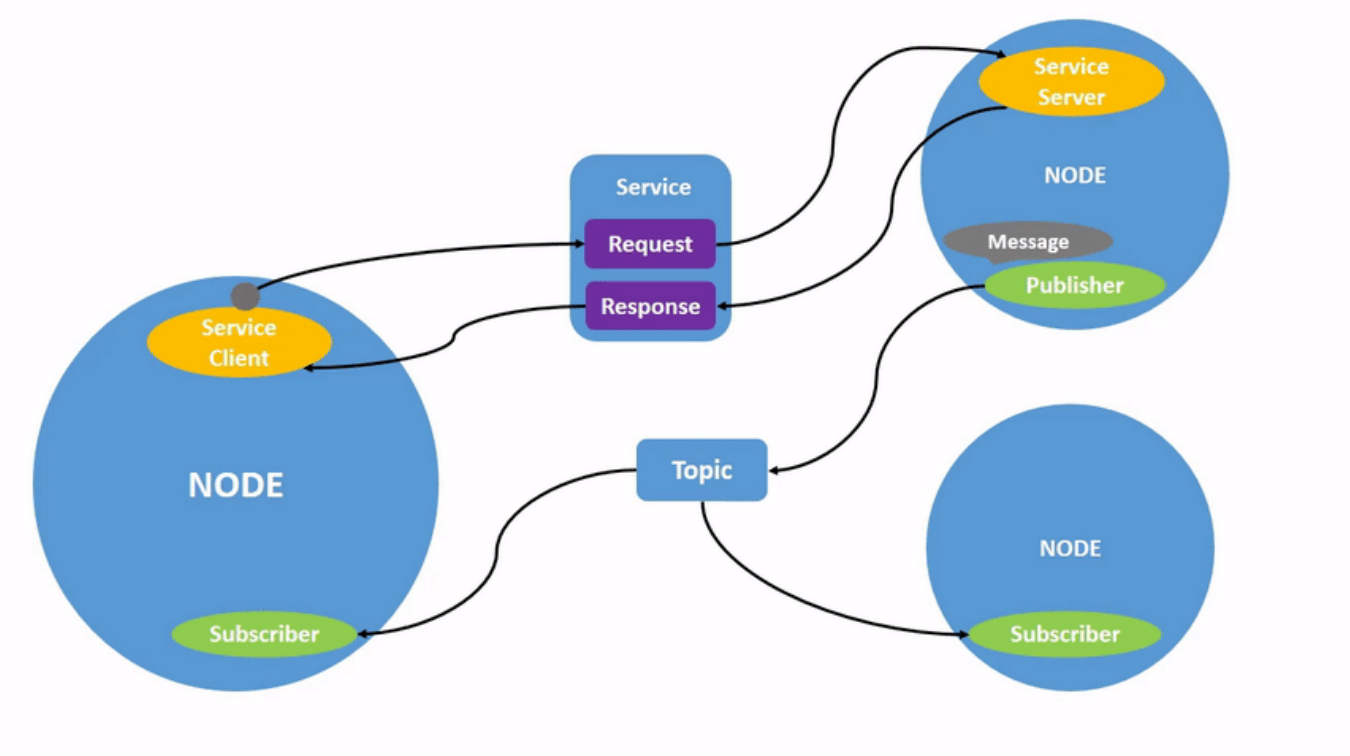
# 1. Theoretical background

## Official documentation

- ROS graph
  - Network of ROS 2 elements (nodes) processing data together at one time.
  - It encompasses all executables and the connections between them
  - Use rqt_graph command to visualize it
- Nodes:
  - Basic functional element of a ROS network responsible for a single module purpose (e.g. one node for controlling wheel motors, one node for controlling a laser range-finder, etc).
  - Each node can send and receive data to other nodes via:
    - Topics, services, actions, or parameters.
  - They can run on different computers.
  - A single executable file can contain multiple nodes.




# 2. Python nodes

- API

## 2.1 Structure

1. Import the modules
2. Create a class that inherits the imported Node class and inside:
   - Define the name of the node in the constructor using the superclass constructor
3. Create a main function :
   - Initialize the ROS client for python
   - Create an instance of your node class
   - Destroy the instance of the node (optonally)
   - Shutdown the ROS client
4. Call the main function inside the if name == 'main' statement

## 2.2 Methods

- rclpy API

- Modules and clases to import

```
import rclpy
from rclpy.node import Node
```

- Definition of the node name

```
super().__init__('my_node')
```

- Display messages in the CLI

```
Node.get_logger().info('<message>')
```

- Initialize the ROS client for python

```
    rclpy.init(args = args)
```

- Destroy the instance of the node (optional)

```
    node.destroy_node()
```

- Shutdown the ROS client

```
    rclpy.shutdown()
```

## 2.3 Configuration

1. Configure the manifesto:
    - **Update the dependencies**
    - Version, description, mail, license

    - Dependencies: If not defined in the initial package creation process
2. Configure the setup.py package file
    - Version, description, mail, license
    - Add an entry point
        - When compiled, the script containing a node is going to be set in an executable file (an entry point)
        - This allows the use of the “ros2 run” comand for the node
        - The entry point is made of: *executable = package.script:main*
3. Build the package:
    - Use the “--symlink-install” flag to avoid compiling every time a change is done in the source code

# 2.2 C++ nodes

- In development ...

# 3. CLI tools

- ROS graph

```
rqt_graph
```

## 3.1 Node tools

- List

```
ros2 node list
```

- Info

```
ros2 node info <node>
```

## 3.2 Remappings

- Allow to change the name of a node when it is launched:

```
ros2 run pkg <node_name> -ros-args -r __node:=<new_name>
```