Introduction to ROS

Marco Bernardi

Internet of Things 2020/2021

Robotics Revolution

Problems in robotic development….before ROS

- Lack of standards
- Little code reusability
- New robot in the lab = start re-coding from scratch
  - Keeping reinventing (or rewriting) device drivers
  - Access to robot’s interfaces,
  - Management of on-board processes,
  - …

What is ROS?

- ROS is an open-source robot operating system
- A set of software libraries and tools that help you build robot applications that work across a wide variety of robotic platforms
- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory and development continued at Willow Garage. Since 2013 managed by OSRF (Open Source Robotics Foundation)

ROS Main Features

- ROS has two "sides"
  - The operating system side , which provides standard operating system services such as:
    - hardware abstraction
    - Low level device control
    - implementation of commonly used functionality
    - message passing between processes
    - package management
  - A suite of user contributed packages that implement common robot functionality such as SLAM, planning, perception, vision, manipulation, etc.

ROS Main Features 2

- Code reuse(exec. nodes, groupedin packages)
- Distributed, modular design (scalable)
- Language independent(C++, Python, Java, …)
- ROS-agnostic libraries(code is ROS in dep.)
- Easy testing (ready-to-use)
- Vibrant community & collaborative environment

ROS =plumbing+ tools+ capabilities+ ecosystem

Plumbing	Tools	Capabilities	Ecosystem
publish-subscribe messaging infrastructure designed to support the quick and easy construction of distributed computing systems.	tools for configuring, starting, introspecting, debugging, visualizing, logging, testing, and stopping distributed computing systems.	a broad collection of libraries that implement useful robot functionality, with a focus on mobility, manipulation, and perception.	ROS is supported and improved by a large community, with a strong focus on integration and documentation.

Robot specific features

- Provides tools for
  - Message Definition
  - Process Control
  - File System
  - Build System
- Provides basic functionalities like:
  - Device Support
  - Navigation
  - Control of Manipulator
  - Object Recognition


Fraunhofer IPA Care-O-bot


Videre Erratic


TurtleBot


Aldebaran Nao


Lego NXT


Shadow Hand


Willow Garage PR2


iRobot Roomba

Robotnik Guardian


Merlin miabotPro


AscTec Quadrotor


CoroWare Corobot


Clearpath Robotics Husky


Clearpath Robotics Kingfisher


Festo Didactic Robotino

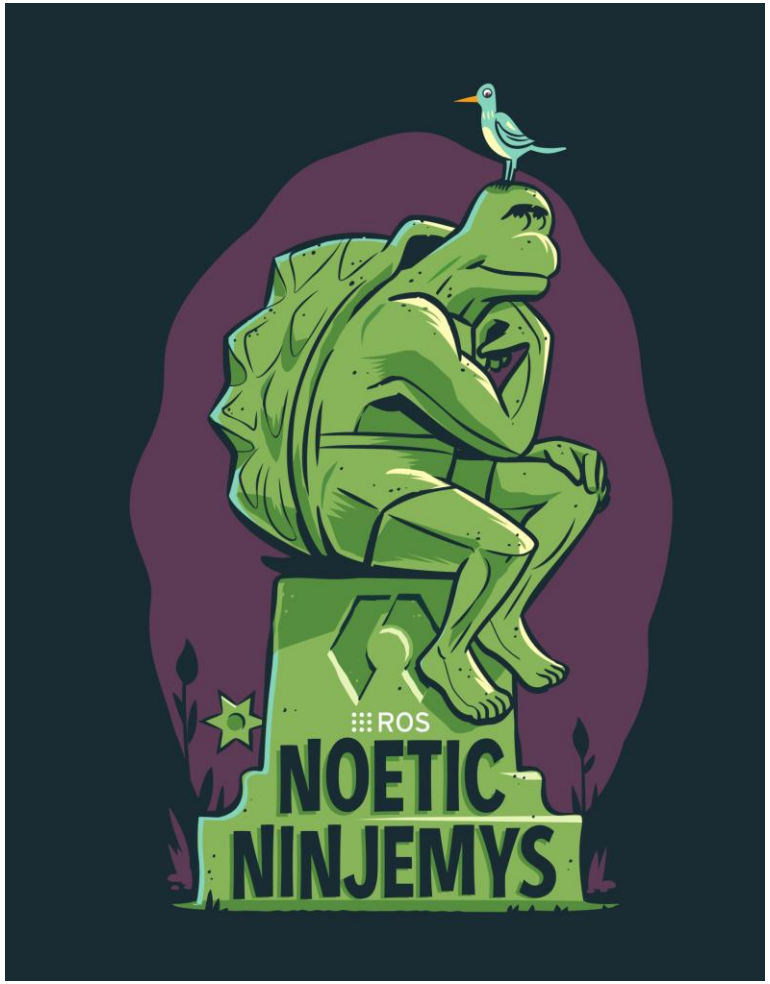
Integration with external libraries

- ROS provides seamless integration of external libraries and popular open-source projects

ROS Version

- ROS is currently supported only on **Ubuntu and Debian**
  - other variants such as Windows,Mac OS X, and Android are considered experimental
- Current ROS Noetic runs on **Ubuntu 20.04**

http://wiki.ros.org/noetic/Installation



ROS ENVIRONMENT

- ROS is fully integrated in the Linux environment: the rosbash package contains useful bash functions and adds tab completion to a large number of ROS utilities
- After installing, ROS, setup.*sh files in '/opt/ ros /<distro>/', need to be sourced to start rosbash

```
marcobernardi@marcobernardi-Precision-3550:~$ source /opt/ros/noetic/setup.bash
marcobernardi@marcobernardi-Precision-3550:~$
```

- After compiling ROS nodes, setup.*sh files in '/devel/, need to be sourced

```
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace$ source devel/setup.bash
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace$
```

- This command needs to be run on every new shell to have access to the ros commands.

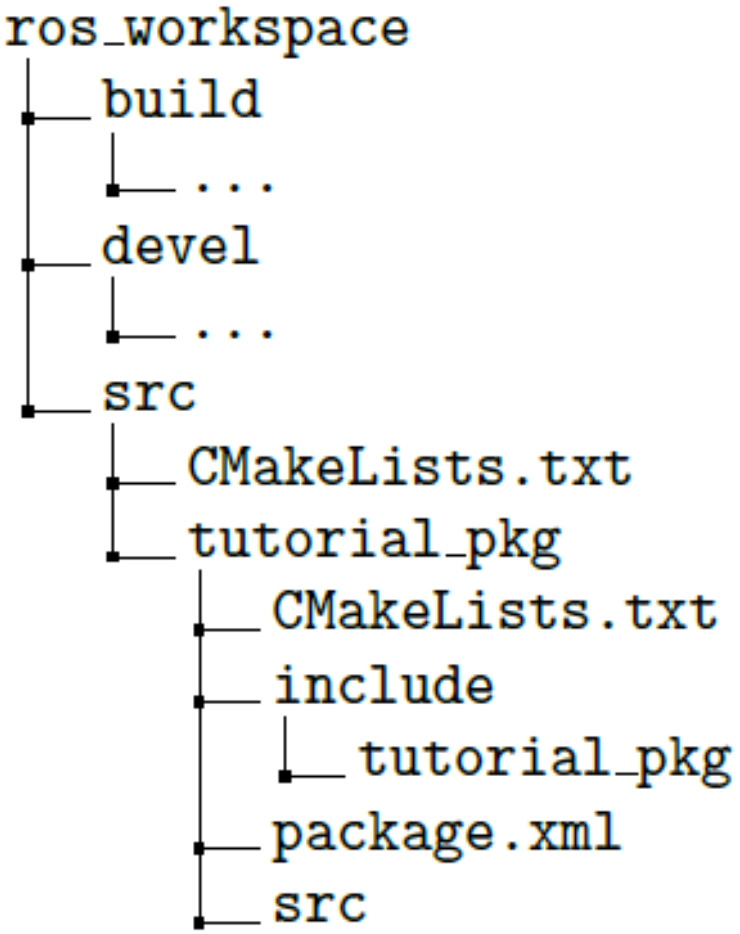
ROS Core Concepts

- Packages
- Nodes
- ROS Masters
- Messages and Topics
- Services
- Actions

Packages

- The ROS packages are the most basic unit of the ROS software.
  - It contains the ROS runtime process (nodes), libraries, configuration files, and so on, which are organized together as a single unit.
  - Packages are the atomic build item and release item in the ROS software.
- A ROS package is a directory inside a catkin workspace that has a package.xml file in it
- A catkin workspace is a set of directories in which a set of related ROS code/packages live (catkin is the **ROS build system**: CMake + Python)
- It is possible to have multiple workspaces, but work can performed on only one at a time

Structure of a workspace



Catkin workspace folders

- Source space: workspace_folder/src
  - Contains the source code of the packages. Each folder withing the source space contains one or more packages
- Build space: workspace_folder/build
  - Where catking invoce the cmake to build the packages in source space. Cmake and catking keep their cache information and other intermediate files here.
- Devel space: workspace_folder/devel
  - Where the build targets are placed before being installed
- Install space: workspace_folder/install
  - Once the targets are build, they can be installed into the install space by invoking the install targets

Layout of a package

- Source files implement nodes, can be written in multiple languages
- Nodes are launched individually or in groups, using launch files

Directory	Explanation
include/	C++ include headers
src/	Source files
msg/	Folder containing Message (msg) types
srv/	Folder containing Service (srv) types
launch/	Folder containing launch files
package.xml	The package manifest
CMakeLists.txt	CMake build file

Create a new package

- **catkin_create_pkg:** Tool for creating a new package

```
catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

```
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace/src$ catkin_create_pkg my_first_package std_msgs rospy roscpp
Created file my_first_package/package.xml
Created file my_first_package/CMakeLists.txt
Created folder my_first_package/include/my_first_package
Created folder my_first_package/src
Successfully created files in /home/marcobernardi/ROS_TUTORIAL/ros_workspace/src/my_first_package. Please adjust the values in package.xml.
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace/src$
```

Dependencies of a package

- **rospack**: ROS package management tool

```
rospack depends1 <package_name>
```

```
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace$ rospack depends1 my_first_package
roscpp
rospy
std_msgs
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace$
```

Dependencies of a package (another way)

- **Package.xml:** defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages

```
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

Indirect dependencies

- A depencies can have its own depencies

```
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace/src/my_first_package$ rospack depends1 roscpp
cpp_common
message_runtime
rosconsole
roscpp_serialization
roscpp_traits
rosgraph_msgs
rostime
std_msgs
xmlrpcpp
```

- Check all dependencies of a package

rospack depends <package_name>

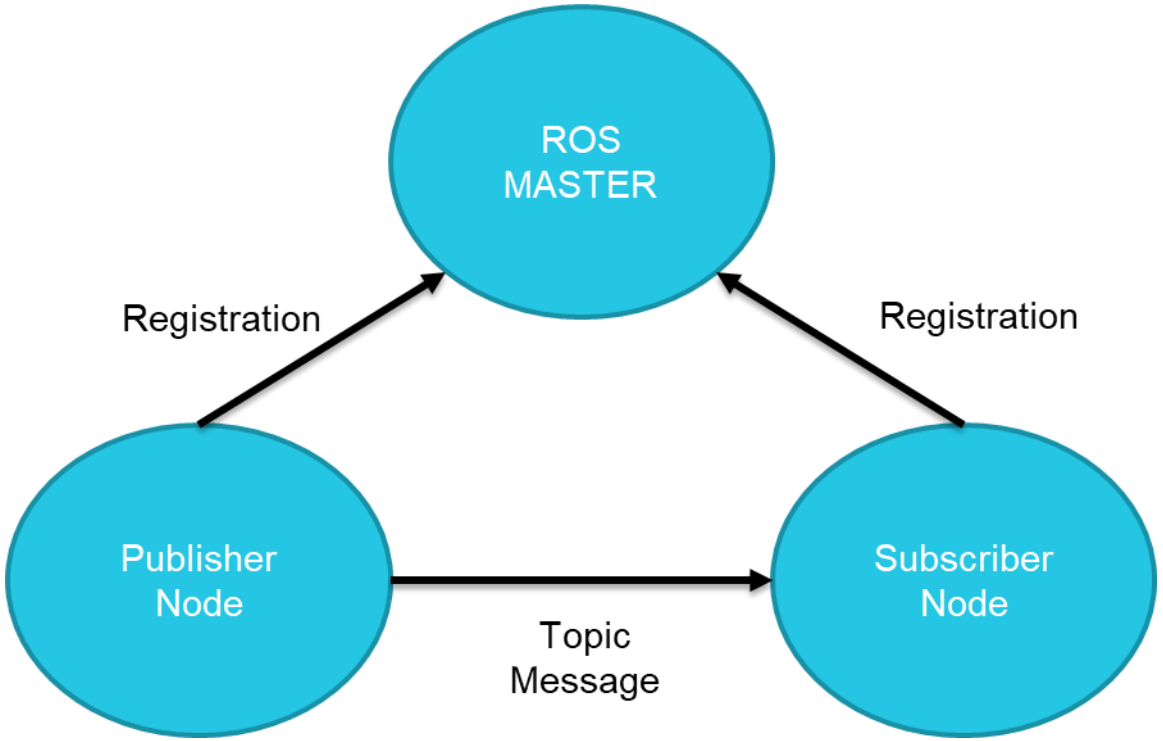
Compiling a package

- Using catkin_make tool
- Build folder: configure and build your packages.
- Devel folder: executables and libraries

```
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace$ catkin_make
Base path: /home/marcobernardi/ROS_TUTORIAL/ros_workspace
Source space: /home/marcobernardi/ROS_TUTORIAL/ros_workspace/src
Build space: /home/marcobernardi/ROS_TUTORIAL/ros_workspace/build
Devel space: /home/marcobernardi/ROS_TUTORIAL/ros_workspace/devel
Install space: /home/marcobernardi/ROS_TUTORIAL/ros_workspace/install
####
#### Running command: "make cmake_check_build_system" in "/home/marcobernardi/ROS_TUTORIAL/ros_workspace/build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/marcobernardi/ROS_TUTORIAL/ros_workspace/devel
-- Using CMAKE_PREFIX_PATH: /home/marcobernardi/ROS_TUTORIAL/ros_workspace/devel;/opt/ros/noetic
-- This workspace overlays: /home/marcobernardi/ROS_TUTORIAL/ros_workspace/devel;/opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.5", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Using empy: /usr/lib/python3/dist-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/marcobernardi/ROS_TUTORIAL/ros_workspace/build/test_results
-- Forcing gtest/gmock from source, though one was otherwise available.
-- Found gtest sources under '/usr/src/googletest': gtests will be built
-- Found gmock sources under '/usr/src/googletest': gmock will be built
-- Found PythonInterp: /usr/bin/python3 (found version "3.8.5")
-- Using Python nosetests: /usr/bin/nosetests3
-- catkin 0.8.9
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - my_first_package
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'my_first_package'
-- ==> add_subdirectory(my_first_package)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/marcobernardi/ROS_TUTORIAL/ros_workspace/build
####
#### Running command: "make -j8 -l8" in "/home/marcobernardi/ROS_TUTORIAL/ros_workspace/build"
####
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace$
```

Nodes

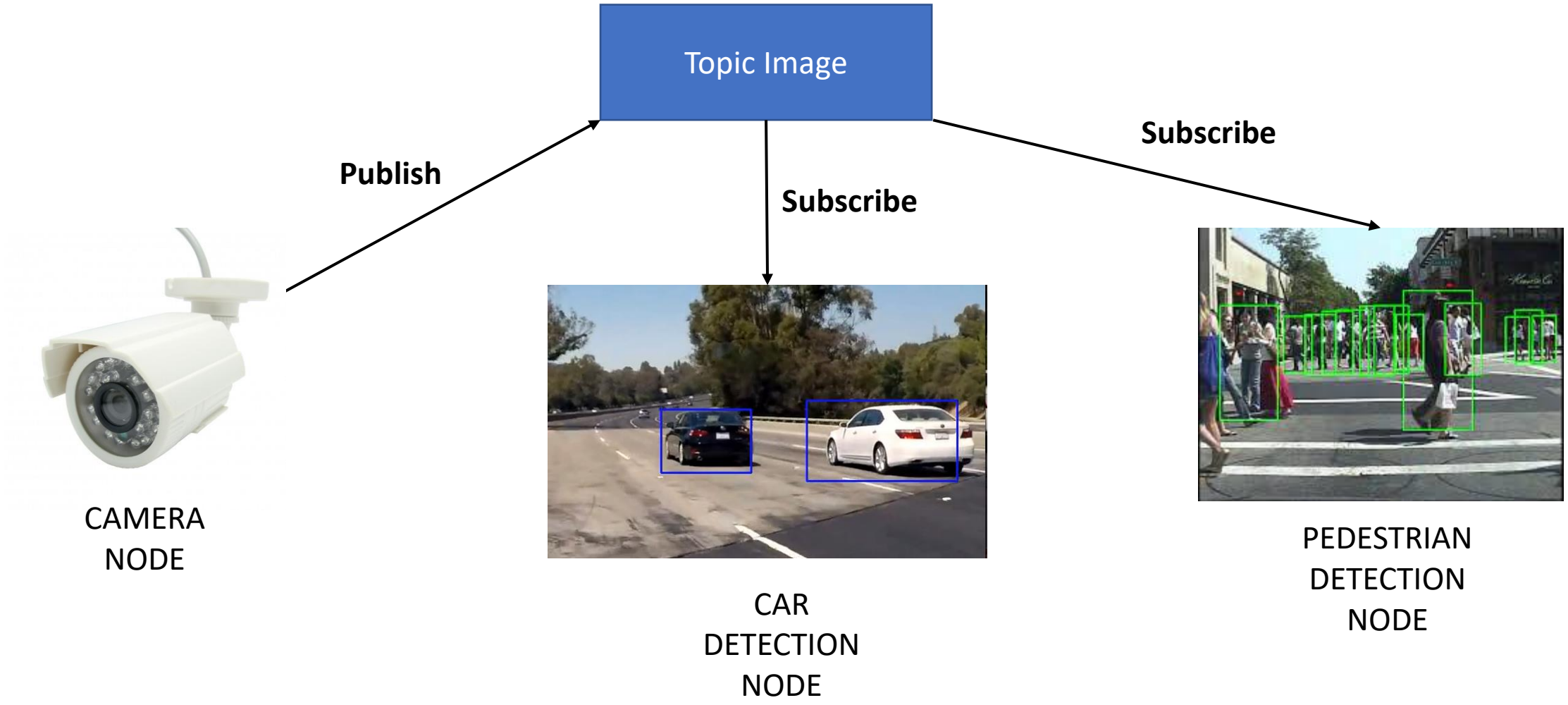
- Single purposed executable programs (e.g. sensor driver(s), actuator driver(s), map building, planner, UI, etc.)



Nodes 2

- Nodes are written using a ROS client library
  - roscpp C++ client library
  - rospy python client library
- Individually compiled, executed, and managed
- Nodes can **publish** or **subscribe** to a Topic
- Nodes can also provide or use a **Service** or an **Action**

Nodes: a practical example



ROS Master

- The ROS master provides naming and registration services to enable the nodes to locate each other and, therefore, to communicate
- Every node registers at startup with the master

roscore

```
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL$ roscore
... logging to /home/marcobernardi/.ros/log/c2c7ce2e-a4dc-11eb-b3c0-712de123794b/roslaunch-marcobernardi-Precision-3550-10794.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://marcobernardi-Precision-3550:35091/
ros_comm version 1.15.9


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.9

NODES

auto-starting new master
process[master]: started with pid [10802]
ROS_MASTER_URI=http://marcobernardi-Precision-3550:11311/

setting /run_id to c2c7ce2e-a4dc-11eb-b3c0-712de123794b
process[rosout-1]: started with pid [10812]
started core service [/rosout]
```

Information about nodes

- Listing running nodes

rosnode list

```
marcobernardi@marcobernardi-Precision-3550:~$ marcobernardi@marcobernardi-Precision-3550:~$ rosnode list
/rosout
marcobernardi@marcobernardi-Precision-3550:~$
```

- Extracting nodes information

rosnode info /rosout

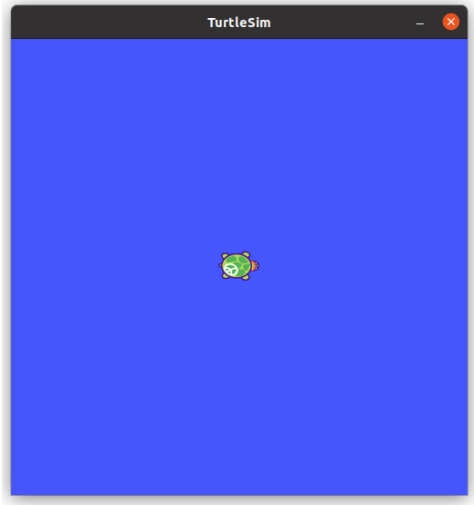
```
marcobernardi@marcobernardi-Precision-3550:~$ rosnode info /rosout
--------------------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/get_loggers
 * /rosout/set_logger_level


contacting node http://marcobernardi-Precision-3550:38295/ ...
Pid: 10812
```

Running a node: rosrun

- use the package name to directly execute a node within the package (without having to know the package path)

```
rosrun [package_name] [node_name]
```

```
marcobernardi@marcobernardi-Precision-3550:~/catkin_ws/src$ rosrun turtlesim turtlesim_node
[ INFO] [1619256489.094648373]: Starting turtlesim with node name /turtlesim
[ INFO] [1619256489.098432194]: Spawning turtle [turtle1] at x=[5,544445], y=[5,544445], theta=[0,000000]
```



Running a node: rosrun 2

- Check the nodes list

```
marcobernardi@marcobernardi-Precision-3550:~$ rosnode list
/rosout
/turtlesim
```

- Remapping argument: change the node name

```
rosrun turtlesim turtlesim_node __name:=mia_tartaruga
```

```
marcobernardi@marcobernardi-Precision-3550:~$ rosnode list
/mia_tartaruga
/rosout
```

The node is up?

- To verify if the node is running use **rosnode ping**

```
rosnode ping [node_name]
```

```
marcobernardi@marcobernardi-Precision-3550:~$ rosnode ping mia_tartaruga
rosnode: node is [/mia_tartaruga]
pinging /mia_tartaruga with a timeout of 3.0s
xmlrpc reply from http://marcobernardi-Precision-3550:40695/     time=0.401735ms
xmlrpc reply from http://marcobernardi-Precision-3550:40695/     time=2.027035ms
xmlrpc reply from http://marcobernardi-Precision-3550:40695/     time=2.059460ms
xmlrpc reply from http://marcobernardi-Precision-3550:40695/     time=1.928329ms
xmlrpc reply from http://marcobernardi-Precision-3550:40695/     time=2.049208ms
```

Kill a node

rosnode kill [node_name]

```
marcobernardi@marcobernardi-Precision-3550:~/ROS_TUTORIAL/ros_workspace$ rosnode kill /mia_tartaruga
killing /mia_tartaruga
killed
```

Topic and messages

- Communication in ROS exploits messages
- Messages are organized in topics
- A node that wants to share information will **publish** messages on a topic(s)
- A node that wants to receive information will **subscribe** to the topic(s)
- ROS master takes care of ensuring that publishers and subscribers can find each other

List of the current topic

- Returns a list of all topics currently subscribed and published.

```
marcobernardi@marcobernardi-Precision-3550:~$ rostopic list -v

Published topics:
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /rosout [rosgraph_msgs/Log] 1 publisher
 * /turtle1/pose [turtlesim/Pose] 1 publisher
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher

Subscribed topics:
 * /rosout [rosgraph_msgs/Log] 1 subscriber
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber
```

Type of a topic

- The publisher and subscriber must send and receive the same "message type".
- A topic type is defined by the message type posted on it.

rostopic type [topic]

rosmsg show [topic]

```
marcobernardi@marcobernardi-Precision-3550:~$ rostopic type /turtle1/pose
turtlesim/Pose
marcobernardi@marcobernardi-Precision-3550:~$ rosmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

Type of a topic 2

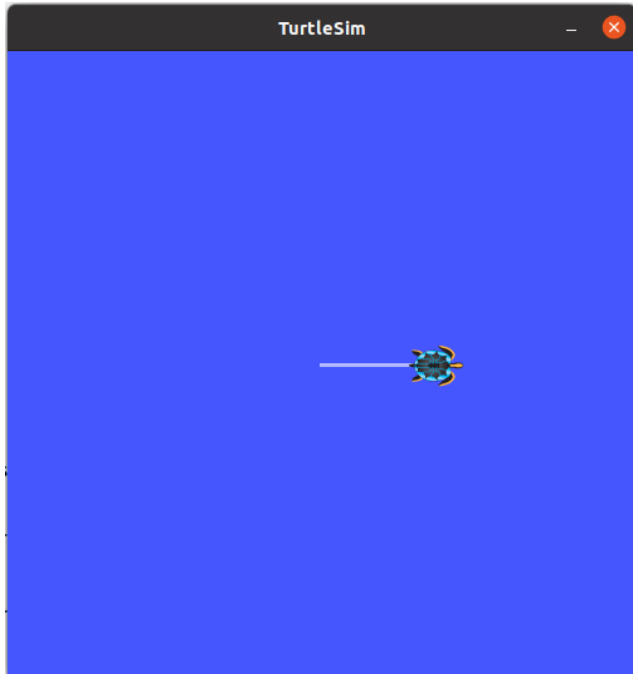
- The message type can consist of more complex structures

```
marcobernardi@marcobernardi-Precision-3550:~$ rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Publish on a topic

rostopic pub [topic] [topic_type] [val1] … [valN]

```
marcobernardi@marcobernardi-Precision-3550:~/catkin_ws/src$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0,0.0,0.0]' '[1.8,0.0,0.0]'
publishing and latching message for 3.0 seconds
```



Print topic messages

- Print messages published to a topic

```
rostopic echo [topic]
```

```
marcobernardi@marcobernardi-Precision-3550:~/catkin_ws/src$ rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 1.8
  y: 0.0
  z: 0.0
---
```

Creating a messages

- Messages in ROS are .msg files stored in the corresponding package folder, within the msg dir
- Supported field types:
  - int8, int16, int32, int64 (plus uint*)
  - float32, float64
  - String
  - time, duration
  - othermsg files
  - variable length array [] and fixed length array [C]
  - Header: timestamp and coordinate frame information

Creating a messages 2

- Make sure that msg files are turned into source code for C++, Python, and other languages
- Package.xml

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

- CmakeLists.xml

```
find_package(catkin REQUIRED
COMPONENTS
roscpp
rospy
std_msgs
message_generation
)
```

```
catkin_package(
...
CATKIN_DEPENDS
message_runtime ...
...)
```

```
add_message_files(
FILES
Num.msg
)
```

Services

- Services are another way that nodes can communicate with each other.
- It allows nodes to send a request and receive a response.

rosservice list

```
marcobernardi@marcobernardi-Precision-3550:~$ rosservice list
/clear
/kill
/mia_tartaruga/get_loggers
/mia_tartaruga/set_logger_level
/reset
/rosout/get_loggers
/rosout/set_logger_level
/rostopic_14939_1619260518680/get_loggers
/rostopic_14939_1619260518680/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
```

Type of a service

- As for the topic, the service are defined by a type.

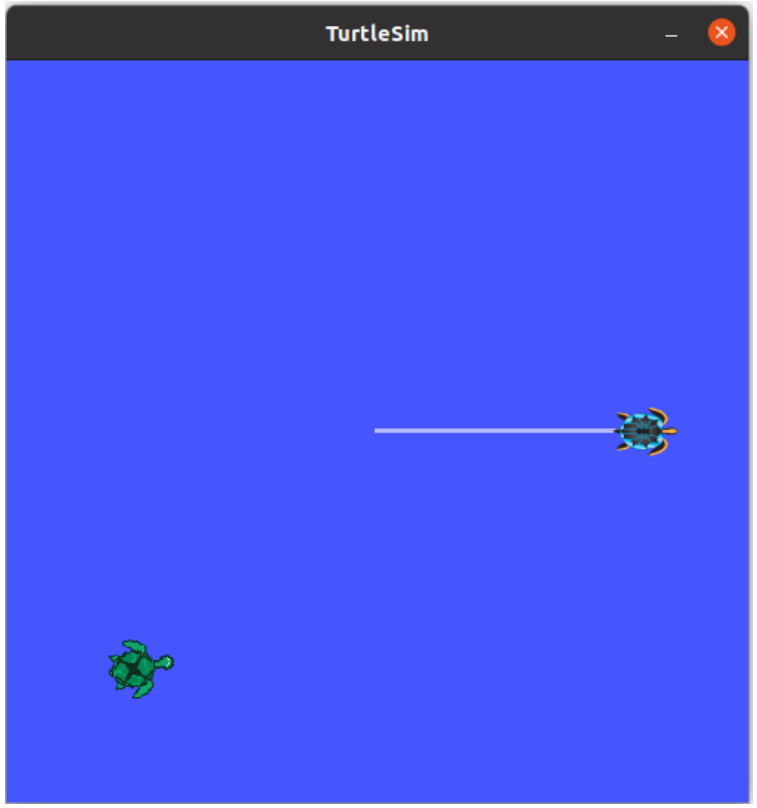
rosservice type spawn

```
marcobernardi@marcobernardi-Precision-3550:~$  rosservice type spawn| rossrv show
float32 x
float32 y
float32 theta
string name
---
string name
```

Call a service

rosservice call [service] [args]

```
marcobernardi@marcobernardi-Precision-3550:~/catkin_ws/src$ rosservice call /spawn 2 2 0.2 ""
name: "turtle2"
```



Creating a service

- service files (srv) are just like msg files, except they contain two parts: a request and a response. The two parts are separated by a '---' line.
- A and B are the request, and Sum is the response.

```
int64 A
int64 B
---
int64 Sum
```

Sum.srv

Creating a service 2

- Make sure that srv files are turned into source code for C++, Python, and other languages
- Package.xml

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

- CmakeLists.xml

```
find_package(catkin REQUIRED
COMPONENTS
roscpp
rospy
std_msgs
message_generation
)
```

```
add_service_files(
FILES
Sum.msg
)
```

Anatomy of ROS NODE

```
ros::Publisher pub;

// function called whenever a message is received
void my_callback(MsgType* m) {
    OtherMessageType m2;
    … // do something with m and valorize m2
    pub.publish(m2);
}

int main(int argc, char** argv){
    // initializes the ros ecosystem
    ros::init(argc, argv);

    // object to access the namespace facilities
    ros::NodeHandle n;

    // tell the world that you will provide a topic named "published_topic"
    pub.advertise<OtherMessageType>("published_topic");

    // tell the world that you will provide a topic named "published_topic"
    Subscriber s =n.subscribe<MessageType*>("my_topic",my_callback);
    ros::spin();
}
```

Additional Resources

- Site: http://www.ros.org/
- Blog: http://www.ros.org/news/
- Documentation: http://wiki.ros.org/