



# ROS 2 LEARNING

## INTRODUCTION

ROS 2 (Robot Operating System 2) is an open source software development kit for robotics applications. The purpose of ROS 2 is to offer a standard software platform to developers across industries that will carry them from research and prototyping through to deployment and production. ROS 2 builds on the success of ROS 1, which is used today in countless robotics applications around the world. Some of the key features of ROS 2 include better real-time performance, improved security, support for multiple operating systems, and enhanced modularity and flexibility. ROS 2 is intended to be a more scalable and robust platform that can meet the needs of a wide range of robotic applications, from small mobile robots to large industrial automation systems.

## INSTALLATION

Distro	Release date	Logo	EOL date
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		December 9th, 2022
Foxy Fitzroy	June 5th, 2020		May 2023

\*the distro's which are highlighted in green are stable versions

I am using Humble Hawksbill distro which is the latest stable version, to perform installation of Humble Hawksbill in your system follow the steps in the following link.  
<http://docs.ros.org/en/humble/Installation.html>

There is also another version called Rolling Ridley which is a rolling development release of ROS 2 .It is used for ROS 2 development and by maintainers who want their packages released and ready for the next stable distribution.

# BEGINNING

## 1) configuring the environment

Sourcing the setup files:

**Cmd 1:** `$ source /opt/ros/humble/setup.bash`

**Cmd 2:** `$ shumble`

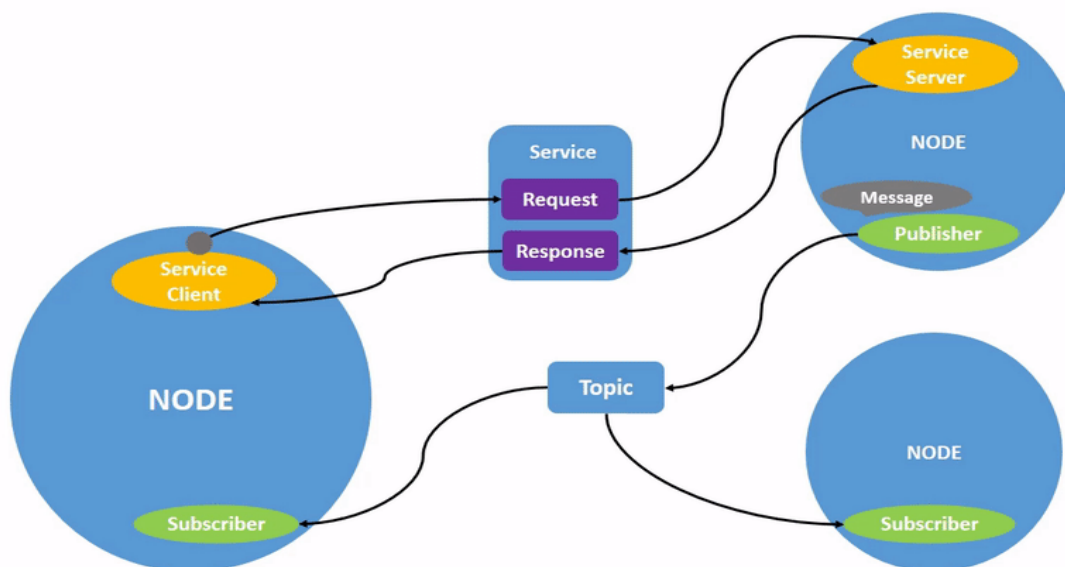
Instead of typing long cmd1 every time to source the ros2 you can type cmd2 to achieve the same by typing the below cmd in terminal.

`$ echo "alias shumble='source /opt/ros/humble/setup.bash'" >> ~/.bashrc`

## 2) Understanding Nodes

**Nodes in ROS2:**

Each node in ROS should be responsible for a single, modular purpose, e.g. controlling the wheel motors or publishing the sensor data from a laser range-finder. Each node can send and receive data from other nodes via topics, services, actions, or parameters.



A full robotic system consists of many nodes working in concert. In ROS 2, a single executable can contain one or more nodes.

**Installing turtlesim packages:**

`$ sudo apt update`

`$ sudo apt install ros-humble-turtlesim`

Listing executables in the package:

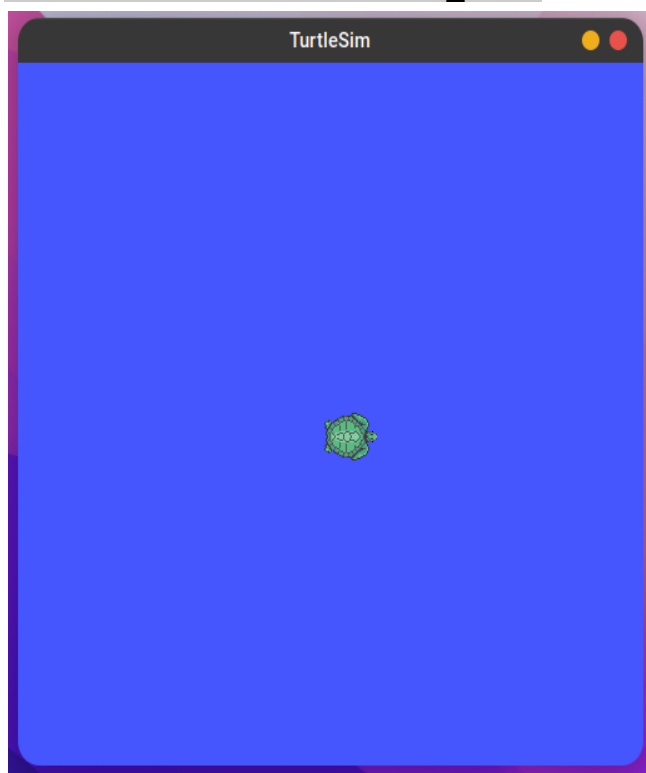
```
$ ros2 pkg executables turtlesim
```

```
bot@bot-HP:~$ ros2 pkg executables turtlesim
turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node
```

Run Nodes:

```
$ ros2 run <package_name> <executable_name>
```

```
$ ros2 run turtlesim turtlesim_node
```



```
$ ros2 run turtlesim turtle_teleop_key
```

```
bot@bot-HP: ~
bot@bot-HP:~$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
-----
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.
'Q' to quit.
█
```

**Nodes list:**

**\$ ros2 node list**

```
bot@bot-HP:~$ ros2 node list
/teleop_turtle
/turtlesim
```

**Node info:**

**\$ ros2 node info /turtlesim**

```
bot@bot-HP:~$ ros2 node info /turtlesim
/turtlesim
Subscribers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /turtle1/cmd_vel: geometry_msgs/msg/Twist
Publishers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /rosout: rcl_interfaces/msg/Log
  /turtle1/color_sensor: turtlesim/msg/Color
  /turtle1/pose: turtlesim/msg/Pose
Service Servers:
  /clear: std_srvs/srv/Empty
  /kill: turtlesim/srv/Kill
  /reset: std_srvs/srv/Empty
  /spawn: turtlesim/srv/Spawn
  /turtle1/set_pen: turtlesim/srv/SetPen
  /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
  /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
  /turtlesim/describe_parameters: rcl_interfaces/srv/DescribeParameters
  /turtlesim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
  /turtlesim/get_parameters: rcl_interfaces/srv/GetParameters
  /turtlesim/list_parameters: rcl_interfaces/srv/ListParameters
  /turtlesim/set_parameters: rcl_interfaces/srv/SetParameters
  /turtlesim/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
Service Clients:

Action Servers:
  /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
Action Clients:
```

***\*ignore if it is overwhelming we will learn it in upcoming chapters***

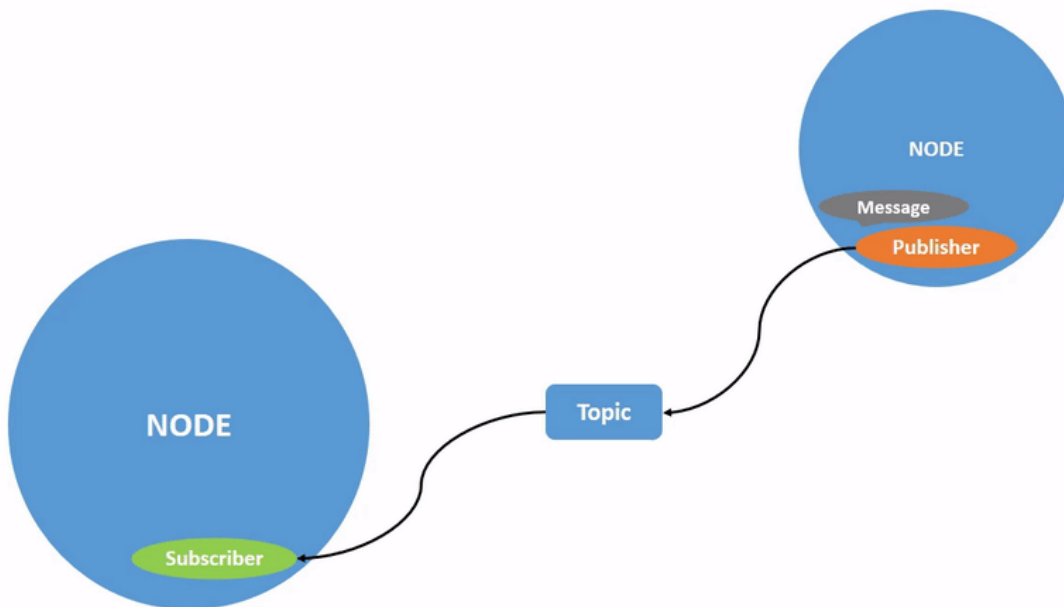
### 3) Understanding Topics

**Topics in ros2:**

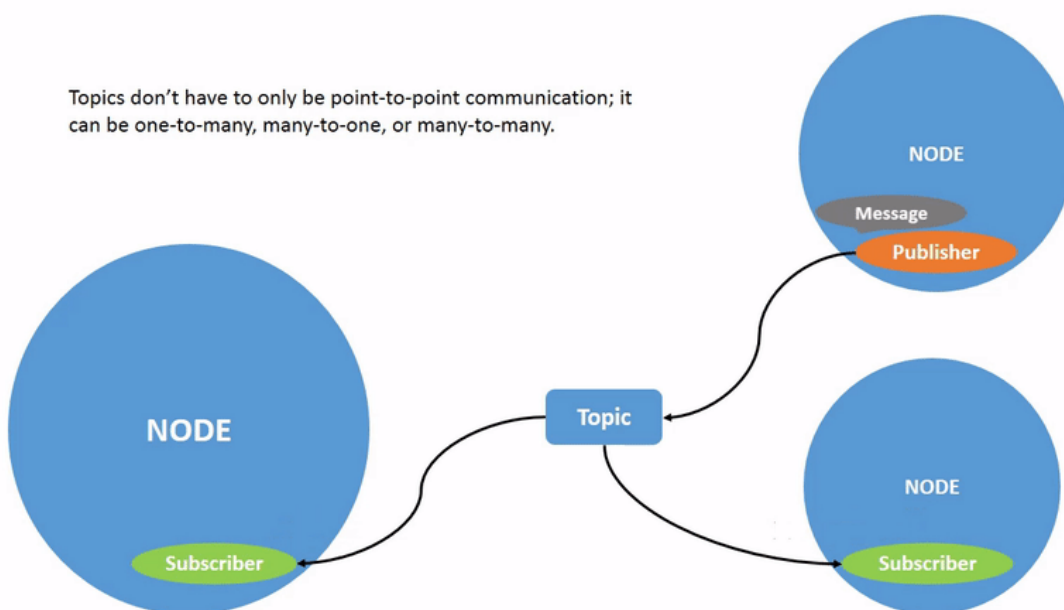
ROS 2 breaks complex systems down into many modular nodes. Topics are a vital element of the ROS graph that act as a bus for nodes to exchange messages.

A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.

Topics are one of the main ways in which data is moved between nodes and therefore between different parts of the system.



Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.



**\$ ros2 topic list**

```
bot@bot-HP:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

```
$ ros2 topic echo /turtle1/pose
```

```
bot@bot-HP:~$ ros2 topic echo /turtle1/pose
x: 4.3751444816589355
y: 8.2113676071167
theta: 1.9839999675750732
linear_velocity: 0.0
angular_velocity: 0.0
---
```

```
$ ros2 topic info
```

```
^Cbot@bot-HP:~$ ros2 topic info /turtle1/pose
Type: turtlesim/msg/Pose
Publisher count: 1
Subscription count: 0
```

```
$ ros2 topic hz /turtle1/pose
```

```
bot@bot-HP:~$ ros2 topic hz /turtle1/pose
WARNING: topic [/turtle1/pose] does not appear to be published yet
average rate: 62.449
  min: 0.015s max: 0.017s std dev: 0.00049s window: 64
average rate: 62.463
  min: 0.015s max: 0.017s std dev: 0.00054s window: 127
average rate: 62.492
  min: 0.015s max: 0.017s std dev: 0.00050s window: 190
```


```
$ ros2 topic pub <topic_name> <msg_type> '<args>'
```

### YAML syntax

```
$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

```
bot@bot-HP:~$ shumble
bot@bot-HP:~$ ros2 node list
/teleop_turtle
/turtlesim
bot@bot-HP:~$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

publisher: beginning loop  
publishing #1: geometry\_msgs.msg.Twist(linear=geometry\_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry\_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8))



## 4) Recording and playing data using bag files

### 1) Record bag files

Let's also make a new directory to store our saved recordings, just as good practice:

```
$ mkdir bag_files
```

```
$ cd bag_files
```

Recording bag file using below cmd:

```
$ ros2 bag record <topic_name>
```

```
$ ros2 bag record /turtle1/cmd_vel
```

```
bot@bot-HP:~/bag_files$ ros2 bag record /turtle1/cmd_vel
[INFO] [1680526611.248407859] [rosbag2_recorder]: Press SPACE for pausing/resuming
[INFO] [1680526611.250748878] [rosbag2_storage]: Opened database 'rosbag2_2023_04_03-18_26_51/rosbag2_2023_04_03-18_26_51_0.db3' for READ_WRITE.
[INFO] [1680526611.250969057] [rosbag2_recorder]: Event publisher thread: Starting
[INFO] [1680526611.252377664] [rosbag2_recorder]: Listening for topics...
```

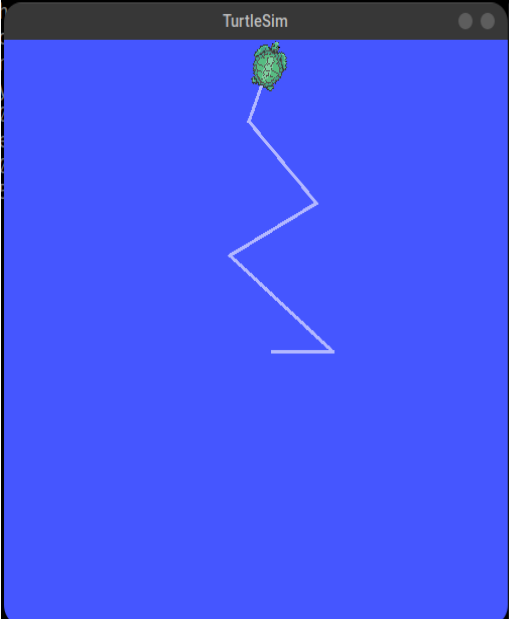
### 2) Bag files info

```
$ ros2 bag info <bag_file_name>
```

```
bot@bot-HP:~/bag_files$ ros2 bag info rosbag2_2023_04_03-18_26_51
Files:          rosbag2_2023_04_03-18_26_51_0.db3
Bag size:       33.0 KiB
Storage id:     sqlite3
Duration:       23.901s
Start:          Apr 3 2023 18:26:53.370 (1680526613.370)
End:            Apr 3 2023 18:27:17.272 (1680526637.272)
Messages:       72
Topic information: Topic: /turtle1/cmd_vel | Type: geometry_msgs/msg/Twist | Count: 72 | Serialization Format: cdr
```

### 3) Bag file data playing

```
$ ros2 bag play subset
```



```
bot@bot-HP:~/bag_files$ ros2 bag play rosbag2_2023_04_03-18_26_51/
[INFO] [1680533173.773310212] [rosbag2_storage]: Opened data base 'rosbag2_2023_04_03-18_26_51/rosbag2_2023_04_03-18_26_51_0.db3' for READ_ONLY.
[INFO] [1680533173.773390925] [rosbag2_player]: Set rate to 1
[INFO] [1680533173.781623934] [rosbag2_player]: Adding keyboard callbacks.
[INFO] [1680533173.781692985] [rosbag2_player]: Press SPACE for Pause/Resume
[INFO] [1680533173.781714115] [rosbag2_player]: Press CURSOR_RIGHT for Play Next Message
[INFO] [1680533173.781731939] [rosbag2_player]: Press CURSOR_UP for Increase Rate 10%
[INFO] [1680533173.781748189] [rosbag2_player]: Press CURSOR_DOWN for Decrease Rate 10%
[INFO] [1680533173.782384926] [rosbag2_storage]: Opened data base 'rosbag2_2023_04_03-18_26_51/rosbag2_2023_04_03-18_26_51_0.db3' for READ_ONLY.
```

### 3) Creating a bag file to record multiple topics

```
$ ros2 bag record -o subset <topic 1> <topic 2> ... <topic n>
```

```
$ ros2 bag record -o subset /turtle1/cmd_vel /turtle1/pose
```

- The `-o` option allows you to choose a unique name for your bag file.
- There is another option you can add to the command, `-a`, which records all the topics on your system.

### 5) Creating Workspace

Create new directory:

```
$ mkdir -p ~/ros2_ws/src # -p, --parents ,make parent directories as needed
```

Best practice is to create a new directory for every new workspace. Inside src folder of ros2\_ws we can create our packages

A single workspace can contain as many packages as you want, each in their own folder. You can also have packages of different build types in one workspace (CMake, Python, etc.). You cannot have nested packages.

Best practice is to have a src folder within your workspace, and to create your packages in there. This keeps the top level of the workspace “clean”.

A trivial workspace might look like:

```
workspace_folder/  
  src/  
    package_1/  
      CMakeLists.txt  
      package.xml  
  
    package_2/  
      setup.py  
      package.xml  
      resource/package_2  
  
    ...  
    package_n/  
      CMakeLists.txt  
      package.xml
```



## 6) Creating new packages:

A package can be considered a container for your ROS 2 code. If you want to be able to install your code or share it with others, then you'll need it organised in a package. With packages, you can release your ROS 2 work and allow others to build and use it easily.

Package creation in ROS 2 uses ament as its build system and colcon as its build tool. You can create a package using either CMake or Python, which are officially supported, though other build types do exist.

### Inside packages:

package.xml -> file containing meta information about the package

CMakeLists.txt -> file that describes how to build the code within the package

The simplest possible package may have a file structure that looks like:

```
my_package/  
  CMakeLists.txt  
  package.xml
```

### Creating packages:

Make sure you are in the src folder before running the package creation command.

#### C++ Package:

```
$ ros2 pkg create --build-type ament_cmake <package_name>
```

#### Python Package:

```
$ ros2 pkg create --build-type ament_python <package_name>
```

You can also specify the below details here or can edit in package.xml later.

```
bot@bot-HP:~$ ros2 pkg create --build-type ament_cmake --  
--build-type          --library-name          --node-name  
--dependencies        --license              --package-format  
--description         --maintainer-email     --package_format  
--destination-directory --maintainer-name
```

## 7) Building packages using colcon

colcon does out of source builds. By default it will create the following directories as peers of the src directory:

- The **build** directory will be where intermediate files are stored. For each package a subfolder will be created in which e.g. CMake is being invoked.
- The **install** directory is where each package will be installed to. By default each package will be installed into a separate subdirectory.
- The **log** directory contains various logging information about each colcon invocation.

Using following cmd in workspace directory for building the packages in src folder

```
$ colcon build
```

**--symlink-install:**

```
$ colcon build --symlink-install
```

**--symlink-install** allows the installed files to be changed by changing the files in the source space (e.g. Python files or other not compiled resources) for faster iteration. (enables changes in python executables will take effect without using colcon build every time when we make changes)

In simple terms, it just eliminates the need for rebuilding packages whenever they make changes in scripts.

**Other colcon commands:**

```
$ colcon list # List all packages in the workspace
```

```
$ colcon graph # List all packages in the workspace in topological order
```

```
$ colcon build --packages-select <name-of-pkg>
```

```
$ colcon build --packages-up-to <name-of-pkg>
```

```
$ colcon test # Test all packages in the workspace
```

```
$ colcon test-result --all # Enumerate all test results
```

**Sourcing the workspace:**

```
$ source ros2_ws/install/setup.sh #you can also make an alias in ~/.bashrc
```

**ros2(underlay) workspace(overlay)**

- Before sourcing the overlay, it is very important that you open a new terminal, separate from the one where you built the workspace.
- Sourcing an overlay in the same terminal where you built, or likewise building where an overlay is sourced, may create complex issues.

## 8) Writing publisher and subscriber in python

A publish-subscribe (pub-sub) pattern is used to facilitate communication between nodes. In this pattern, nodes communicate by publishing messages to a topic, and other nodes that are interested in receiving those messages subscribe to that topic. The publisher node sends the message to the topic, and all subscriber nodes that have registered for that topic receive the message.

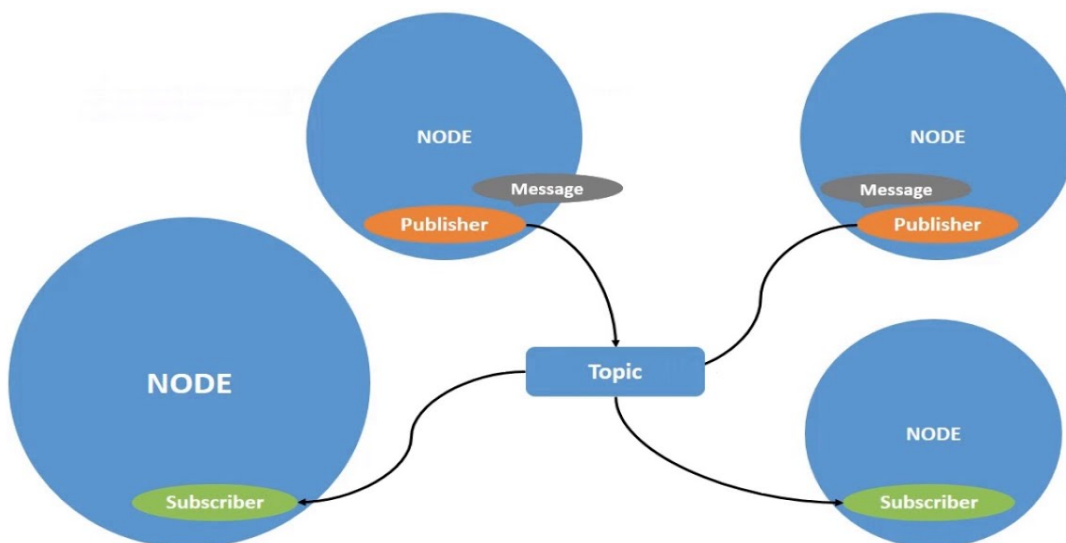
Publish-subscribe communication is widely used in ROS for various purposes, such as:

**Sensor data:** Sensors such as cameras, lidars, and ultrasonic sensors publish data to a topic, and other nodes that need that data subscribe to the topic. For example, a robot may use a camera to capture images and publish them to a topic, and a node that performs image processing may subscribe to the same topic to receive those images.

**Control commands:** Nodes that are responsible for controlling the robot can publish commands to a topic, and other nodes that need to execute those commands can subscribe to that topic. For example, a node responsible for controlling the robot's arm may publish commands to a topic, and a node responsible for controlling the robot's gripper may subscribe to that topic to receive those commands.

**Status updates:** Nodes can publish status updates to a topic, and other nodes can subscribe to that topic to receive those updates. For example, a robot may publish status updates about its battery level, and a node that is responsible for monitoring the robot's health may subscribe to that topic to receive those updates.

Overall, the pub-sub pattern allows for flexible and scalable communication between nodes in ROS.



(github)

```
bot@bot-HP:~$ shumble
bot@bot-HP:~$ rosrun turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_
QPA_PLATFORM=wayland to run on Wayland anyway.
[INFO] [1680191996.764956313]: Starting turtlesim
with node name /turtlesim
[INFO] [1680191996.764956313]: TurtleSim
[INFO] [1680192078.810152616]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.825414297]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.841770307]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.858116253]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.873437682]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.889817702]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.906170264]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.923483476]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.937743837]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.954088780]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.969467450]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192078.986080956]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192079.003042711]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192079.018564694]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
[INFO] [1680192079.034522132]: [Pose_Subscriber]: (9.576444625
854492,5.544444561004639)
```

(github)

```
bot@bot-HP:~$ shumble
bot@bot-HP:~$ ros2 run turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_
QPA_PLATFORM=wayland to run on Wayland anyway.
[INFO] [1680191996.764956313] [turtlesim]: Starting turtlesi
m with node name /turtlesim
[INFO] [1680191996.764956313] [turtlesim]: TurtleSim/RobotI
turtle1] at x=[0.0, 0.0] y=[0.0, 0.0] theta=[0.0]
[INFO] [1680192078.810152616] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.825414297] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.841770307] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.858116253] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.873437682] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.889817702] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.906170264] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.923483476] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.937743837] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.954088780] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.969467450] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192078.986080956] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192079.003042711] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192079.018564694] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
[INFO] [1680192079.034522132] [Pose_Subscriber]: (9.576444625
854492, 5.544444561004639)
```

### 3) Creating a simple publisher to publish a topic

([github](#))

After creating node add, To allow the ros2 run command to run your node, you must add the entry point to setup.py (src/pub\_sub\_py/setup.py)

```
bot@bot-HP:~$ shumble
bot@bot-HP:~$ rosws
bot@bot-HP:~$ ros2 run pub_sub_py hello
[INFO] [1680191031.751613692] [First_Node]: Hello 0
[INFO] [1680191032.234654925] [First_Node]: Hello 1
[INFO] [1680191032.734072766] [First_Node]: Hello 2
[INFO] [1680191033.234069001] [First_Node]: Hello 3
[INFO] [1680191033.734202049] [First_Node]: Hello 4
[INFO] [1680191034.234589246] [First_Node]: Hello 5
[INFO] [1680191034.734106914] [First_Node]: Hello 6
[INFO] [1680191035.234570892] [First_Node]: Hello 7
```

### 4) Creating multiple publisher and subscribers

([github](#))

After creating node add, To allow the ros2 run command to run your node, you must add the entry point to setup.py (src/pub\_sub\_py/setup.py)

The screenshot displays a terminal window with three separate ROS2 nodes running. The first terminal window shows the execution of 'shumble' and 'ros2exam', followed by 'ros2 run exam\_nodes shop'. The second terminal window shows 'shumble', 'ros2exam', and 'ros2 run exam\_nodes farmer1'. The third terminal window shows 'shumble', 'ros2exam', and 'ros2 run exam\_nodes farmer2'. The RQT graph window shows the network of nodes: /farmer\_1, /farmer\_2, /item/farmer1, /item/farmer2, and /shop. The graph indicates that /farmer\_1 and /farmer\_2 are publishers, and /item/farmer1 and /item/farmer2 are subscribers. The /shop node is also shown as a subscriber.

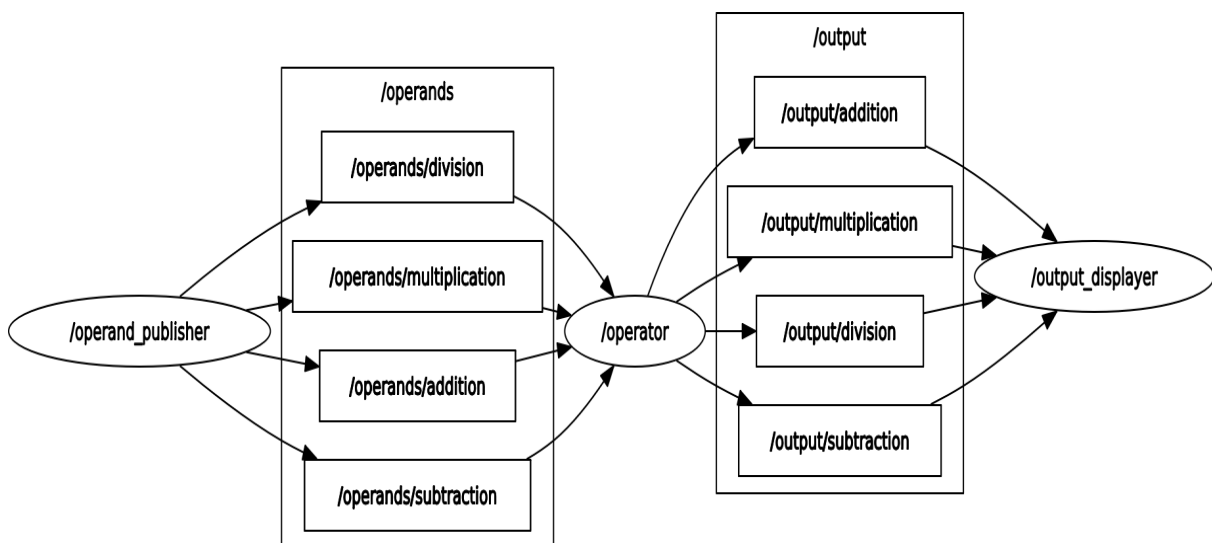
## 5) Creating a multiple nodes to perform basic arithmetic operations using python ([github](#))

After creating node add, To allow the ros2 run command to run your node, you must add the entry point to setup.py (src/pub\_sub\_py/setup.py) then source all(underlay and overlay appropriately.

```

bot@bot-HP:~$ shumble
bot@bot-HP:~$ ros2 run pub_sub_py operand
[INFO] [1680193729.407985163] [operand_publisher]: The operands are published
[INFO] [1680193730.388001937] [operand_publisher]: The operands are published
[INFO] [1680193731.388617346] [operand_publisher]: The operands are published
[INFO] [1680193732.387981640] [operand_publisher]: The operands are published
[INFO] [1680193733.387982958] [operand_publisher]: The operands are published
[INFO] [1680193734.387989609] [operand_publisher]: The operands are published
[INFO] [1680193735.388004451] [operand_publisher]: The operands are published
bot@bot-HP:~$ ros2 run pub_sub_py operator
[INFO] [1680193730.407452334] [operator]: The result of addition is published
[INFO] [1680193730.408430154] [operator]: The result of subtraction is published
[INFO] [1680193730.409737210] [operator]: The result of multiplication is published
[INFO] [1680193730.410652552] [operator]: The result of division is published
[INFO] [1680193731.388494706] [operator]: The result of addition is published
[INFO] [1680193731.389390992] [operator]: The result of subtraction is published
[INFO] [1680193731.390671470] [operator]: The result of multiplication is published
[INFO] [1680193731.391652952] [operator]: The result of division is published
bot@bot-HP:~$ shumble
bot@bot-HP:~$ ros2 run pub_sub_py output
[INFO] [1680193733.412132373] [output_displayer]: The output of addition is: 8
[INFO] [1680193733.413086438] [output_displayer]: The output of subtraction is: 0
[INFO] [1680193733.413947725] [output_displayer]: The output of multiplication is: 16
[INFO] [1680193733.414712337] [output_displayer]: The output of division is: 1
[INFO] [1680193734.390970864] [output_displayer]: The output of addition is: 10
[INFO] [1680193734.392400212] [output_displayer]: The output of subtraction is: 0
[INFO] [1680193734.393152269] [output_displayer]: The output of multiplication is: 25
[INFO] [1680193734.394470238] [output_displayer]: The output of division is: 1
[INFO] [1680193735.391104238] [output_displayer]: The output of addition is: 12
[INFO] [1680193735.391898754] [output_displayer]: The output of subtraction is: 0
[INFO] [1680193735.394112926] [output_displayer]: The output of multiplication is: 36
[INFO] [1680193735.395913306] [output_displayer]: The output of division is: 1
[INFO] [1680193736.391262177] [output_displayer]: The output of addition is: 14
[INFO] [1680193736.392556775] [output_displayer]: The output of subtraction is: 0

```



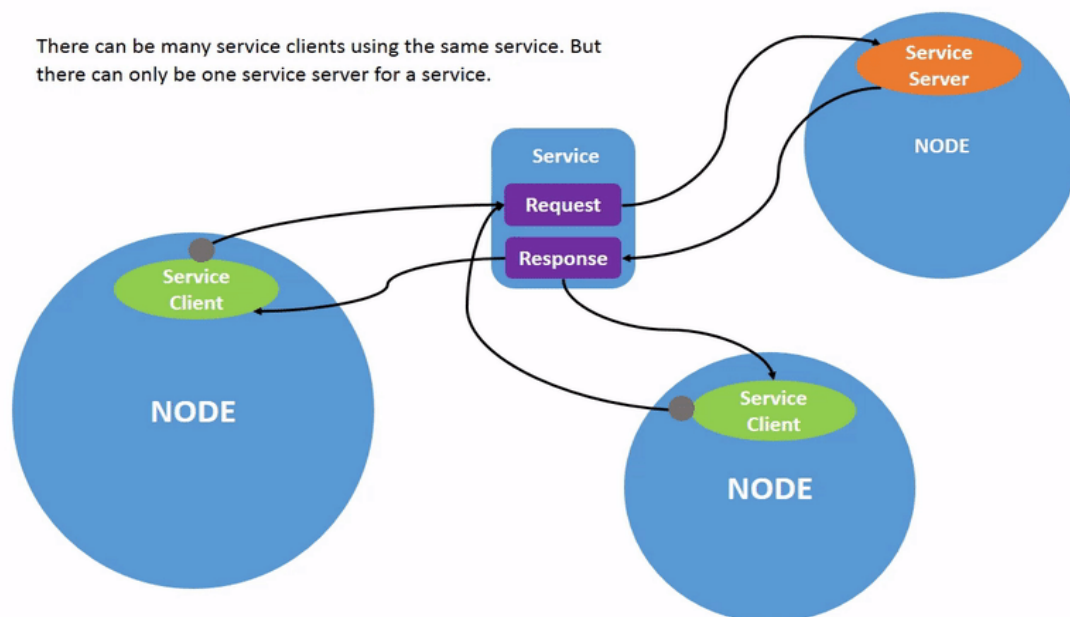


## 9) Writing simple service and client using python

A service client is a node in ROS that requests services from a service server. In the client-server model of ROS, a service client sends a request to a service server and waits for a response. The service client and server communicate using a request-response protocol, where the client sends a request message and the server sends back a response message.

The service client is responsible for creating a request message and sending it to the server. It then waits for the server to process the request and send back a response. Once the response is received, the client can then use the data in the response to carry out any necessary actions.

Service clients are commonly used in ROS for various tasks, such as querying the state of a robot, requesting a map from a mapping service, or requesting information from a sensor. In general, a service client is useful for any situation where a node needs to make a request for information or an action to be performed, and then wait for a response from another node.



[OBJ:OBJ]

### Creating a package for interfaces:

Creating custom `.msg` and `.srv` files in their own package, and then utilising them in a separate package. Both packages should be in the same workspace.

```
$ ros2 pkg create --build-type ament_cmake custom_interfaces
```

Inside custom\_interfaces package folder

```
$ mkdir msg
```

```
$ mkdir srv
```

Inside msg folder create a file named Num.msg containing following contents:

```
int64 num
```

Inside srv folder create a file named AddTwoInts.srv containing following contents:

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

### Edit Cmakelist.txt:

Add the following contents in it.

```
find_package(rosidl_default_generators REQUIRED)
find_package(builtin_interfaces REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Num.msg"
  "srv/AddTwoInts.srv"
)
```

### Edit package.xml:

Add the following contents in it.

```
<depend>geometry_msgs</depend>
  <build_depend>rosidl_default_generators</build_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>
```

### Build:

```
$ colcon build --packages-select tutorial_interfaces
```

### Confirm creation:

Source the workspace. Now you can confirm that your interface creation worked by using the ros2 interface show command:

```
$ ros2 interface show custom_interfaces/srv/AddTwoInts
```

```
bot@bot-HP:~$ ros2 interface show custom_interfaces/srv/AddTwoInts
int64 a
int64 b
---
int64 sum
```



**\$ ros2 interface show custom\_interfaces/msg/Num**

```
bot@bot-HP:~$ ros2 interface show custom_interfaces/msg/Num
int64 num
```

**Write the service and client nodes:**

[\(github\)](#)

**Add entry point:**

To allow the ros2 run command to run your node, you must add the entry point to **setup.p** inside the package

```
bot@bot-HP:~$ ros2 run service_client_py addtwoints_client 2 4
[INFO] [1680546376.881778346] [add_two_client_async]: The result of
addition is 6
bot@bot-HP:~$ ros2 run service_client_py addtwoints_client 55 45
[INFO] [1680546380.286958611] [add_two_client_async]: The result of
addition is 100
bot@bot-HP:~$ ros2 run service_client_py addtwoints_client 25 75
[INFO] [1680546385.355745855] [add_two_client_async]: The result of
addition is 100
bot@bot-HP:~$ ros2 run service_client_py addtwoints_client 55 45
[INFO] [1680546396.502706039] [add_two_client_async]: The result of
addition is 100
bot@bot-HP:~$

bot@bot-HP:~$ ros2 run service_client_py addtwoints_s
ervice
[INFO] [1680546376.857862645] [add_two_ints_service]:
Incoming request
a:2 b:4
[INFO] [1680546380.269591789] [add_two_ints_service]:
Incoming request
a:55 b:45
[INFO] [1680546385.324306527] [add_two_ints_service]:
Incoming request
a:25 b:75
[INFO] [1680546396.485059765] [add_two_ints_service]:
Incoming request
a:55 b:45
```

## Rqt tool

rqt is a graphical user interface (GUI) framework for ROS that allows developers to create custom plugins to visualise and interact with ROS data. It provides a set of tools for debugging and analysing robot systems, such as visualising sensor data, controlling robot movement, and visualising robot models. rqt is extensible and allows developers to easily create new plugins or modify existing ones to fit their specific needs. This makes it a powerful tool for robotics development and testing.

**Installing rqt:**

```
$ sudo apt update
```

```
$ sudo apt install ~nros-humble-rqt*
```

To run rqt:

```
$ rqt
```

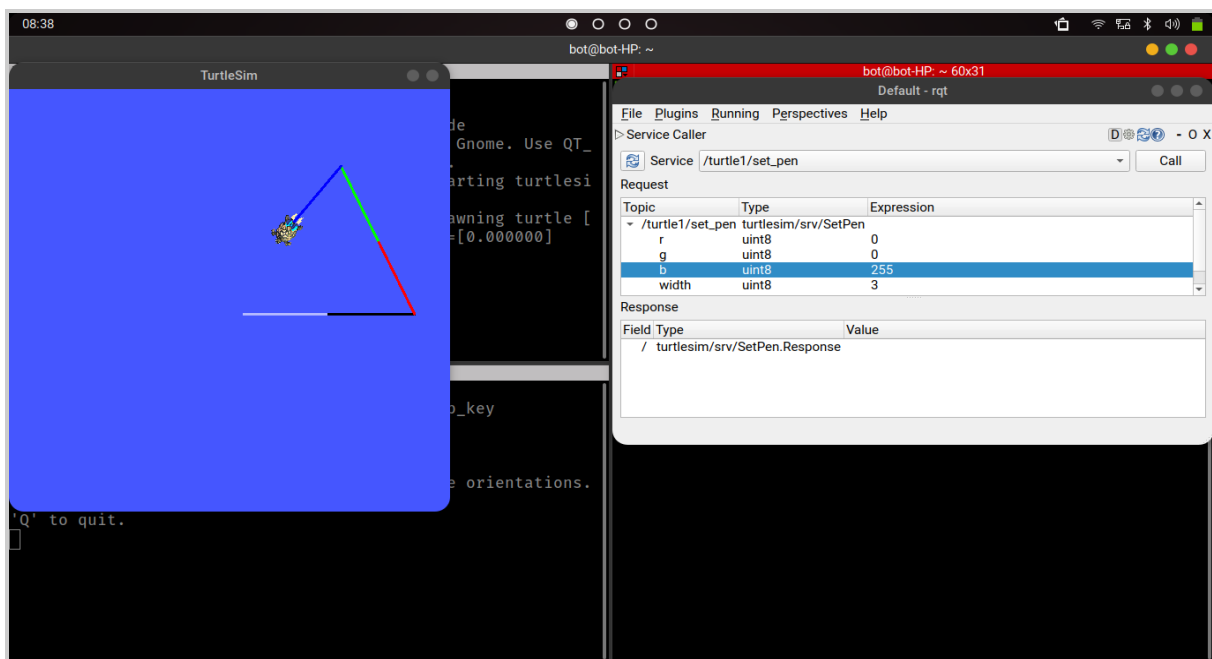
select Plugins > Services > Service Caller

Use the refresh button to the left of the Service dropdown list to ensure all the services of your turtlesim node are available.

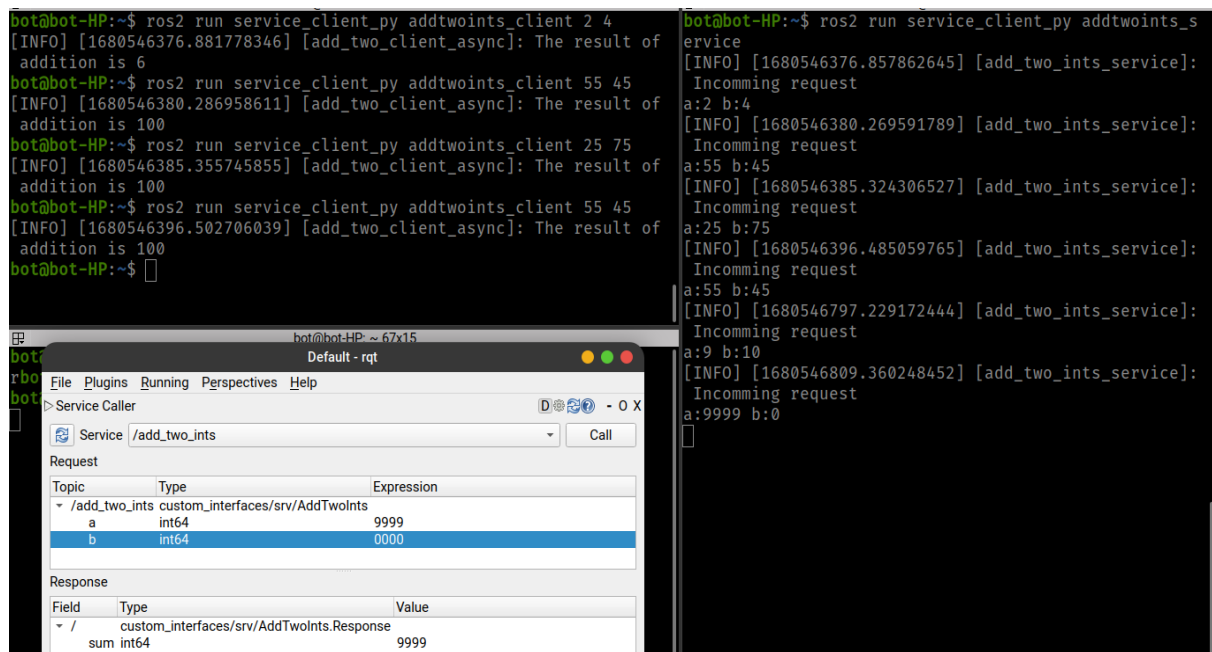
Click on the Service dropdown list to see turtlesim's services, and select any service eg, /turtle1/set/pin

The values for r, g and b, which are between 0 and 255, set the colour of the pen turtle1 draws with, and width sets the thickness of the line.

To have turtle1 draw with a distinct red line, change the value of r to 255, and the value of width to 3. Call the service after updating the values.



Now try it for the current service:



## 10) Writing an action server and client

An action server provides a way for a client to send a goal to a server and receive feedback and result messages. Actions are used in situations where a goal takes a long time to execute, and the client needs to be informed about the progress of the goal and the final result.

The basic workflow of an action server is as follows:

1. The client sends a goal message to the server.
2. The server receives the goal message and begins processing it.
3. The server sends feedback messages to the client periodically to update it on the progress of the goal.
4. The server sends a result message to the client when the goal has been completed.

Some use cases for action servers in robotics include:

**Autonomous navigation:** An action server can be used to provide feedback to a client about the progress of a robot's path planning and navigation, such as the robot's current location and obstacles encountered along the way.

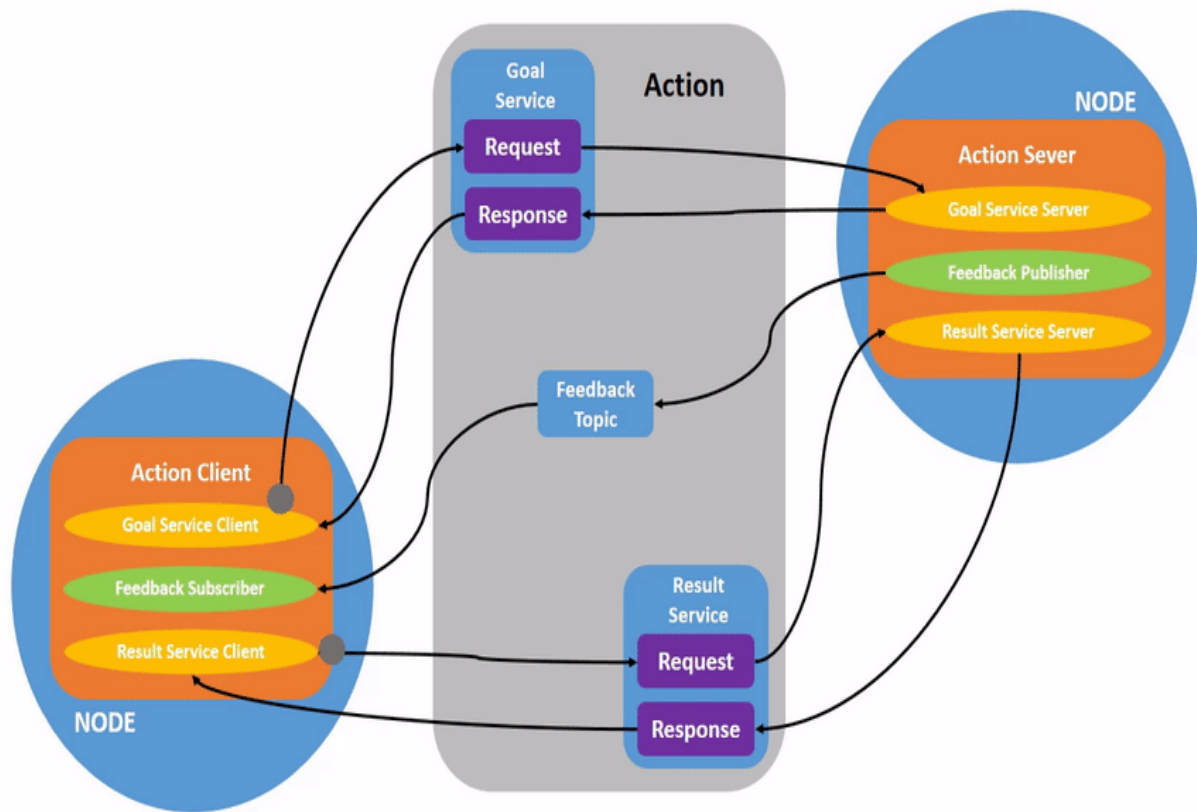
**Manipulation tasks:** An action server can be used to provide feedback to a client about the progress of a robot's manipulation tasks, such as grasping an object or moving it to a desired location.

**Perception tasks:** An action server can be used to provide feedback to a client about the progress of a robot's perception tasks, such as object detection or tracking.

**Calibration:** An action server can be used to provide feedback to a client about the progress of a robot's calibration tasks, such as camera or sensor calibration.

Overall, action servers provide a powerful way to manage long-running tasks in robotics, where the client needs to be updated on the progress and outcome of the task.

In short, Actions are a form of asynchronous communication in ROS 2. Action clients send goal requests to action servers. Action servers send goal feedbacks and result to action clients.



### Adding action file to custom\_interface package:

Inside custom interfaces create a folder named action and create a `.action` file. Then Inside action folder create a file named CustomActionOne.action containing following contents:

```
int32 order
---
int32[] sequence
---
int32[] partial_sequence
```

### Edit Cmakelist.txt (inside custom\_interfaces):

Add the action in as like we added it before

```
rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Num.msg"
  "srv/AddTwoInts.srv"
  "action/CustomActionOne.action"
)
```

### Edit package.xml (inside custom\_interfaces):

Add the following in it for the purpose of importing time in a python file.

```
<build_depend>builtin_interfaces</build_depend>
<exec_depend>builtin_interfaces</exec_depend>
```

### Create a separate package for action server node:

```
$ ros2 pkg create --build-type ament_python action_server --dependencies
custom_interface
```

### Write the action server nodes:

[\(github\)](#)

### Add entrypoint:

Add the entry point in the `setup.py` file inside the `action_server` package.

```
"fibonacci_server = action_server.custom_action_server:main",
"fibonacci_client = action_server.custom_action_client:main"
```

### Run action server:

```
bot@bot-HP:~$ ros2 run action_server fibonacci_server
[INFO] [1680620749.812418511] [fibonacci_action_server]: Executing goal...
[INFO] [1680620749.813203402] [fibonacci_action_server]: Feedback: array('i', [0, 1, 1])
[INFO] [1680620750.815708696] [fibonacci_action_server]: Feedback: array('i', [0, 1, 1, 2])
[INFO] [1680620751.818252251] [fibonacci_action_server]: Feedback: array('i', [0, 1, 1, 2, 3])
[INFO] [1680620752.820855046] [fibonacci_action_server]: Feedback: array('i', [0, 1, 1, 2, 3, 5])
[ ]

bot@bot-HP:~$ ros2 run action_server fibonacci_client
[INFO] [1680620749.824380025] [fibonacci_action_client]: Goal accepted :)
[INFO] [1680620749.825984307] [fibonacci_action_client]: Received feedback: array('i', [0, 1, 1])
[INFO] [1680620750.817499643] [fibonacci_action_client]: Received feedback: array('i', [0, 1, 1, 2])
[INFO] [1680620751.820256714] [fibonacci_action_client]: Received feedback: array('i', [0, 1, 1, 2, 3])
[INFO] [1680620752.822817542] [fibonacci_action_client]: Received feedback: array('i', [0, 1, 1, 2, 3, 5])
[INFO] [1680620753.825884169] [fibonacci_action_client]: Result: array('i', [0, 1, 1, 2, 3, 5])
bot@bot-HP:~$ [ ]
```

## 11) Launch files

The launch system in ROS 2 is responsible for helping the user describe the configuration of their system and then execute it as described. The configuration of the system includes what programs to run, where to run them, what arguments to pass them, and ROS-specific conventions which make it easy to reuse components throughout the system by giving them each a different configuration. It is also responsible for monitoring the state of the processes launched, and reporting and/or reacting to changes in the state of those processes.

Launch files are used to automate the process of starting ROS nodes and configuring their parameters. They allow you to start multiple nodes at once and specify their command-line arguments, node names, and other parameters. Additionally, launch files allow you to specify the order in which nodes are started, and they can be used to group related nodes together into logical units.

### Creating launch files:

Create a folder named launch inside the package (eg. pub\_sub\_py)

```
$ mkdir launch
```

Launch files written in Python, XML, or YAML can start and stop different nodes as well as trigger and act on various events.

Create a file named `.launch.py` (i am using python) eg. farmer\_shop.launch.py ([github](#))

### Edit the package.xml file:

Add the following lines in it.

```
<exec_depend>ros2launch</exec_depend>
```

### Edit setup.py file:

Add these imports

```
import os
from glob import glob
```

Add these in data\_files:

```
(os.path.join('share', package_name, 'launch/'),
 glob('launch/*launch,[pxy][yma]*')),
```

So i will look like this:

```
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']),
    (os.path.join('share', package_name, 'launch/'),
     glob('launch/*launch,[pxy][yma]*')),
],

```

**Launching:**

**Format:**

**`$ ros2 launch <package_name> <launch_file_name>`**

**`$ ros2 launch pub_sub_py farmer_shop.launch.py`**

```

bot@bot-HP:~$ ros2 launch pub_sub_py farmer_shop.launch.py
[INFO] [launch]: All log files can be found below /home/bot/.ros/log/2023-04-04-21-36-29-536043-bot-HP-21821
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [farmer1-1]: process started with pid [21823]
[INFO] [farmer2-2]: process started with pid [21825]
[INFO] [shop-3]: process started with pid [21827]
[shop-3] [INFO] [1680624391.159532743] [shop]: apple is received
[farmer1-1] [INFO] [1680624391.159546495] [farmer_1]: farmer1 is transporting apple
[shop-3] [INFO] [1680624391.163240122] [shop]: orange is received
[farmer2-2] [INFO] [1680624391.179621694] [farmer_2]: farmer 2 is transporting orange
[farmer1-1] [INFO] [1680624392.141493318] [farmer_1]: farmer1 is transporting apple
[shop-3] [INFO] [1680624392.141990277] [shop]: apple is received
[farmer2-2] [INFO] [1680624392.163737240] [farmer_2]: farmer 2 is transporting orange
[shop-3] [INFO] [1680624392.164225996] [shop]: orange is received
[farmer1-1] [INFO] [1680624393.142000763] [farmer_1]: farmer1 is transporting apple
[shop-3] [INFO] [1680624393.142438025] [shop]: apple is received
[farmer2-2] [INFO] [1680624393.163058509] [farmer_2]: farmer 2 is transporting orange
[shop-3] [INFO] [1680624393.163570673] [shop]: orange is received
[farmer1-1] [INFO] [1680624394.141476953] [farmer_1]: farmer1 is transporting apple
[shop-3] [INFO] [1680624394.142178095] [shop]: apple is received
[farmer2-2] [INFO] [1680624394.162974369] [farmer_2]: farmer 2 is transporting orange
[shop-3] [INFO] [1680624394.163478050] [shop]: orange is received
[farmer1-1] [INFO] [1680624395.142190605] [farmer_1]: farmer1 is transporting apple
[shop-3] [INFO] [1680624395.142690300] [shop]: apple is received
[farmer2-2] [INFO] [1680624395.163046691] [farmer_2]: farmer 2 is transporting orange
[shop-3] [INFO] [1680624395.163748435] [shop]: orange is received
[farmer1-1] [INFO] [1680624396.141480216] [farmer_1]: farmer1 is transporting apple
[shop-3] [INFO] [1680624396.142166727] [shop]: apple is received
[farmer2-2] [INFO] [1680624396.163514517] [farmer_2]: farmer 2 is transporting orange
[shop-3] [INFO] [1680624396.164051975] [shop]: orange is received
[farmer1-1] [INFO] [1680624397.142250956] [farmer_1]: farmer1 is transporting apple

```

## Resources

ROS 2 Official Documentation: <http://docs.ros.org/en/humble/Tutorials.html>

Hummingbird Youtube Channel: <https://www.youtube.com/@hummingbird19>

Robotics Back-End Youtube Playlist:

<https://www.youtube.com/playlist?list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy>

**Finally a very important information:**

"ROS 2" not "ROS2" - apparently, the space between the letters is a big deal.



**ROS2**



**ROS 2**

Document created by Manoj M

Linkedin - <https://www.linkedin.com/in/manoj-murali-/>