CSC398 introduction to Autonomous Robots
- Introduction into ROS2 (1) -

Ubbo Visser

Department of Computer Science
College of Arts and Sciences
University of Miami

September 2026

UNIVERSITY OF MIAMI
ROBOCANES

OVERVIEW

- ROS 2 architecture & philosophy
- DDS discovery, nodes, and topics
- ROS 2 CLI commands
- ament/colcon workspace and build system
- Launch-files
- Gazebo / Isaac Sim

WHAT IS ROS?

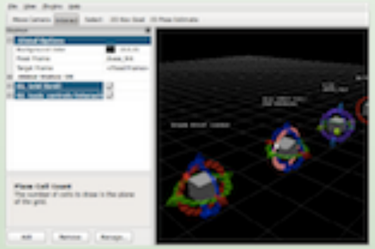
- ROS 2: Robot Operating System 2

=

Plumbing

- Process management
- Inter-process communication
- Device drivers

+



Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

+

Capabilities

- Control
- Planning
- Perception
- Mapping
- Manipulation

+

Ecosystem

- Package organization
- Software distribution
- Documentation
- Tutorials

Source: ros.org

HISTORY OF ROS / ROS 2

- Originally developed in 2007 at Stanford (AI Lab) and Willow Garage
- ROS 2 was designed for production robotics: distributed discovery, QoS, security, and real-time use
- Developed and maintained by the Open Source Robotics community
- Widely used in research and industry

Source: ros.org

ROS 2 PHILOSOPHY

- Distributed / peer-to-peer

  Nodes discover each other through DDS; no central ROS master is required.

- Standard communication

  Topics, services, actions, parameters, and ROS interfaces.

  Quality of Service (QoS)

  Communication behavior can be tuned for reliability, durability, history, and deadlines.

- Multi-language

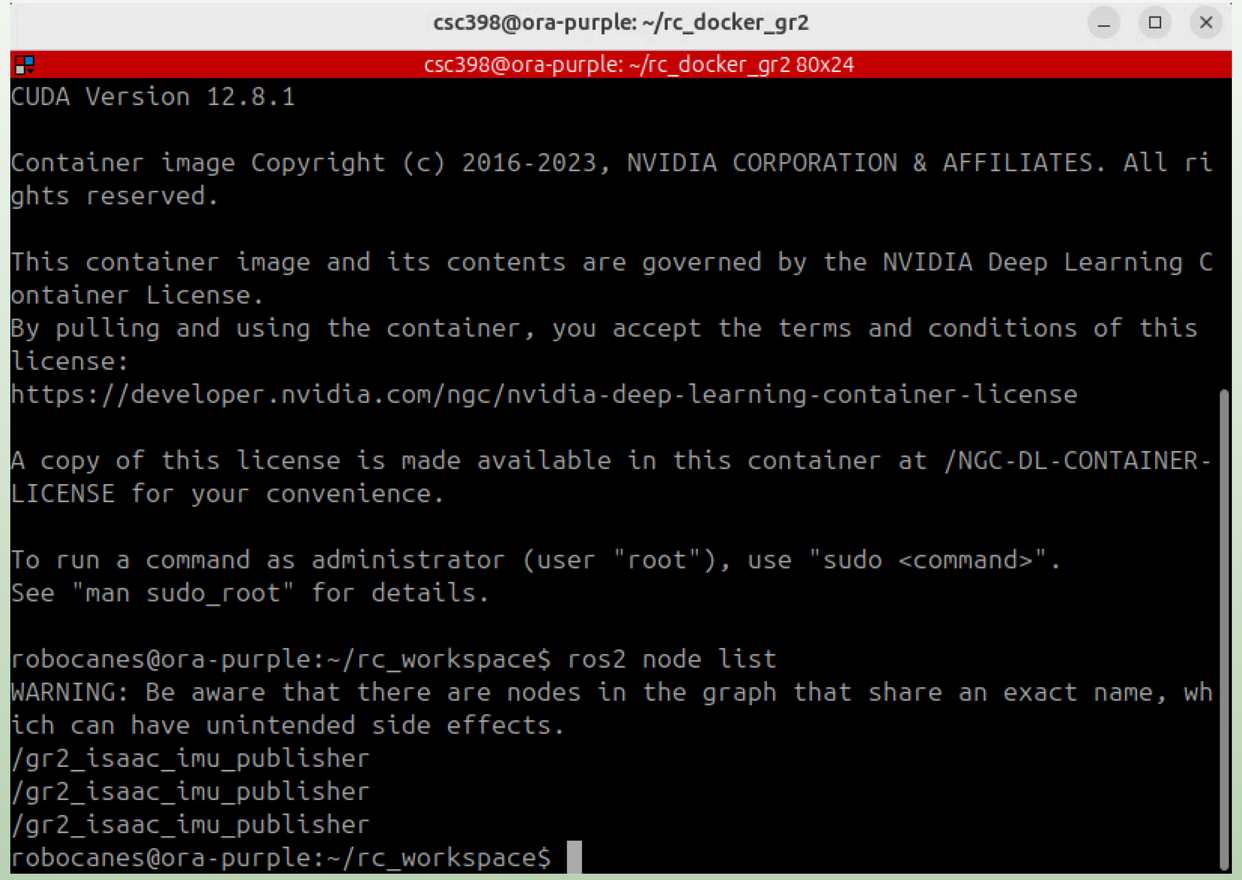
  Primary client libraries are rclcpp (C++) and rclpy (Python).

- Open source

  ROS 2 and most ecosystem software are open source.

ROS2 DISCOVERY

- DDS
- Discovery
- No ROS core/Master!
- Better QoS policies



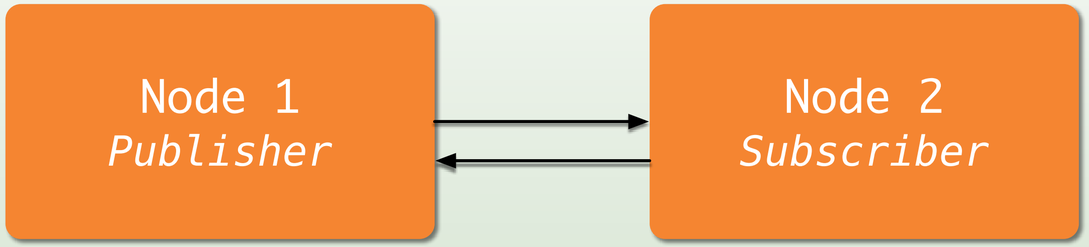
No Master/Core needs to be started

```
~$ros2 node list
```

Details: https://docs.ros.org/en/foxy/Concepts/About-Domain-ID.html

ROS NODES

- Executable program with a single purpose
- Needs to be compiled, can be individually compiled, executed and also managed (e.g. a program showing joint states)
- Organized in *packages*



Run node with

```
~$ros2 run package_name node_name
```

List active nodes

```
~$ros2 node list
```

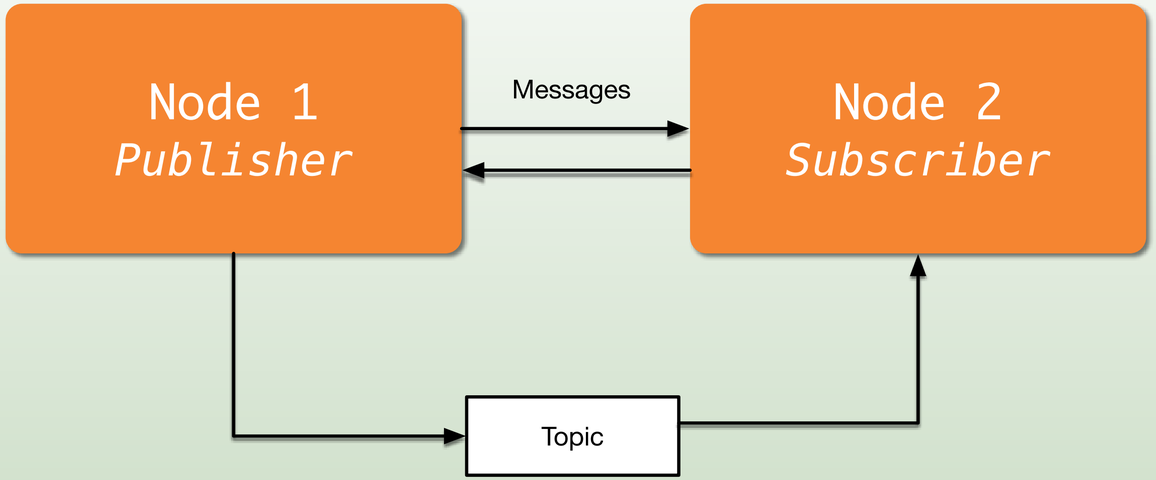
Node information

```
~$ros2 node info node_name
```

Details: ROS2 nodes

ROS TOPICS

- Nodes communicate via *topics*
  - *Publish* or *subscribe* to topics
  - Common: *1* publisher, *n* subscribers
- A topic stands for a flow or stream of data, *messages* in ROS language



List active topics

```
~$ros2 topic list
```

Subscribe and print the contents of a topic with

```
~$ros2 topic echo /topic
```

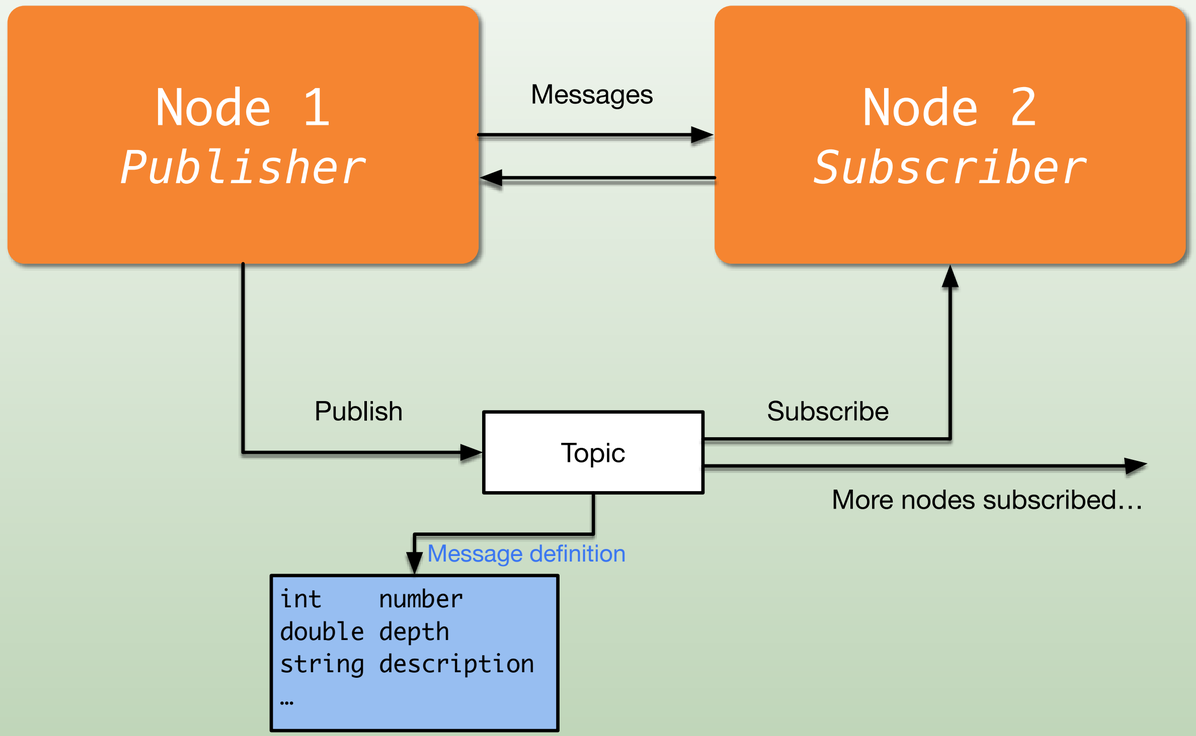
Show information about a topic with

```
~$ros2 topic info /topic
```

Details: ROS2 nodes

ROS MESSAGES

- Data structure defines *type* of topic
- Nested structure of primitive data types and arrays
- Defined in .msg files



List type of topic

```
~$ros2 topic type /topic
```

Publish a message to a topic

```
~$ros2 topic pub /topic pkg/msg/Type “{field: value}”
```

Details at
http://wiki.ros.org/Messages

ROS MESSAGES

- Pose Stamped Example

geometry_msgs/Point Message

```
# This contains the position of a
point in free space
float64 x
float64 y
float64 z
```

sensor_msgs/Image Message

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
Geometry msgs/Pose pose
  geometry_msgs/Point position
     float64 x
     float64 y
     float64 z
  geometry_msgs/Quaternion
 orientation
      float64 x
      float64 y
      float64 z
      float64 w
```

EXAMPLE

- Run and analyse *talker* node

Run talker node

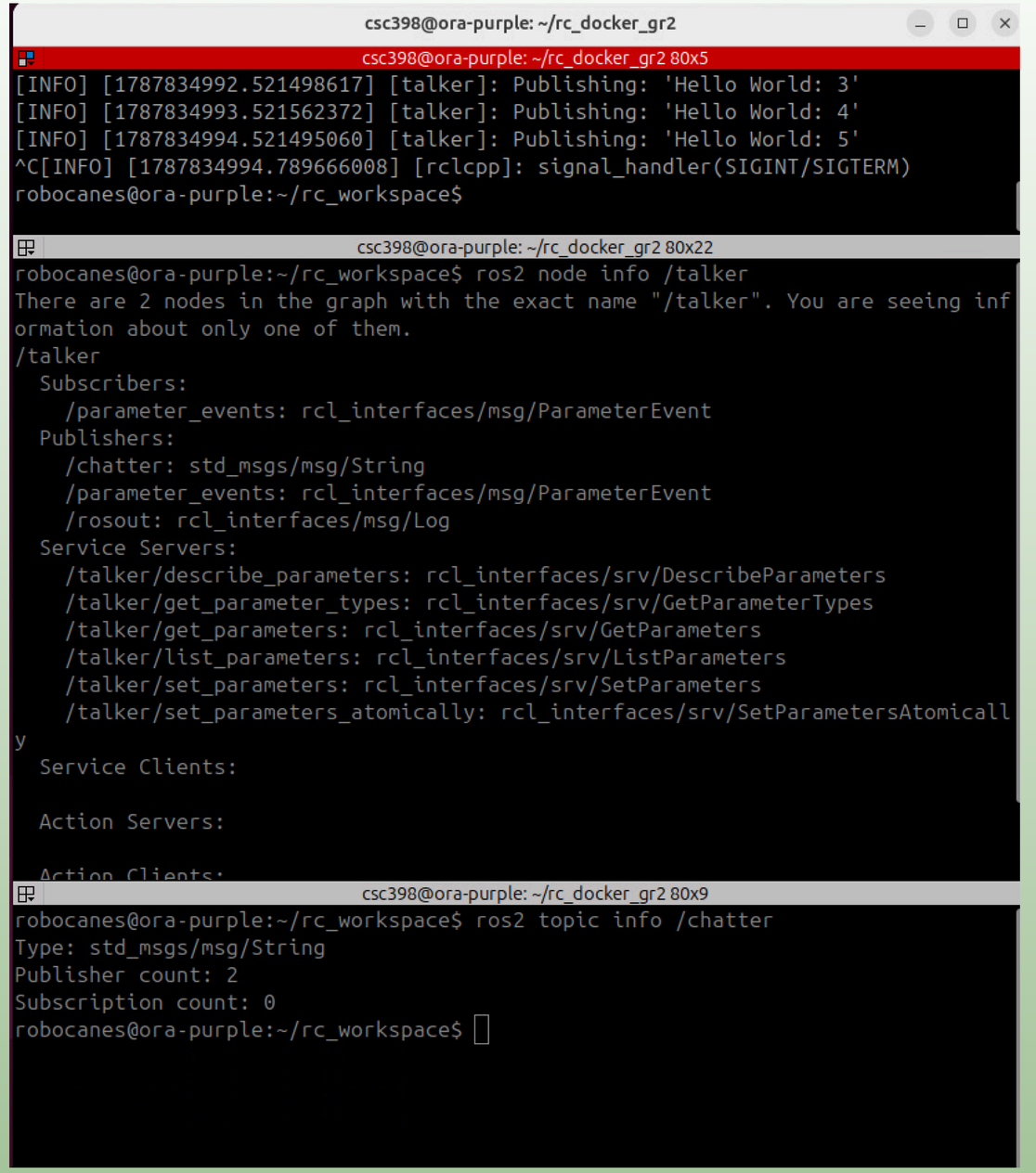
```
~$ros2 run demo_nodes_cpp talker
```

Info about node in another terminal

```
~$ros2 node info /talker
```

Info about topic in a 3rd terminal

```
~$ros2 topic info /chatter
```

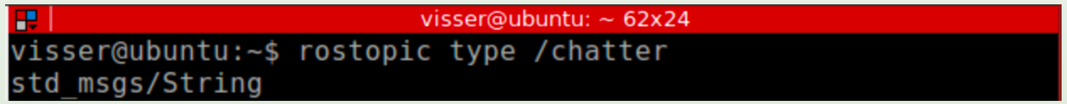


EXAMPLE

- Analyse *chatter* topic

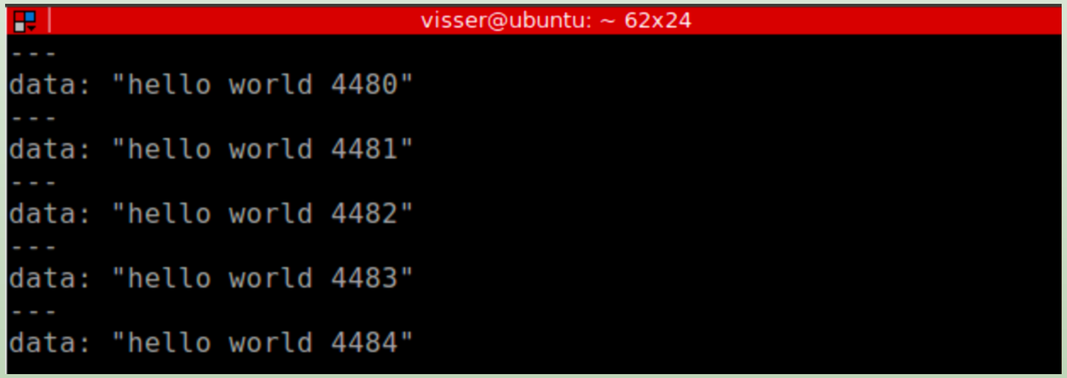
Type of topic

```
~$ros2 topic type /chatter
```



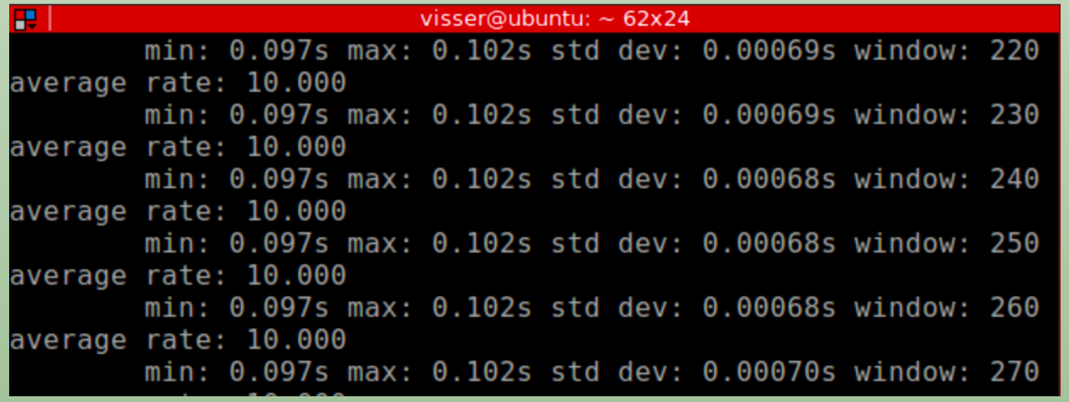
Message contents

```
~$ros2 topic echo /chatter
```



Frequency of publishing

```
~$ros2 topic hz /chatter
```

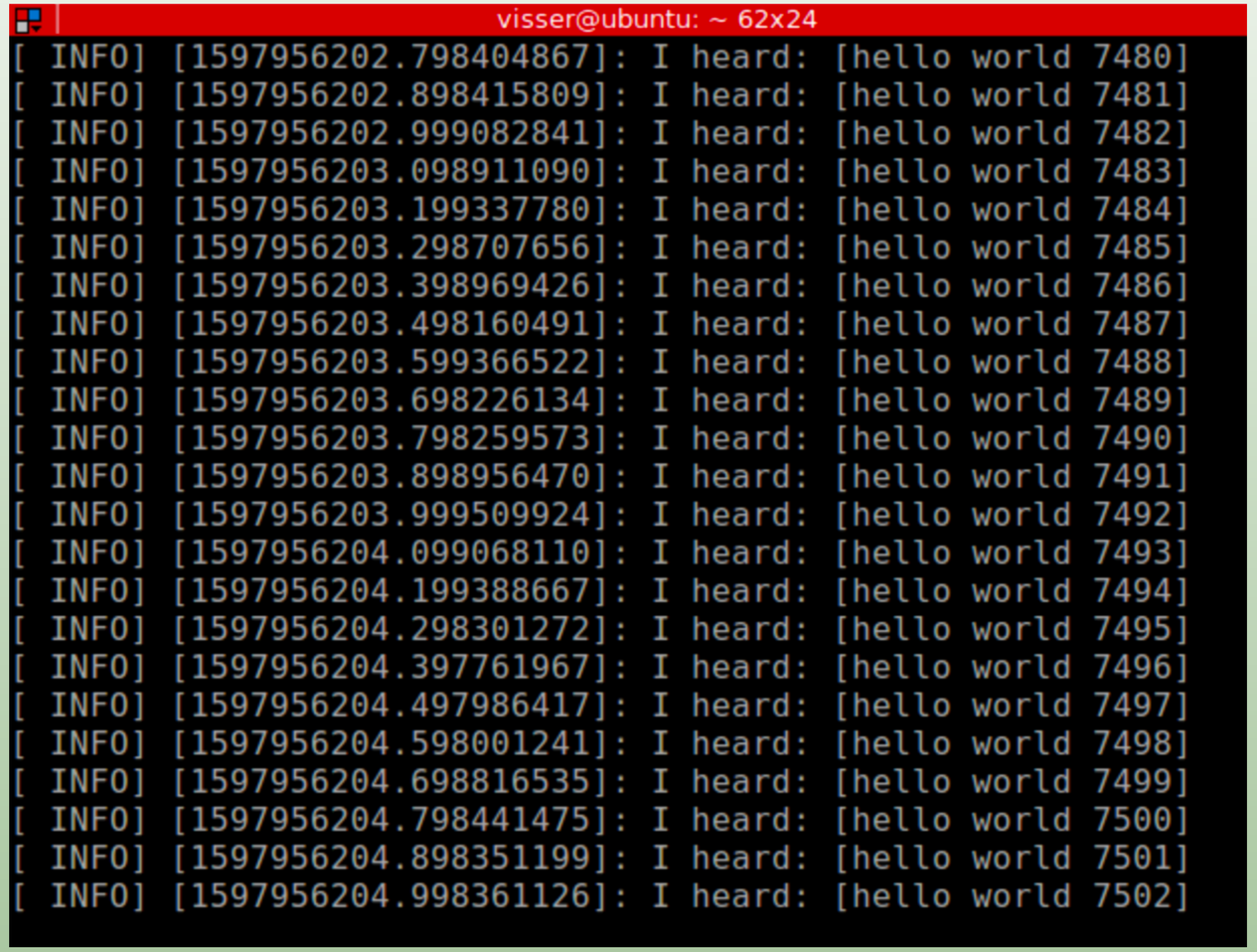


EXAMPLE

- Starting new *listener* node

Type of topic

```
~$ros2 run demo_nodes_cpp listener
```

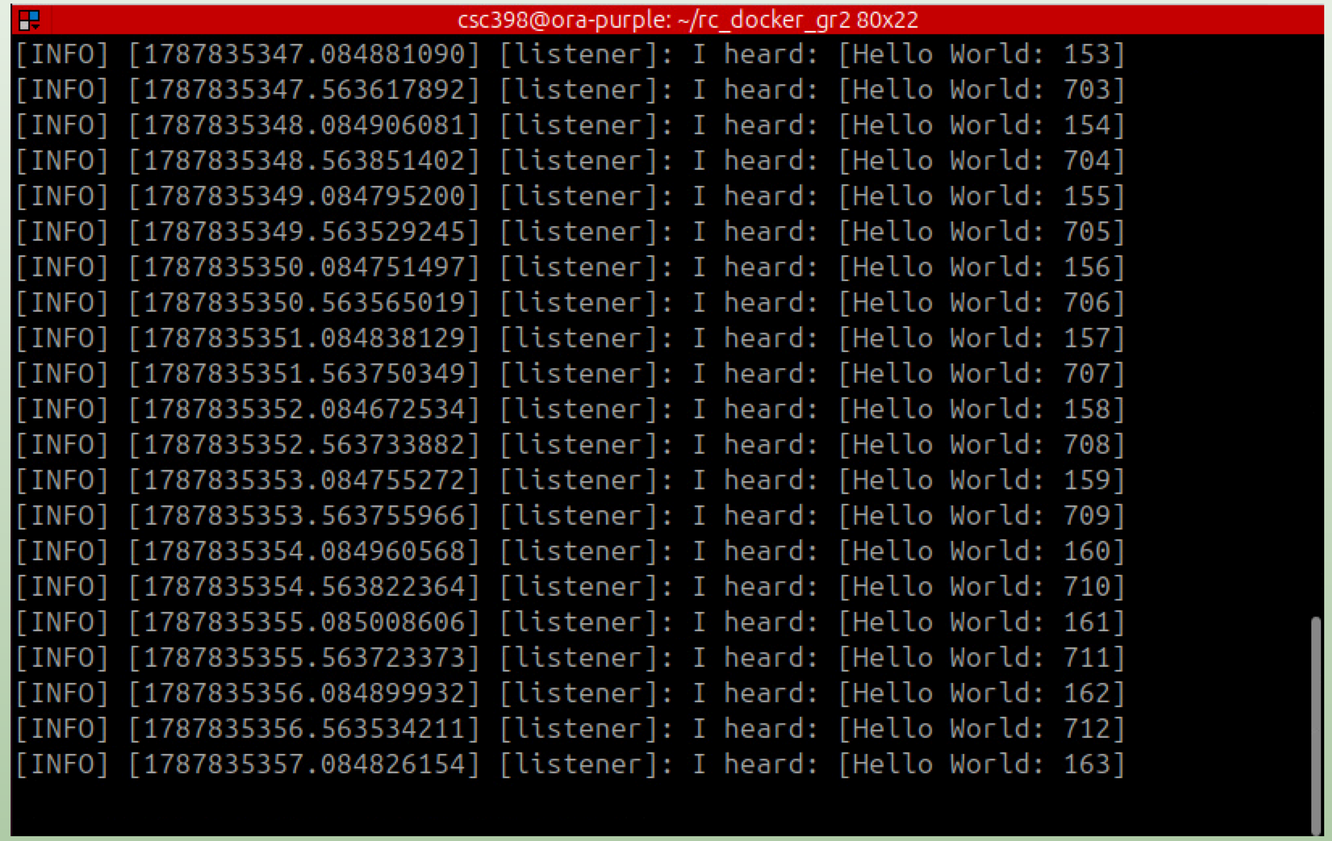


EXAMPLE

- Starting new *listener* node

Type of topic

```
~$ros2 run demo_nodes_cpp listener
```

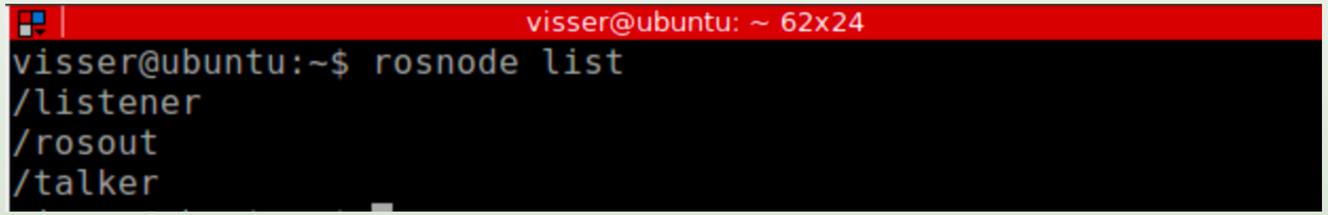


EXAMPLE

- Back to Terminal 3: analyse

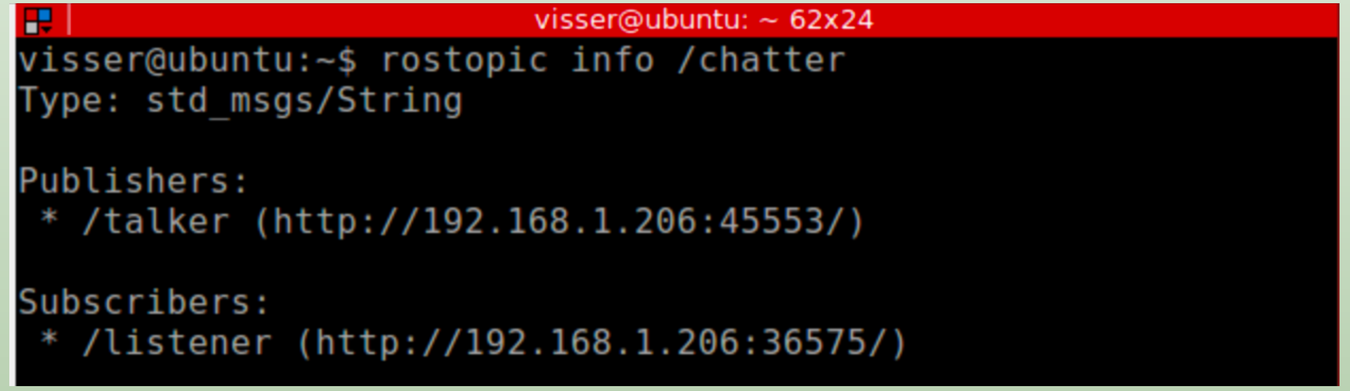
New listener node visible

```
~$ros2 node list
```



Show the connection of the nodes over the chatter topic with

```
~$ros2 topic info /chatter
```



EXAMPLE

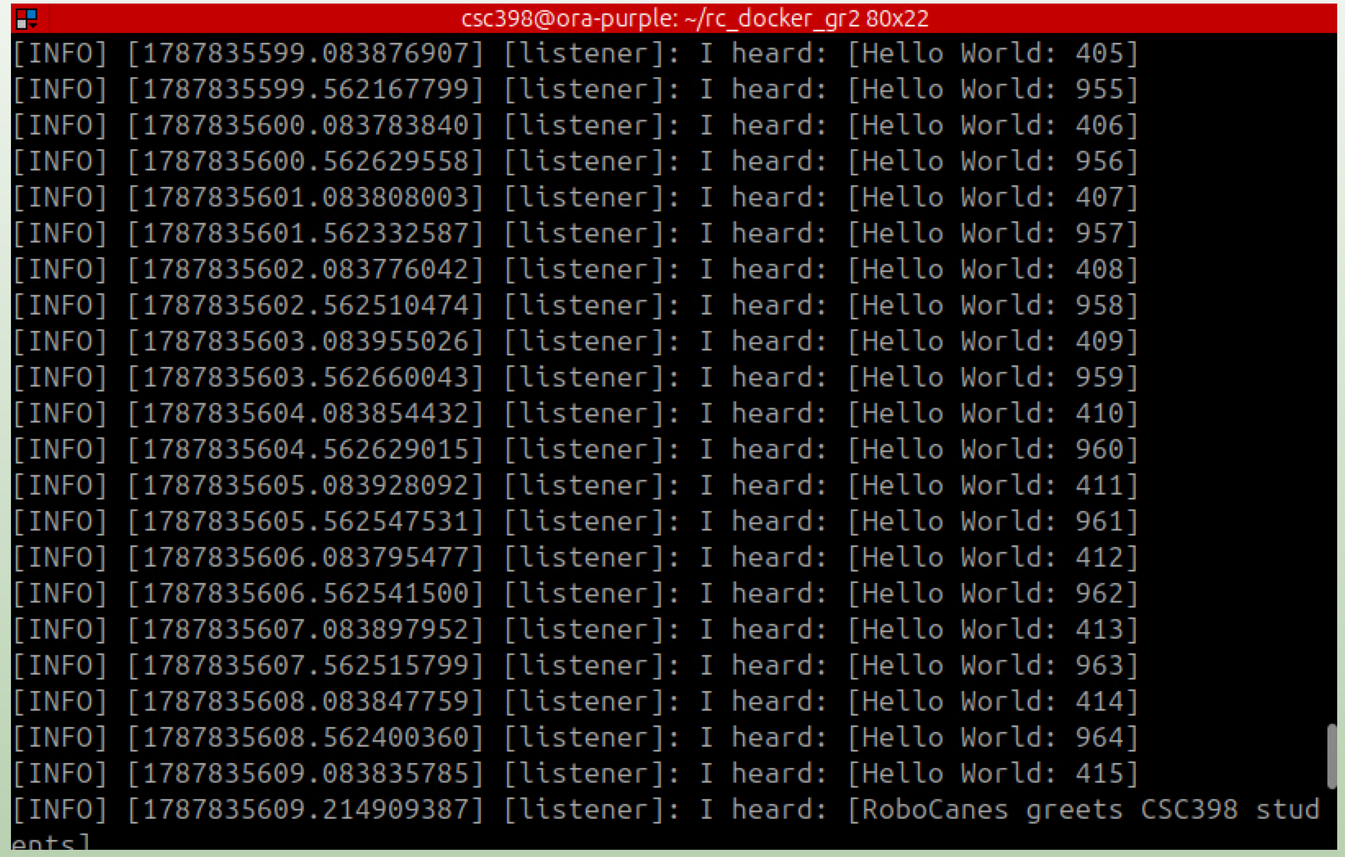
- Terminal 3: publish own message in terminal

Close talker node in T2, Ctrl-C

Publish own message

```
~$ros2 topic pub /chatter
std_msgs/String "data: 'RoboCanes
greets CSC398 students'"
```

Message shows up in T4 (listener)



ROS WORKSPACE ENVIRONMENT

- Defines context for the current workspace
- Default workspace loaded with

```
~$source /opt/ros/humble/setup.bash
```

Overlay your catkin workspace with

```
~$cd [ROS2_WS]
~$source install/setup.bash
```

Check your workspace with

```
~$echo $AMENT_PREFIX_PATH
```

Details: https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html

COLCON BUILD SYSTEM

- *ROS 2 packages use ament as the build system and colcon to build workspaces.*
- Navigate to your ROS 2 workspace, e.g.:

```
~$cd ~/[ROS2_WS]
```

- Build the workspace or selected packages

```
~$colcon build –packages-select [package_name]
```

- Update your environment after a new package is build

```
~$source install/setup.bash
```

Details: https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html

COLCON BUILD SYSTEM

- The catkin workspace contains the following spaces

The *source space* contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

The *build space* is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

The *development (install) space* is where built targets are placed (prior to being installed).

Clean entire workspace with

```
~$rm -rf build install log
```

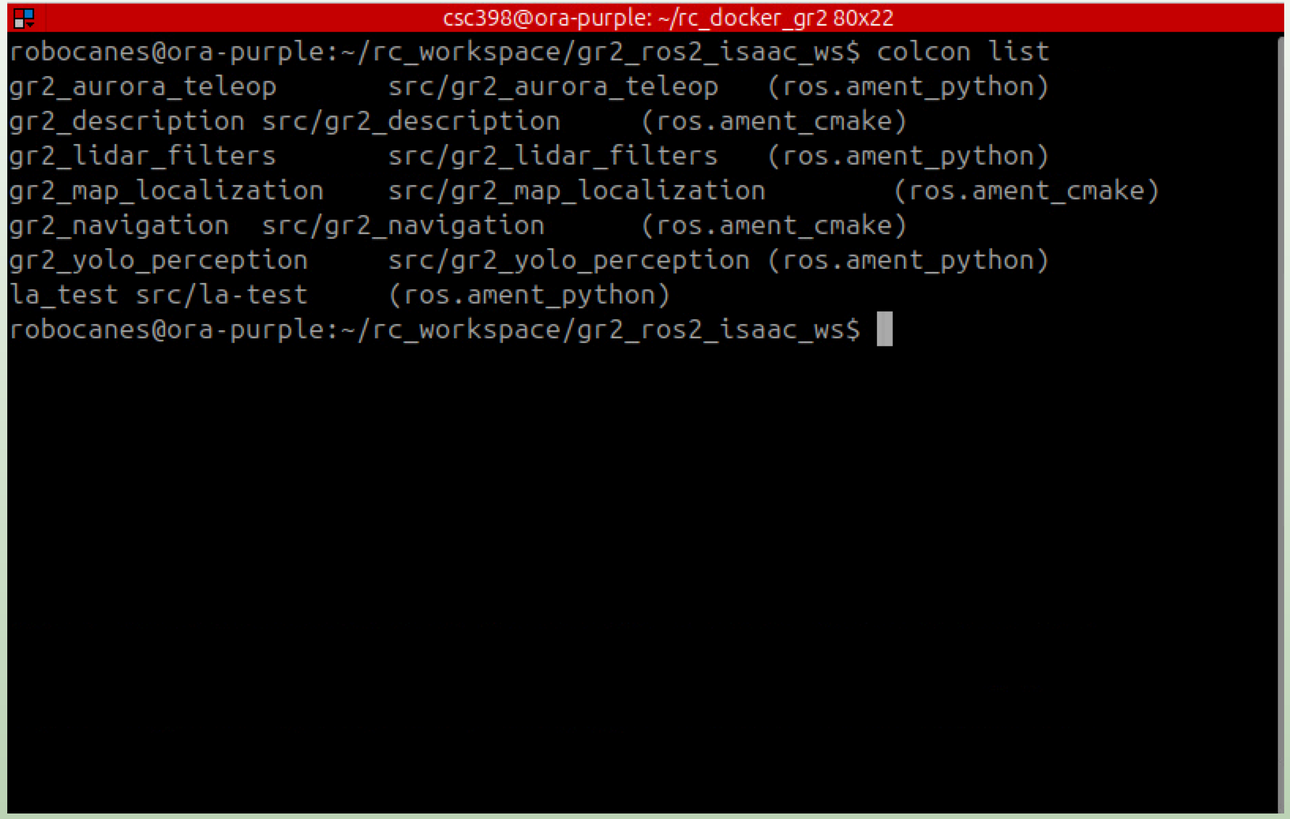
COLCON BUILD SYSTEM

- colcon provides workspace-wide and package-selective builds.

```
~$colcon list
```

- Example: set CMake build type to Release

```
~$colcon build --cmake-args
-DCMAKE_BUILD_TYPE=Release
```



Details: https://colcon.readthedocs.io/en/released/

EXAMPLE

- In your ROS2 workspace

```
~$cd [ROS2_WS]
```

- Build packages

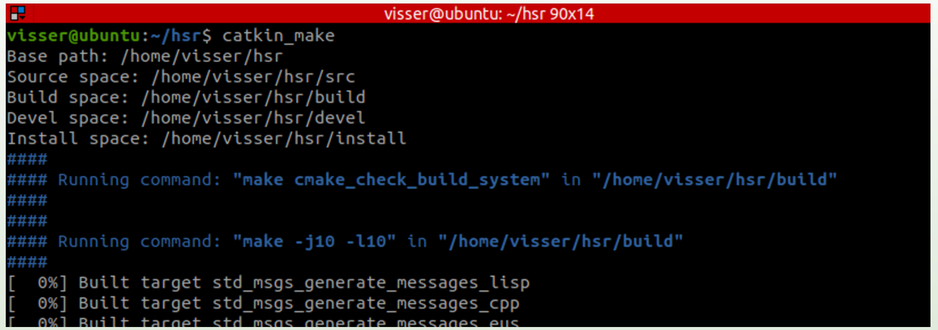
```
~$colcon build
```

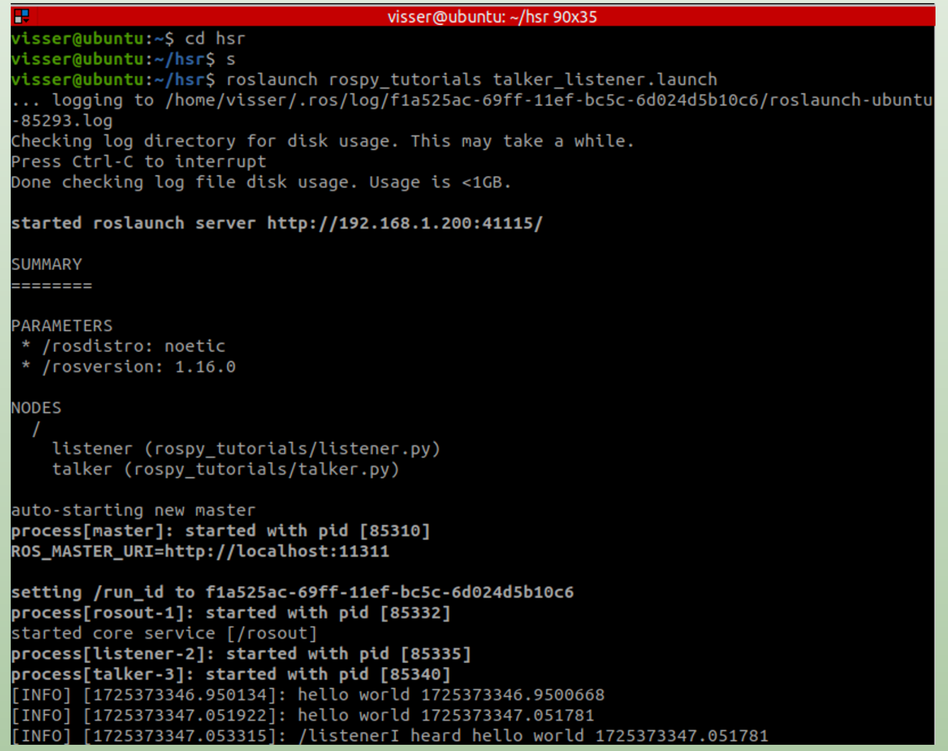
- Re-source your workspace

```
~$source install/setup.bash
```

- Launch node

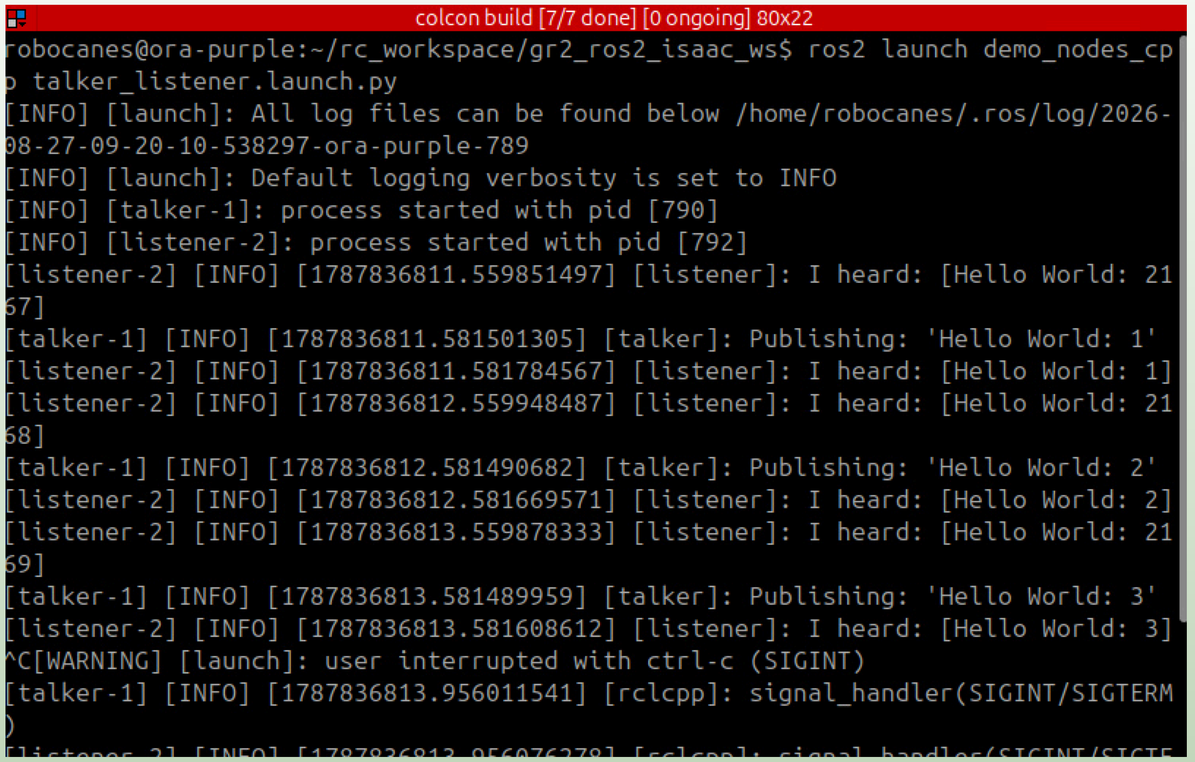
```
~$ros2 launch demo_nodes_cpp talker_listener.launch.py
```





ROS2 LAUNCH

- *ROS 2 launch starts and configures multiple nodes.* A launch file also acts like a config file
- Launch descriptions are commonly written in Python (.launch.py); XML and YAML are also supported.



ros2 launch demo_nodes_cpp talker_listener.launch.py

Start launch file from package

```
~$ros2 launch package_name file_name.launch.py
```

Details: https://docs.ros.org/en/humble/Tutorials/Intermediate/Launch/Launch-Main.html

ROS 2 PYTHON LAUNCH STRUCTURE

talker_listener.launch.xml

```
talker_listener.launch.xml
1   <launch>
2     <node pkg="demo_nodes_cpp" exec="talker" output="screen" />
3     <node pkg="demo_nodes_cpp" exec="listener" output="screen" />
4   </launch>
5
```

- **LaunchDescription:** collection of launch actions
- **node**: action that starts a ROS 2 node
- **package:** package containing the executable
- **executable:** executable to run
- **name:** node name
- **parameters/remappings:** node configuration
- **output**='screen': print node output to the terminal

Details: https://docs.ros.org/en/humble/Tutorials/Intermediate/Launch/Creating-Launch-Files.html

ROS 2 LAUNCH ARGS

- Create reusable launch files with DeclareLaunchArgument

```
DeclareLaunchArgument('arg_name', default_value='default_value')
```

- Read arguments with LaunchConfiguration

```
LaunchConfiguration('arg_name')
```

- Set arguments on the command line

```
Ros2 launch pkg_launch_file.launch.py arg_name:=value
```

```
<?xml version="1.0"?>
<launch>

  <!-- these are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="extra_gazebo_args" default=""/>
  <arg name="gui" default="true"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>
  <arg name="verbose" default="true"/>
  <arg name="output" default="screen"/>
  <arg name="world" default="gazebo_ros_range"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/test/test_worlds/$(arg world).worl
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="extra_gazebo_args" value="$(arg extra_gazebo_args)"/>
    <arg name="gui" value="$(arg gui)"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
    <arg name="verbose" value="$(arg verbose)"/>
    <arg name="output" value="$(arg output)"/>
  </include>

</launch>
```

Details: https://docs.ros.org/en/humble/Tutorials/Intermediate/Launch/Using-Substitutions.html

ROS 2 NESTED LAUNCH FILES

- Use IncludeLaunchDescription for modular launch files

```
IncludeLaunchDescription(...)
```

- Find package share directories with ament_index_python

```
get_package_share_directory('package_name')
```

- Pass launch arguments to included descriptions

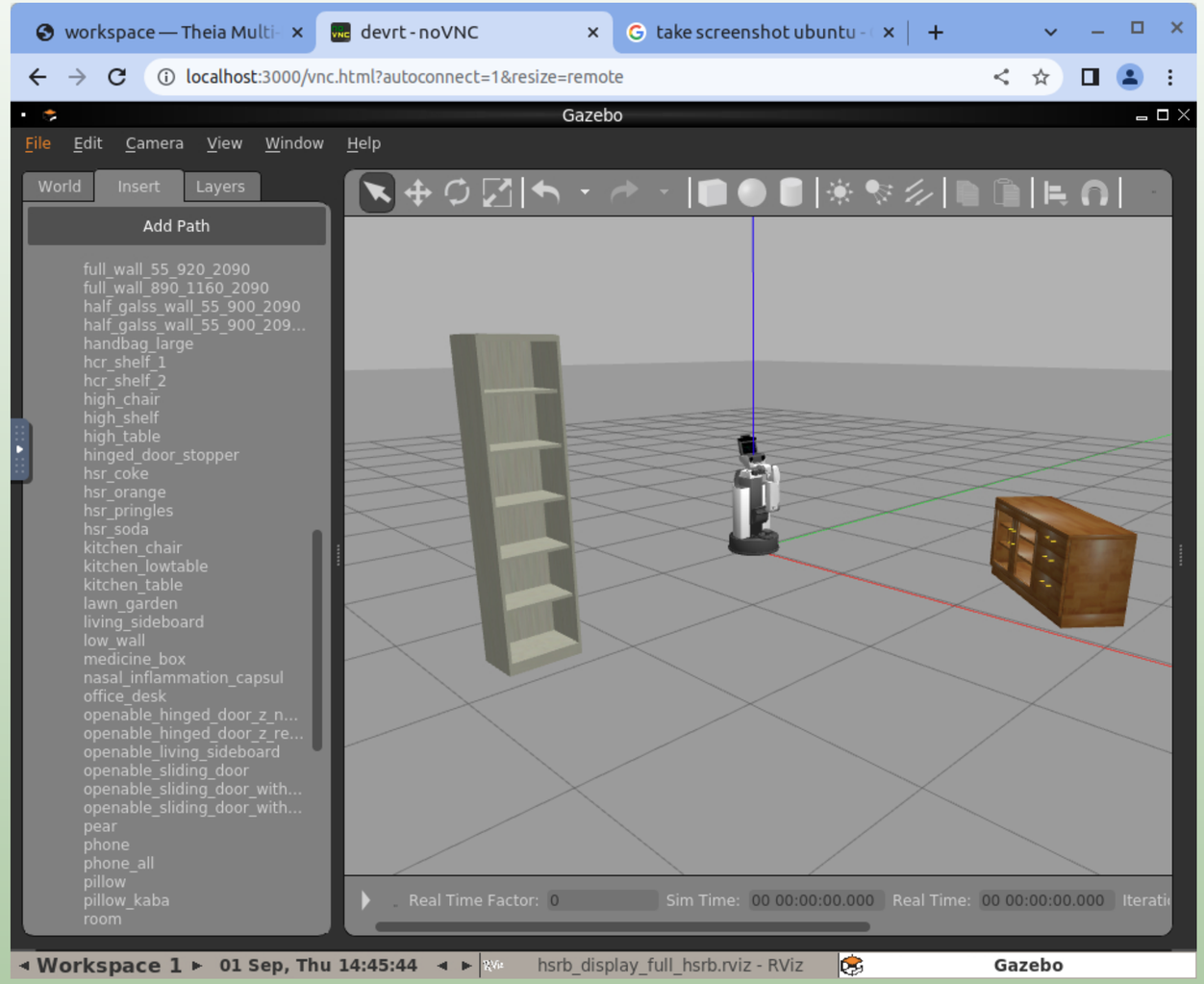
```
launch_arguments={'arg_name': 'value'}.items()
```

```python
def generate_launch_description():
    package_path = get_package_share_directory("gr2_navigation")
    nav2_bringup_path = get_package_share_directory("nav2_bringup")
    default_params_file = os.path.join(package_path, "config", "gr2_nav2_params.yaml")

    params_file = LaunchConfiguration("params_file")
    use_sim_time = LaunchConfiguration("use_sim_time")
    autostart = LaunchConfiguration("autostart")

    return LaunchDescription(
        [
            DeclareLaunchArgument("params_file", default_value=default_params_file),
            DeclareLaunchArgument("use_sim_time", default_value="true"),
            DeclareLaunchArgument("autostart", default_value="true"),
            IncludeLaunchDescription(
                PythonLaunchDescriptionSource(
                    os.path.join(nav2_bringup_path, "launch", "navigation_launch.py")
                ),
                launch_arguments={
                    "params_file": params_file,
                    "use_sim_time": use_sim_time,
                    "autostart": autostart,
                }.items(),
            ),
        ]
    )
```

Details: https://docs.ros.org/en/humble/Tutorials/Intermediate/Launch/Using-ROS2-Launch-For-Large-Projects.html

GAZEBO SIMULATOR

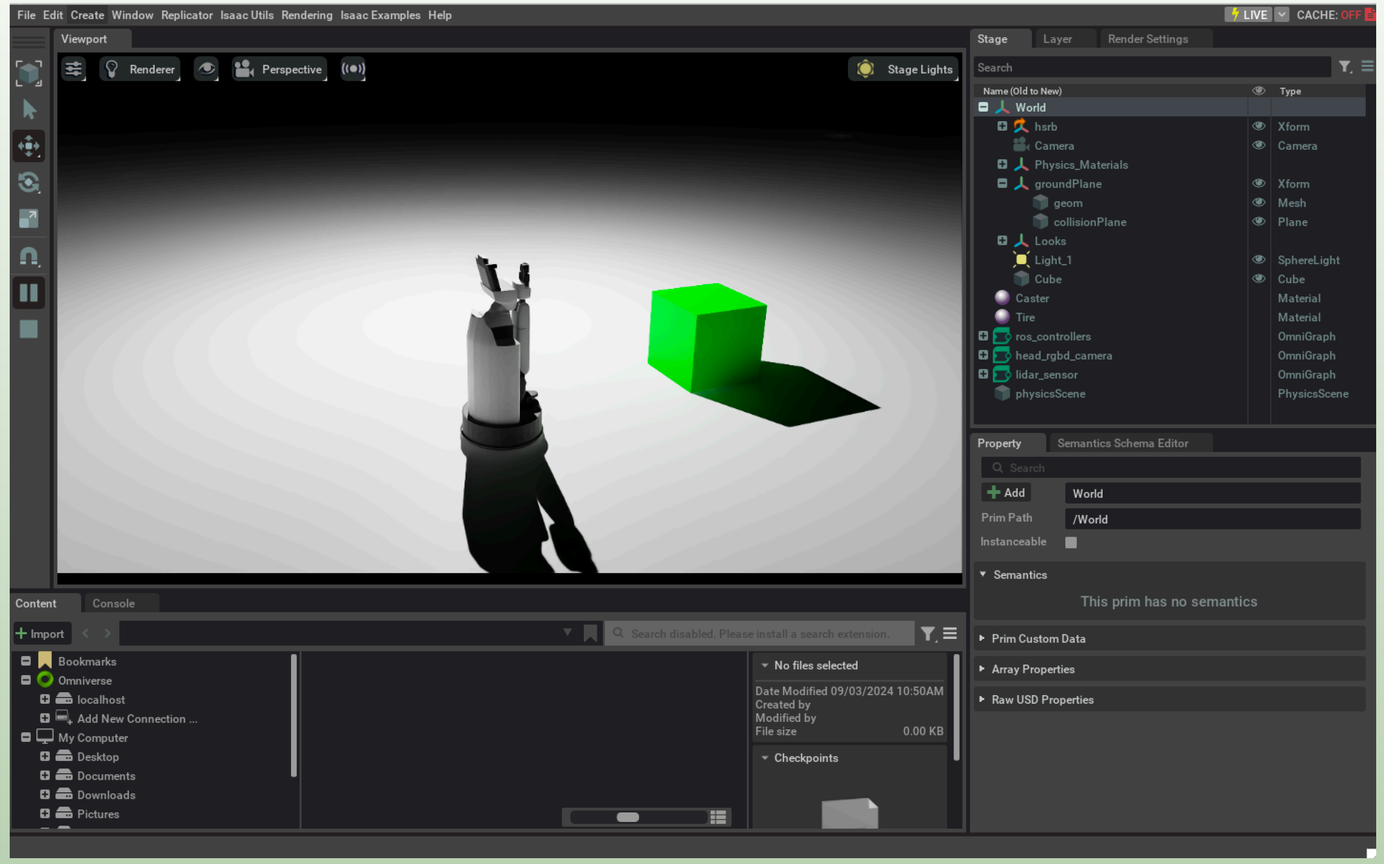
- Simulate 3D rigid-body dynamics
- Simulate a variety of sensors including noise
- 3D visualization and user interaction
- Includes a database of many robots and environments (Gazebo worlds)
- Provides a ROS interface
- Extensible with plugins




```
~$ros2 launch ros_gz_sim gz_sim.launch.py
```

ISAAC SIM

- Simulate 3D rigid-body dynamics
- Simulate a variety of sensors including noise
- 3D visualization and user interaction
- Includes a database of many robots and environments
- Provides a ROS interface
- Extensible with plugins




```
$ python run_isaac_gr2_aurora.py
```

FURTHER REFERENCES

- ROS 2 Documentation
  - https://docs.ros.org/en/humble/
- Installation
  - https://docs.ros.org/en/humble/Installation.html
- Tutorials
  - https://docs.ros.org/en/humble/Tutorials.html
- ROS Index
  - https://index.ros.org/
- ROS 2 Concepts
  - https://docs.ros.org/en/humble/Concepts.html
- ROS 2 CLI
  - https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools.html
- colcon
  - https://colcon.readthedocs.io/
- ROS 2 Design
  - https://design.ros2.org/

ACKNOWLEDGEMENT

Material adapted from ROS 2 documentation and the original ETH Zürich ROS introduction (https://rsl.ethz.ch/)