ROS2 101

From Installation to Topics and Services

Roberto Marino, PhD

Computational Modeling for Robotics and Human Computer Interaction
University of Messina - FCRLab
roberto.marino@unime.it

May 26, 2026



UniMe
1548

Supported Platforms

ROS2 Jazzy Jalisco (recommended)

- Ubuntu 24.04 LTS (Noble) – Tier 1
- Long-term support until **May 2029**
- `amd64`, `arm64`, `arm32`

ROS2 Humble Hawksbill (LTS)

- Ubuntu 22.04 LTS (Jammy) – Tier 1
- Support until **May 2027**

Before you start

- Fresh Ubuntu install recommended
- 64-bit system required
- Internet connection needed
- ~2 GB disk space for desktop install
- Python 3.10+ already included

Check: `lsb_release -a` to confirm Ubuntu version

Step 1 – Locale & APT Setup

Set locale to UTF-8

```
sudo apt update && sudo apt install -y 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
```

Add the ROS2 APT repository

```
sudo apt install -y software-properties-common curl

sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc \
     | sudo gpg --dearmor -o /usr/share/keyrings/ros-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) \
  signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] \
  http://packages.ros.org/ros2/ubuntu \
  $(. /etc/os-release && echo $UBUNTU_CODENAME) main" \
  | sudo tee /etc/apt/sources.list.d/ros2.list

sudo apt update
```

Step 2 – Install ROS2

Desktop (recommended)
Includes RViz2, rqt, demos

```
sudo apt install -y \
  ros-jazzy-desktop
```

Base only
Core libs, no GUI tools

```
sudo apt install -y \
  ros-jazzy-ros-base
```

Install Python tools and dev utilities

```
sudo apt install -y \
  python3-colcon-common-extensions \
  python3-rosdep \
  python3-pip \
  ros-dev-tools

sudo rosdep init
rosdep update
```

Step 3 – Source & Verify

Source the setup script

```
source /opt/ros/jazzy/setup.bash

# Add to .bashrc so every terminal has ROS2 ready
echo "source /opt/ros/jazzy/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Quick smoke test

```
ros2 --version

# Run the classic talker/listener demo
ros2 run demo_nodes_py talker       # terminal 1
ros2 run demo_nodes_py listener     # terminal 2
```

If you see `"Hello World: 1"` in the listener – **installation is complete**.

Step 4 – Create a Workspace

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
colcon build
source install/setup.bash
echo "source ~/ros2_ws/install/setup.bash" >> ~/.bashrc
```

Create a Python package

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python my_robot \
    --dependencies rclpy std_msgs geometry_msgs \
                   sensor_msgs std_srvs

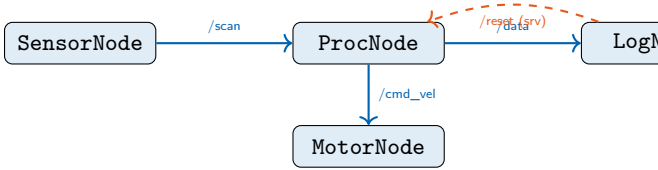
# Build and source
cd ~/ros2_ws
colcon build --packages-select my_robot
source install/setup.bash
```

- `ament_python` – pure Python package (no C++ compiler needed)
- `-symlink-install` avoids rebuilding when you only edit `.py` files

The ROS2 Computation Graph

- **Node** – independent process, identified by a name
- **Topic** – anonymous pub/sub channel, asynchronous
- **Service** – synchronous request/response call
- **Action** – long-running task with feedback
- **Parameter** – runtime-configurable variable per node

Solid = topic Dashed = service call



Key CLI Commands

Nodes & Topics

```
ros2 node list
ros2 node info /my_node
ros2 topic list
ros2 topic echo /scan
ros2 topic hz   /scan
```

Services & Parameters

```
ros2 service list
ros2 service call /enable \
  std_srvs/srv/SetBool \
  "{data: true}"
ros2 param list   /my_node
ros2 param get    /my_node kp
ros2 param set    /my_node kp 2.5
ros2 param dump   /my_node
```

Minimal Node

Every Python node inherits from `rclpy.node.Node`.

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

class MinimalNode(Node):

    def __init__(self):
        super().__init__('minimal_node')
        self.get_logger().info(
            'Node started!')

def main(args=None):
    rclpy.init(args=args)
    node = MinimalNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Key calls

- `rclpy.init()` – initialises DDS
- `rclpy.spin()` – event loop (blocking)
- `destroy_node()` – releases resources
- `rclpy.shutdown()` – cleans up DDS
- `get_logger()` – thread-safe logger

Run

```
ros2 run my_robot \
       minimal_node
```

Periodic Timer

`create_timer(period_s, callback)` fires at a fixed rate. It honours `/use_sim_time`.

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

class TimerNode(Node):

    def __init__(self):
        super().__init__('timer_node')
        self._count = 0
        # create_timer(period_seconds, callback)
        self._timer = self.create_timer(0.5, self._timer_callback)

    def _timer_callback(self):
        self._count += 1
        t = self.get_clock().now().nanoseconds / 1e9
        self.get_logger().info(f'[tick {self._count:3d}]  t = {t:.2f} s')

def main(args=None):
    rclpy.init(args=args)
    rclpy.spin(TimerNode())
    rclpy.shutdown()
```

Publisher

```python
from rclpy.node import Node
from geometry_msgs.msg import Twist
import rclpy

class VelPublisher(Node):

    def __init__(self):
        super().__init__('vel_publisher')
        # create_publisher(MsgType, topic, qos)
        self._pub = self.create_publisher(
            Twist, '/cmd_vel', 10)
        self.create_timer(0.1, self._publish)
        self.get_logger().info(
            'Publishing /cmd_vel at 10 Hz')

    def _publish(self):
        msg = Twist()
        msg.linear.x  = 0.5   # m/s forward
        msg.angular.z = 0.1   # rad/s
        self._pub.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    rclpy.spin(VelPublisher())
    rclpy.shutdown()
```

Common message types

Package	Type
std_msgs	String
std_msgs	Float64
geometry_msgs	Twist
geometry_msgs	Pose
sensor_msgs	LaserScan
sensor_msgs	Image
nav_msgs	Odometry

The second arg (10) is the **QoS queue depth**: messages buffered when the network is slow.

Subscriber

```python
import math, rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan

class ScanSubscriber(Node):

    def __init__(self):
        super().__init__('scan_subscriber')
        # create_subscription(Type, topic,
        #                     callback, qos)
        self.create_subscription(
            LaserScan,
            '/scan',
            self._scan_cb,
            10)
        self.get_logger().info(
            'Subscribed to /scan')

    def _scan_cb(self, msg: LaserScan):
        valid = [
            r for r in msg.ranges
            if math.isfinite(r)
            and msg.range_min <= r
                              <= msg.range_max
        ]
        if valid:
            self.get_logger().info(
              f'Nearest: {min(valid):.3f} m')

def main(args=None):
    rclpy.init(args=args)
    rclpy.spin(ScanSubscriber())
    rclpy.shutdown()
```

QoS – best-effort vs reliable

```python
from rclpy.qos import (
  QoSProfile,
  ReliabilityPolicy)

qos = QoSProfile(
  depth=10,
  reliability=
    ReliabilityPolicy
    .BEST_EFFORT)

self.create_subscription(
  LaserScan,
  '/scan',
  self._scan_cb,
  qos)
```

Publisher and subscriber QoS **must be compatible** or the connection is silently dropped.

Service Server

```python
import rclpy
from rclpy.node import Node
from std_srvs.srv import SetBool

class MotorServer(Node):

    def __init__(self):
        super().__init__('motor_server')
        self._motor_on = False
        # create_service(SrvType, name, callback)
        self.create_service(SetBool, '/enable_motor', self._handle)
        self.get_logger().info('Motor server ready.')

    def _handle(self, request: SetBool.Request,
                response: SetBool.Response) -> SetBool.Response:
        self._motor_on  = request.data
        response.success = True
        response.message = 'Motor ON' if self._motor_on else 'Motor OFF'
        self.get_logger().info(response.message)
        return response

def main(args=None):
    rclpy.init(args=args)
    rclpy.spin(MotorServer())
    rclpy.shutdown()
```

Service Client (async)

```python
import rclpy
from rclpy.node import Node
from std_srvs.srv import SetBool

class MotorClient(Node):

    def __init__(self):
        super().__init__('motor_client')
        self._client = self.create_client(
            SetBool, '/enable_motor')

        while not self._client\
                .wait_for_service(1.0):
            self.get_logger().info(
                'Waiting for server...')

        self._send(True)
        self._timer = self.create_timer(
            3.0, self._delayed_disable)

    def _send(self, enable: bool):
        req = SetBool.Request()
        req.data = enable
        future = self._client.call_async(req)
        future.add_done_callback(
            self._on_response)

    def _on_response(self, future):
        r = future.result()
        self.get_logger().info(
            f"Response: '{r.message}'")

    def _delayed_disable(self):
        self._send(False)
        self._timer.cancel()
```

call_async vs call

- `call_async()` – non-blocking, returns a Future; callback invoked by the executor
- `call()` – blocking; **never** use inside a spinning node (deadlock!)

Custom service files

Create `srv/SetSpeed.srv`:

```
# Request
float64 target_speed
---
# Response
bool   success
string message
```

Add to `package.xml` and `setup.py` via `rosidl_generate_interfaces`.

Declaring and Reading Parameters

```python
from rclpy.node import Node
from rcl_interfaces.msg import (
    SetParametersResult)

class ControlNode(Node):

    def __init__(self):
        super().__init__('control_node')

        # Declare with default values
        self.declare_parameter('kp',          1.0)
        self.declare_parameter('kd',          0.1)
        self.declare_parameter('max_vel',     2.0)
        self.declare_parameter('robot_name','panda')

        # Read at startup
        self._read_params()

        # Callback on runtime change
        self.add_on_set_parameters_callback(
            self._on_change)

    def _read_params(self):
        self._kp   = self.get_parameter(
                         'kp').value
        self._kd   = self.get_parameter(
                         'kd').value
        self._name = self.get_parameter(
                         'robot_name').value
```

```python
    def _on_change(self, params):
        for p in params:
            self.get_logger().info(
              f'{p.name} = {p.value}')
            if (p.name == 'kp'
                    and p.value <= 0):
                return
                    SetParametersResult
                    (
                    successful=False,
                    reason='kp must be
                        > 0')
        self._read_params()
        return SetParametersResult(
            successful=True)
```

CLI

```
ros2 param list   /control_node
ros2 param get    /control_node kp
ros2 param set    /control_node kp
     3.0
ros2 param dump   /control_node \
     > params.yaml
```

Parameters from YAML

params.yaml

```yaml
control_node:
  ros__parameters:
    kp: 2.0
    kd: 0.3
    max_vel: 1.5
    robot_name: "panda"
```

Load in launch file

```python
Node(
  package='my_robot',
  executable='control_node',
  parameters=['params.yaml']
)
```

Load at runtime via CLI

```bash
ros2 run my_robot control_node \
  --ros-args \
  --params-file params.yaml

# Override one value inline
ros2 run my_robot control_node \
  --ros-args \
  --params-file params.yaml \
  -p kp:=3.5
```

setup.py & package.xml

setup.py

```
from setuptools import setup

package_name = 'my_robot'

setup(
    name=package_name,
    version='0.1.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/'
         'resource_index/packages',
         ['resource/' + package_name]),
        ('share/' + package_name,
         ['package.xml']),
        ('share/my_robot/launch',
         ['launch/robot.launch.py']),
    ],
    install_requires=['setuptools'],
    entry_points={
        'console_scripts': [
          'control_node = '
          'my_robot.control_node:main',
        ],
    },
)
```

package.xml (excerpt)

```
<package format="3">
  <name>my_robot</name>

  <buildtool_depend>
    ament_python
  </buildtool_depend>

  <depend>rclpy</depend>
  <depend>std_msgs</depend>
  <depend>geometry_msgs</depend>
  <depend>sensor_msgs</depend>
  <depend>std_srvs</depend>

  <export>
    <build_type>
      ament_python
    </build_type>
  </export>
</package>
```

Launch File

```python
# launch/robot.launch.py
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import (
    DeclareLaunchArgument)
from launch.substitutions import (
    LaunchConfiguration)

def generate_launch_description():
    fwd = DeclareLaunchArgument(
        'forward_speed',
        default_value='0.5')

    robot = Node(
        package='my_robot',
        executable='robot_node',
        name='robot_node',
        parameters=[{
            'forward_speed':
              LaunchConfiguration(
                'forward_speed'),
            'stop_dist': 0.4,
        }],
        remappings=[
          ('/cmd_vel','/robot/cmd_vel')],
        output='screen',
    )
```

Build & run

```
cd ~/ros2_ws
colcon build \
  --symlink-install
source install/setup.bash

# Launch
ros2 launch my_robot \
  robot.launch.py

# With argument
ros2 launch my_robot \
  robot.launch.py \
  forward_speed:=0.3
```

`--symlink-install` lets you edit `.py` files without rebuilding.

Summary & Next Steps

What we covered

- Installing ROS2 Jazzy on Ubuntu 24.04
- Workspace creation with `colcon`
- Nodes – `Node`, `spin`, timers
- Topics – pub/sub with QoS
- Services – request/response pattern
- Parameters – declare, read, change at runtime
- `setup.py`, `package.xml`, launch files

Next steps

- **Actions** – long-running tasks (`rclpy.action`)
- **tf2** – coordinate frame transforms
- **Lifecycle nodes** – managed state machine
- **Gazebo / RViz2** – simulation & visualisation
- **Nav2** – full navigation stack
- **MoveIt2** – motion planning

References

`https://docs.ros.org/en/jazzy`
`https://github.com/ros2/examples`
`https://roboticsbackend.com/ros2`

Happy coding with ROS2!