shutdown() # stop the ROS2 communication



if __name__=='__main__':

    main()






# In the main function we
# first initialiW ROS2 communication
# everything we should write anout the node behaviour should be between the rclpy.init and rclpy.shutdown commands
# to create a node we use oop by creating a class for the node that inherits the nodes behaviour
```

```
badawy@Badawy:~/ROS/ROS_ws_demo$ cd src
badawy@Badawy:~/ROS/ROS_ws_demo/src$ cd demo
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ code node.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ chmod +x node.py
chmod: cannot access 'node.py': No such file or directory
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ chmod +x node.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ ls
demo     package.xml  setup.cfg  test
node.py  resource     setup.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ 
```

UNDERSTANDING GET_LOG...

messages that occur during program execution. Instead of just printing to the screen (which disappears), logs:

- **Persist** - messages are recorded and can be reviewed later
- **Are organized** - ROS2 collects logs from all nodes in a central location
- **Have severity levels** - messages are categorized as:
  - `.debug()` - detailed debugging info
  - `.info()` - general informational messages (what you're using)
  - `.warn()` - warning messages
  - `.error()` - error messages
  - `.fatal()` - critical errors

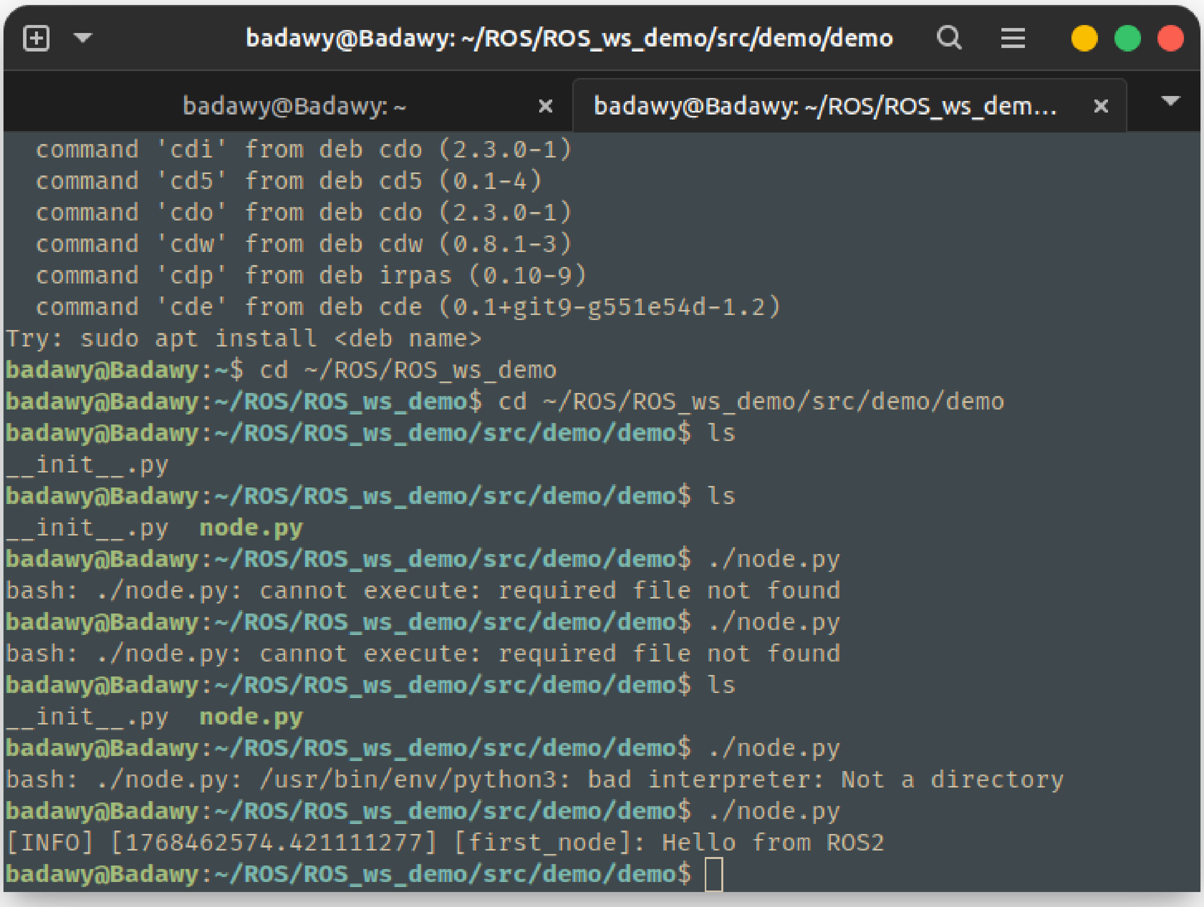
Why Use Logs Instead of `print()`?

ROS2's logging system is superior to `print()` because:

1. All logs from all nodes are collected in one place
2. You can filter logs by severity level
3. You can set different verbosity levels at runtime
4. Logs include timestamps and node names automatically

When you run your node, you'll see the message appear in the ROS2 log output, helping you track what's happening in your system.

if we run this code as an executable it prints out the logs in the terminal



badawy@Badawy: ~/ROS/ROS_ws_demo/src/demo/demo
badawy@Badawy: ~
badawy@Badawy: ~/ROS/ROS_ws_dem...

```
  command 'cdi' from deb cdo (2.3.0-1)
  command 'cd5' from deb cd5 (0.1-4)
  command 'cdo' from deb cdo (2.3.0-1)
  command 'cdw' from deb cdw (0.8.1-3)
  command 'cdp' from deb irpas (0.10-9)
  command 'cde' from deb cde (0.1+git9-g551e54d-1.2)
Try: sudo apt install <deb name>
badawy@Badawy:~$ cd ~/ROS/ROS_ws_demo
badawy@Badawy:~/ROS/ROS_ws_demo$ cd ~/ROS/ROS_ws_demo/src/demo/demo
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo/demo$ ls
__init__.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo/demo$ ls
__init__.py  node.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo/demo$ ./node.py
bash: ./node.py: cannot execute: required file not found
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo/demo$ ./node.py
bash: ./node.py: cannot execute: required file not found
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo/demo$ ls
__init__.py  node.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo/demo$ ./node.py
bash: ./node.py: /usr/bin/env/python3: bad interpreter: Not a directory
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo/demo$ ./node.py
[INFO] [1768462574.421111277] [first_node]: Hello from ROS2
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo/demo$ 
```

the problem about this executable here that it only prints out the loggs one time and then shutdowns the system but we need a continous loop system

In ROS2, get_logger().info() is a method that outputs informational messages to the log system. In your code:

self.get_logger().info("Hello from ROS2")
This prints "Hello from ROS2" to the logs when your node runs.

What is a Log?
A log is a record of events or messages that occur during program execution. Instead of just printing to the screen (which disappears), logs:

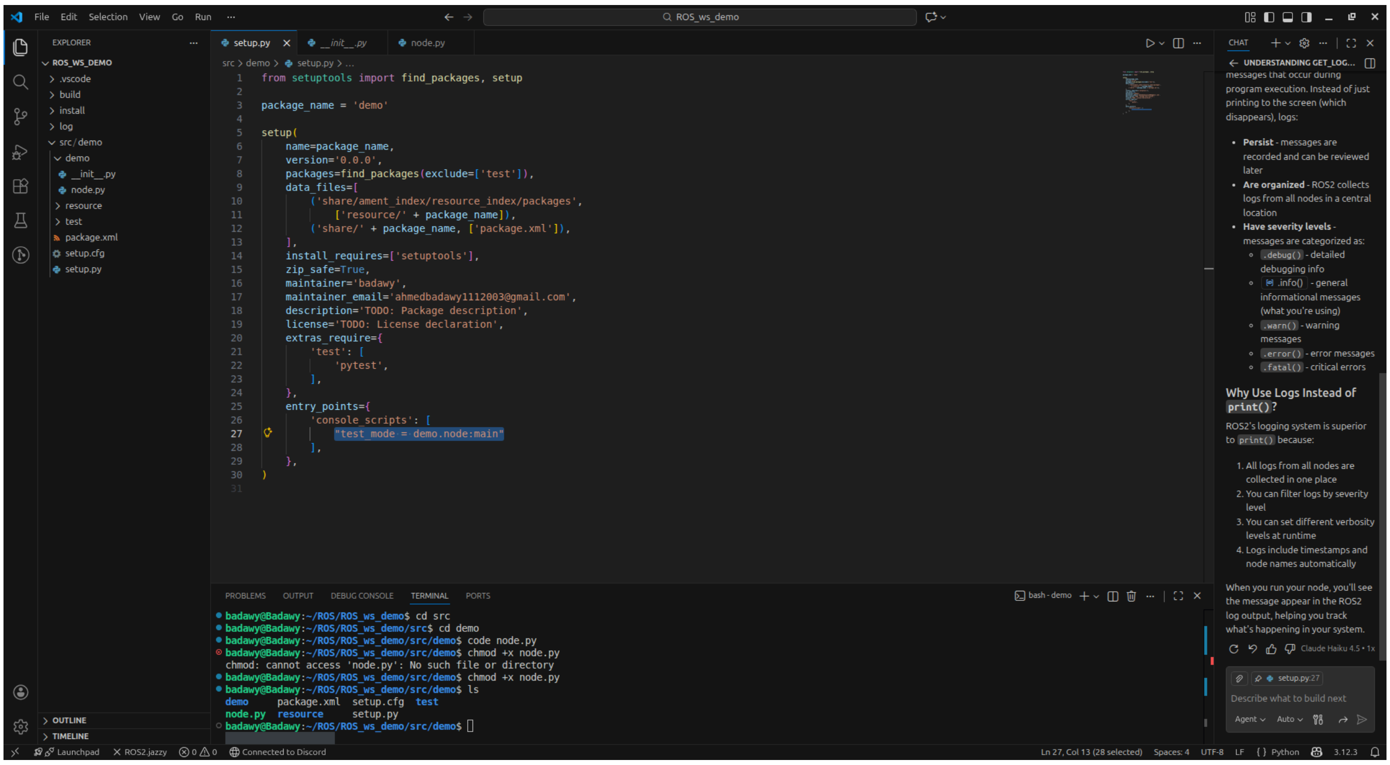
Persist - messages are recorded and can be reviewed later
Are organized - ROS2 collects logs from all nodes in a central location
Have severity levels - messages are categorized as:
.debug() - detailed debugging info
.info() - general informational messages (what you're using)
.warn() - warning messages
.error() - error messages
.fatal() - critical errors
Why Use Logs Instead of print()?
ROS2's logging system is superior to print() because:

All logs from all nodes are collected in one place
You can filter logs by severity level
You can set different verbosity levels at runtime
Logs include timestamps and node names automatically
When you run your node, you'll see the message appear in the ROS2 log output, helping you track what's happening in your system.

2-After adding the rclpy.spin() function the node keeps working inside the terminal and only stops when i interrupt it

Important Note

in the code above we only made the node to act as a python script so its not working as a ROS2 functionale node so we must install the node by adding it into the entry points inside the setup.py file



```python
from setuptools import find_packages, setup

package_name = 'demo'

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='badawy',
    maintainer_email='ahmedbadawy1112003@gmail.com',
    description='TODO: Package description',
    license='TODO: License declaration',
    extras_require={
        'test': [
            'pytest',
        ],
    },
    entry_points={
        'console_scripts': [
            "test_mode = demo.node:main"
        ],
    },
)
```

```
badawy@Badawy:~/ROS/ROS_ws_demo$ cd src
badawy@Badawy:~/ROS/ROS_ws_demo/src$ cd demo
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ code node.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ chmod +x node.py
chmod: cannot access 'node.py': No such file or directory
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ chmod +x node.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ ls
demo     package.xml  setup.cfg  test
node.py  resource     setup.py
badawy@Badawy:~/ROS/ROS_ws_demo/src/demo$ 
```

messages that occur during program execution. Instead of just printing to the screen (which disappears), logs:

- **Persist** - messages are recorded and can be reviewed later
- **Are organized** - ROS2 collects logs from all nodes in a central location
- **Have severity levels** - messages are categorized as:
  - `.debug()` - detailed debugging info
  - `.info()` - general informational messages (what you're using)
  - `.warn()` - warning messages
  - `.error()` - error messages
  - `.fatal()` - critical errors

Why Use Logs Instead of `print()`?

ROS2's logging system is superior to `print()` because:

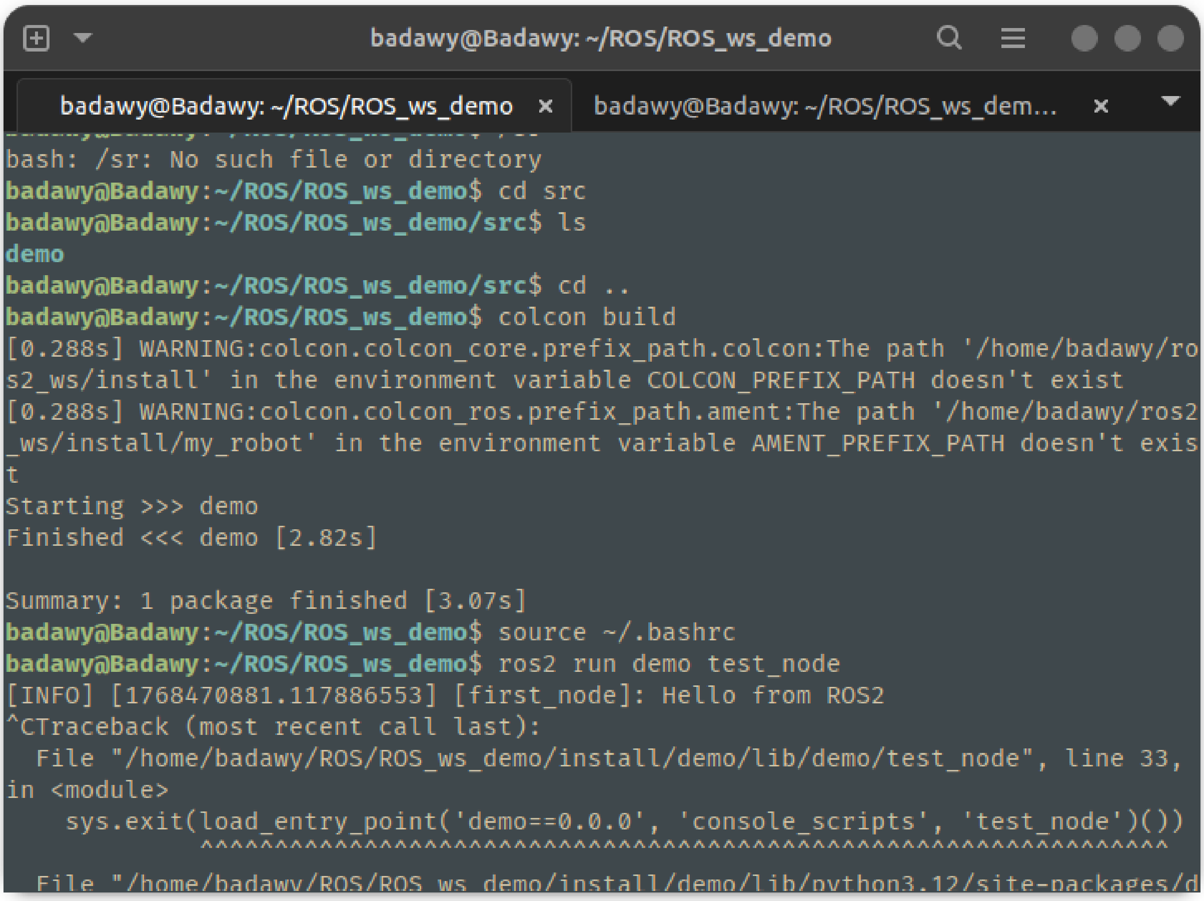
1. All logs from all nodes are collected in one place
2. You can filter logs by severity level
3. You can set different verbosity levels at runtime
4. Logs include timestamps and node names automatically

When you run your node, you'll see the message appear in the ROS2 log output, helping you track what's happening in your system.

After installing the node file inside the setup.py file we must go to the src directory and build the workspace again using colcon build and source the workspace again

colcon build
source ~/.bashrc

now if i use ros2 run "node name inside the setup.py file "



```
badawy@Badawy: ~/ROS/ROS_ws_demo
badawy@Badawy: ~/ROS/ROS_ws_demo    badawy@Badawy: ~/ROS/ROS_ws_dem...
bash: /sr: No such file or directory
badawy@Badawy:~/ROS/ROS_ws_demo$ cd src
badawy@Badawy:~/ROS/ROS_ws_demo/src$ ls
demo
badawy@Badawy:~/ROS/ROS_ws_demo/src$ cd ..
badawy@Badawy:~/ROS/ROS_ws_demo$ colcon build
[0.288s] WARNING:colcon.colcon_core.prefix_path.colcon:The path '/home/badawy/ro
s2_ws/install' in the environment variable COLCON_PREFIX_PATH doesn't exist
[0.288s] WARNING:colcon.colcon_ros.prefix_path.ament:The path '/home/badawy/ros2
_ws/install/my_robot' in the environment variable AMENT_PREFIX_PATH doesn't exis
t
Starting >>> demo
Finished <<< demo [2.82s]

Summary: 1 package finished [3.07s]
badawy@Badawy:~/ROS/ROS_ws_demo$ source ~/.bashrc
badawy@Badawy:~/ROS/ROS_ws_demo$ ros2 run demo test_node
[INFO] [1768470881.117886553] [first_node]: Hello from ROS2
^CTraceback (most recent call last):
  File "/home/badawy/ROS/ROS_ws_demo/install/demo/lib/demo/test_node", line 33,
in <module>
    sys.exit(load_entry_point('demo==0.0.0', 'console_scripts', 'test_node')())
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/badawy/ROS/ROS_ws_demo/install/demo/lib/python3.12/site-packages/d
```

for this node we have three different names
file name inside the src file
node name inside the class in the file
and the executable name inside the setup file
its better to have the same name for all

every time u change the python code for the node you must build the project and source the bashrc file

you can skip this by running colcon build --symlink-install
so evey time you change the python code you dont need to build the project * u still need to source the bashrc*

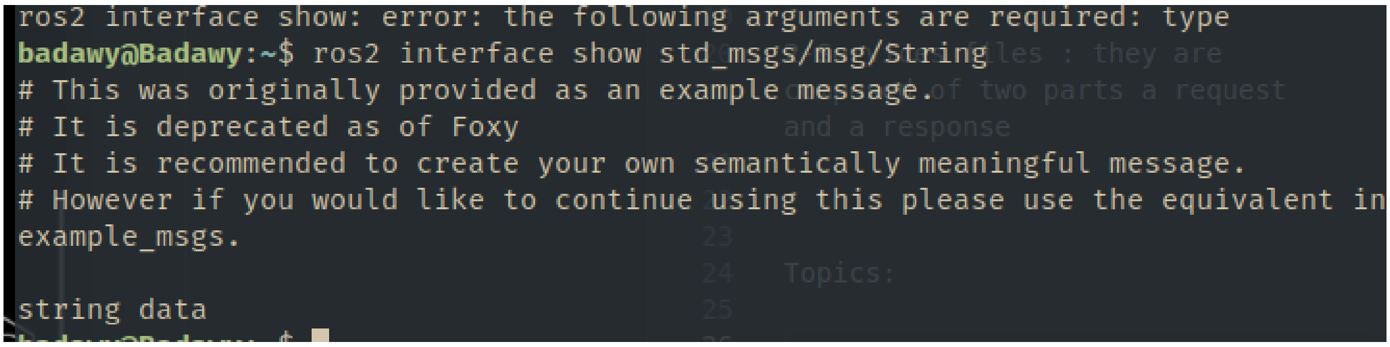
Fourth Step:Build the Workspace

cd ~/ros2_ws
colcon build

Fifth Step: Source the Local setup

source install/setup.bash

We must differ between the nodes name inside ros (first node)
file_name: my_node
executable node inside the setup.py file which will be used to run in terminal



```
ros2 interface show: error: the following arguments are required: type
badawy@Badawy:~$ ros2 interface show std_msgs/msg/String
# This was originally provided as an example message.
# It is deprecated as of Foxy
# It is recommended to create your own semantically meaningful message.
# However if you would like to continue using this please use the equivalent in
example_msgs.

string data
```

The intrface is string type and its name is data

to print a message every second u use a ros2 timer and a callback