

# Introduction to ROS



RE510

2024. 03. 18. ~ 20234. 03. 29.

MORIN lab

Professor : Jinwhan Kim

TA : Donghyun Kim

# Table of Contents

---

1. What is ROS?
2. ROS core concepts
3. ROS basic practice
  1. ROS installation
  2. Publisher/subscriber node with custom message(C++, python)
  3. Launch file
4. Assignment #1

## Appendix

1. Ubuntu basic command
2. ROS2

# What is ROS?



# What is ROS?

## ROS(Robot operating system)

**ROS** is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

The ROS logo consists of a 3x3 grid of dots to the left of the letters "ROS" in a bold, sans-serif font.The open robotics logo features a stylized blue icon of a robot head or a keyhole shape to the left of the words "open" and "robotics" stacked vertically in a sans-serif font.

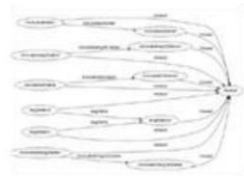
<https://wiki.ros.org/>

# What is ROS?

ROS(Robot operating system)



=



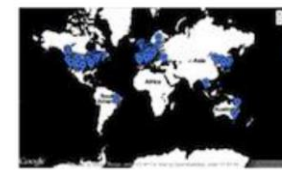
+



+



+



## Plumbing

- Process management
- Inter-process communication
- Device drivers

## Tools

- Simulator
- Visualization
- Graphical user interface
- Data logging

## Capabilities

- Control
- Planning
- Perception
- Mapping
- Manipulation

## Ecosystem

- Package organization
- Software distribution
- Documentation
- Tutorials

# What is ROS?

## ROS versions

Distro	Release date	Poster	Tuturtle, turtle in tutorial	EOL date
ROS Noetic Ninjemys (Recommended)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014

<http://wiki.ros.org/Distributions>

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
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019
Bouncy Bolson	July 2nd, 2018		July 2019
Ardent Apalone	December 8th, 2017		December 2018

<https://docs.ros.org/en/foxy/Releases.html>

# Environment setup for ROS

Development environment for ROS	
Operating system	Ubuntu 20.04
ROS version	ROS Noetic Ninjemys
Computer architecture	amd64
IDE	Visual studio Code
Programming language	Python 3, C++14(or latest)
Simulator	Gazebo 11.x

# ROS core concepts



# ROS core concepts

## ROS terms

- **Node**

The smallest unit of executable processors. It can be regarded as single executable program. In ROS, a system is consist of many nodes. Each node transmits and receives data by message communication.

- **Package**

One or more nodes, information for node execution, etc. Also, bundles of packages are called as metapackages.

- **Message**

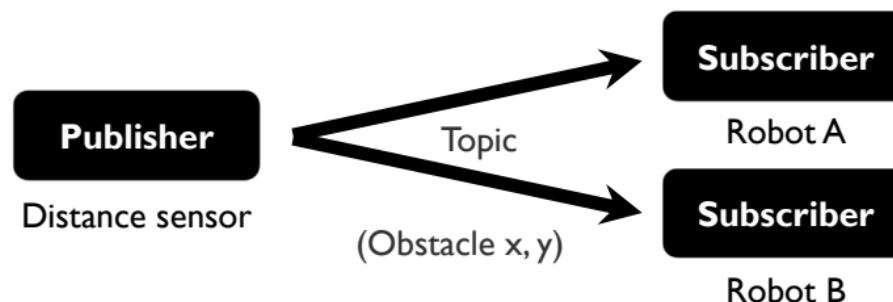
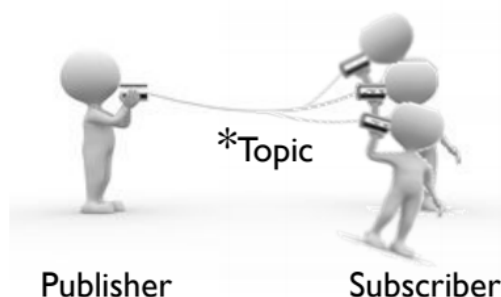
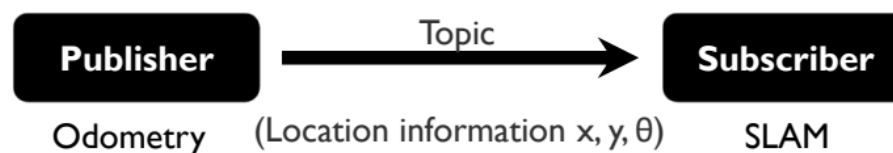
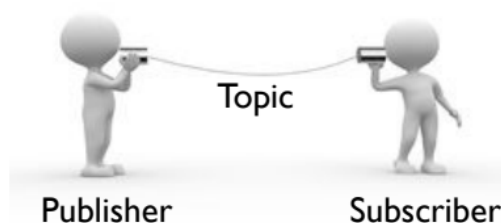
Data is transmitted and received through message between nodes. Messages can have various types such as integer, floating point, and Boolean.

- **Topic**

Topic is named stream of messages with a defined type. Nodes communicate with each other by publishing messages to topics.

# ROS core concepts

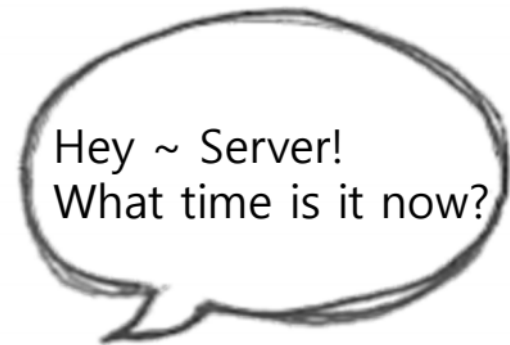
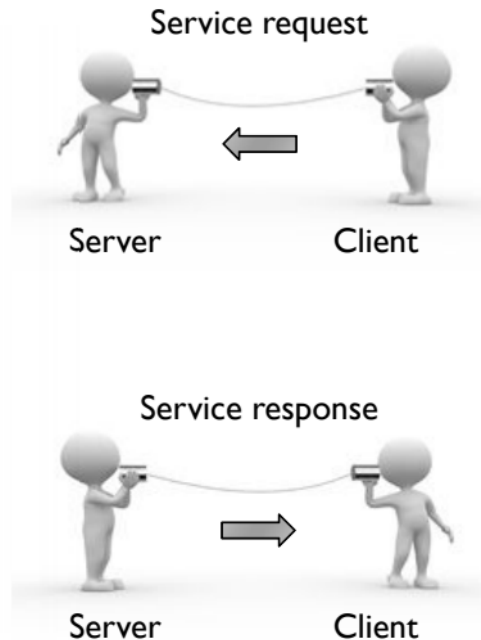
## Topic, publisher, subscriber



\* 1:1 Publisher and Subscriber communication is also possible for Topic, and 1:N, N:1, N:N communication is also possible depending on the purpose.

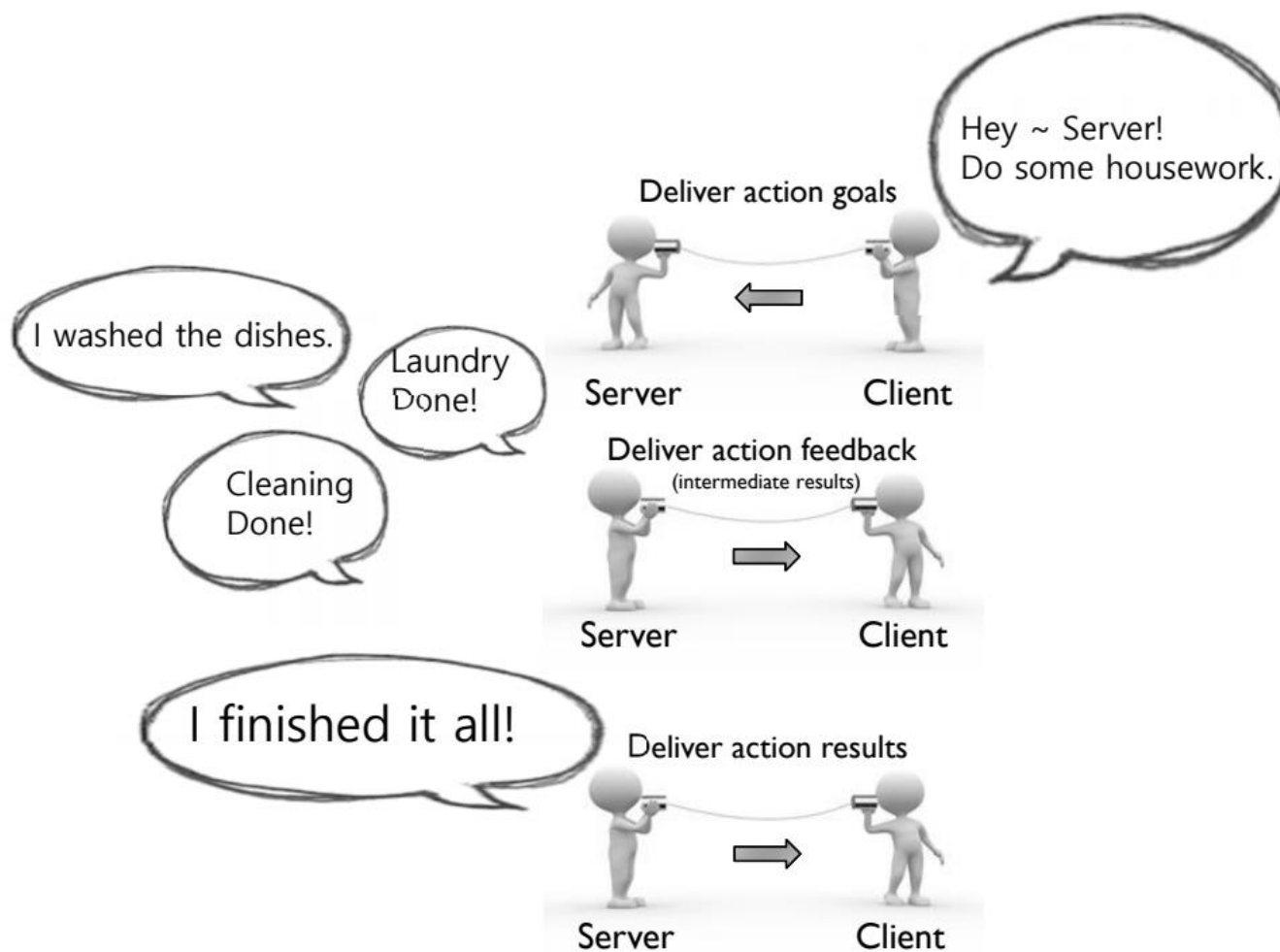
# ROS core concepts

## Service, Service server, Service client



# ROS core concepts

## Action, action server, action client





# ROS core concepts with ROS 1

## ROS Master

- Manages the communication between nodes (processes)
- Every node registers at startup with the master

ROS Master

Start a master with

```
> roscore
```

# ROS core concepts with ROS 1

## ROS Nodes

- Single purpose, executable program
- Individually compiled, executed, and managed
- Organized in packages

Run a node with

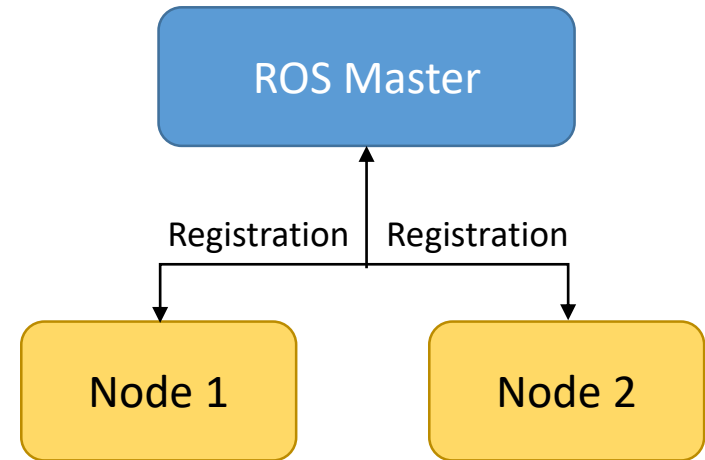
```
> rosrun package_name node_name
```

See active nodes with

```
> rosnode list
```

Retrieve information about a node with

```
> rosnode info node_name
```



# ROS core concepts with ROS 1

## ROS Topics

- Nodes communicate over topics
  - Nodes can publish or subscribe to a topic
  - Typically, 1 publisher and n subscriber
- Topic is a name for a stream of messages

List active topics with

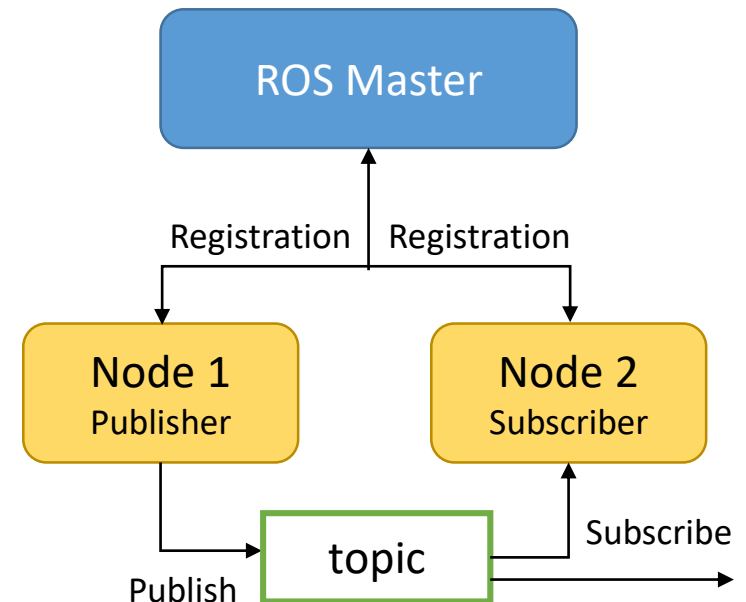
```
> rostopic list
```

Subscribe and print the contents of a topic with

```
> rostopic echo /topic
```

Show information about a topic with

```
> rostopic info /topic
```



# ROS core concepts with ROS 1

## ROS Messages

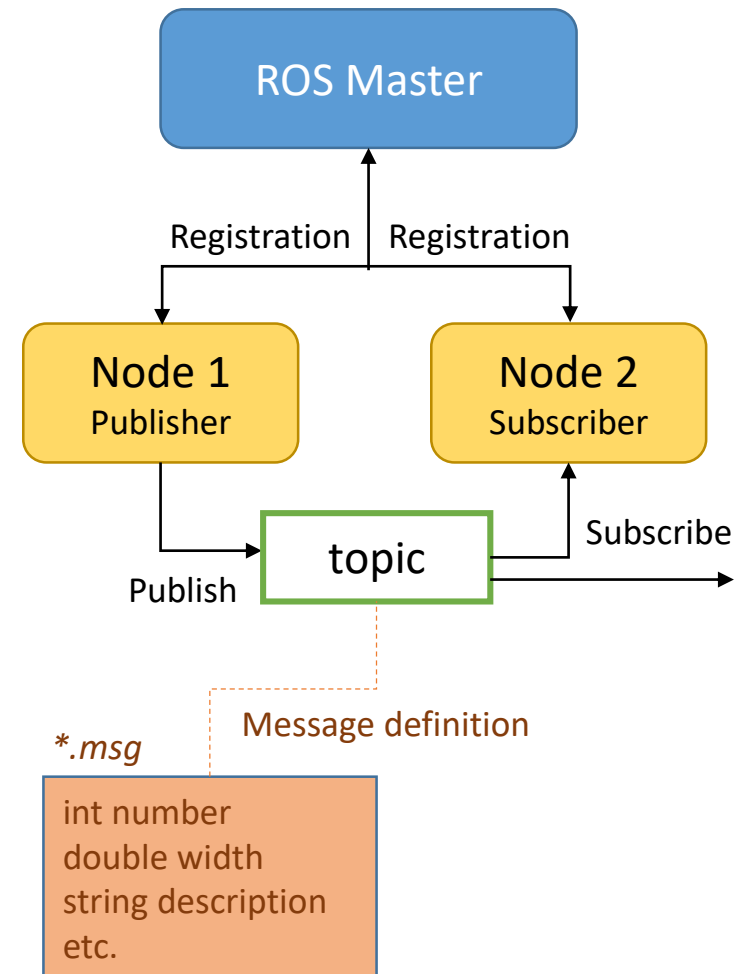
- Data structure defining the type of a topic
- Comprised of a nested structure of integers, floats, Booleans, strings, etc. and arrays of objects
- Defined in *\*.msg* file

See the type of a topic

```
> rostopic type /topic
```

Publish a message to a topic

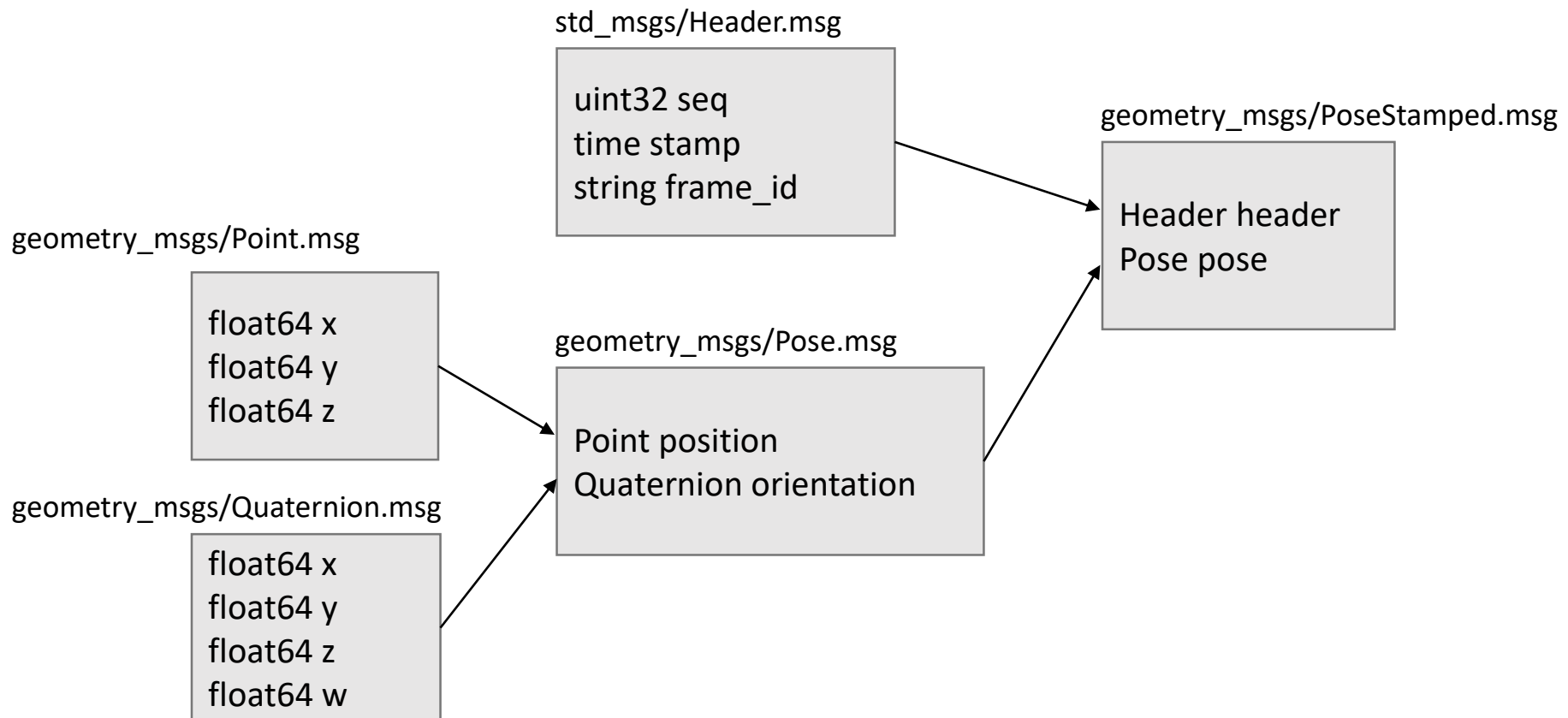
```
> rostopic pub /topic type data
```





# ROS core concepts with ROS 1

## ROS Messages example



# ROS1 basic practice

# ROS basic practice

## ROS 1 installation

<http://wiki.ros.org/noetic/Installation/Ubuntu>

### 1. Installation

#### 1.1 Configure your Ubuntu repositories

Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse." You can [follow the Ubuntu guide](#) for instructions on doing this.

#### 1.2 Setup your sources.list

Setup your computer to accept software from packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

**Mirrors** [Source Debs](#) are also available

#### 1.3 Set up your keys

```
sudo apt install curl # if you haven't already installed curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

#### 1.4 Installation

First, make sure your Debian package index is up-to-date:

```
sudo apt update
```

Now pick how much of ROS you would like to install.

**Desktop-Full Install: (Recommended)** : Everything in **Desktop** plus 2D/3D simulators and 2D/3D perception packages

```
sudo apt install ros-noetic-desktop-full
```

or [click here](#)

**Desktop Install:** Everything in **ROS-Base** plus tools like **rqt** and **rviz**

```
sudo apt install ros-noetic-desktop
```

or [click here](#)

**ROS-Base: (Bare Bones)** ROS packaging, build, and communication libraries. No GUI tools.

```
sudo apt install ros-noetic-ros-base
```

# ROS 1 basic practice

## ROS1 setup

```
> code ~/.bashrc
```

- Add the following commands

```
export ROS_MASTER_URI=http://localhost:11311  
export ROS_IP=localhost  
source /opt/ros/noetic/setup.bash
```



# ROS 1 basic practice

## ROS 1 turtlesim example

```
> roscore
```

```
> rosrun turtlesim turtlesim_node
```

```
> rosrun turtlesim turtle_teleop_key
```

```
> rqt_graph
```

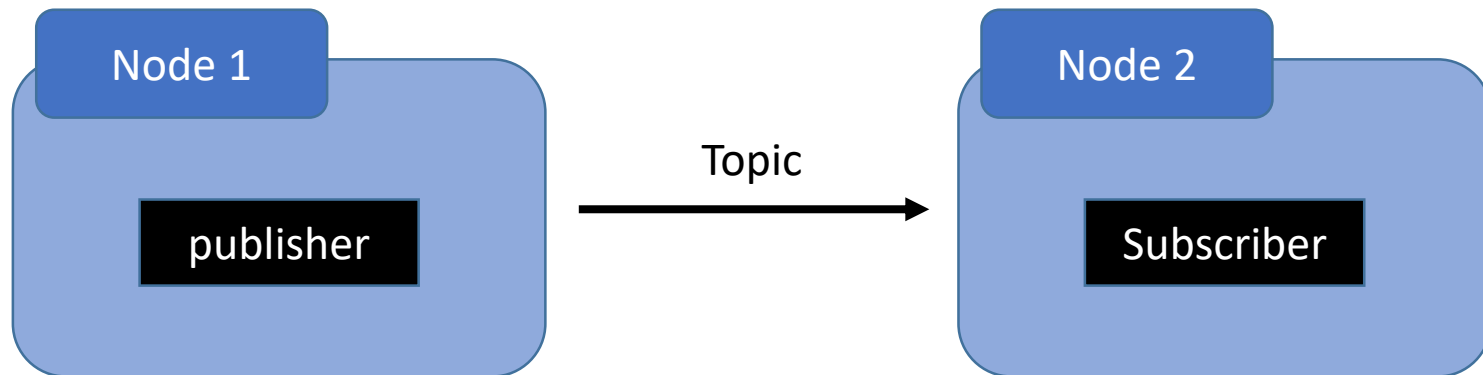
```
> rostopic list
```

```
> rostopic info /turtle1/cmd_vel
```

```
> rostopic echo /turtle1/cmd_vel
```

# ROS 1 basic practice

## Publisher/Subscriber node programming practice



# ROS 1 basic practice

## Setup catkin workspace build system

- Setup catkin\_ws

```
> mkdir -p ~/catkin_ws/src
> cd ~/catkin_ws
> catkin_make
> source devel/setup.bash
```

- Setup alias for convenience

```
> code ~
```

find *.bashrc* and put the below codes

```
source ~/catkin_ws/devel/setup.bash
alias cm='cd ~/catkin_ws && catkin_make'
alias cs='cd~/catkin_ws/src'
alias eb='code ~/.bashrc'
alias sb='source ~/.bashrc'
```

# ROS 1 basic practice

Create a new package for practice(C++)

<http://wiki.ros.org/ROS/Tutorials>

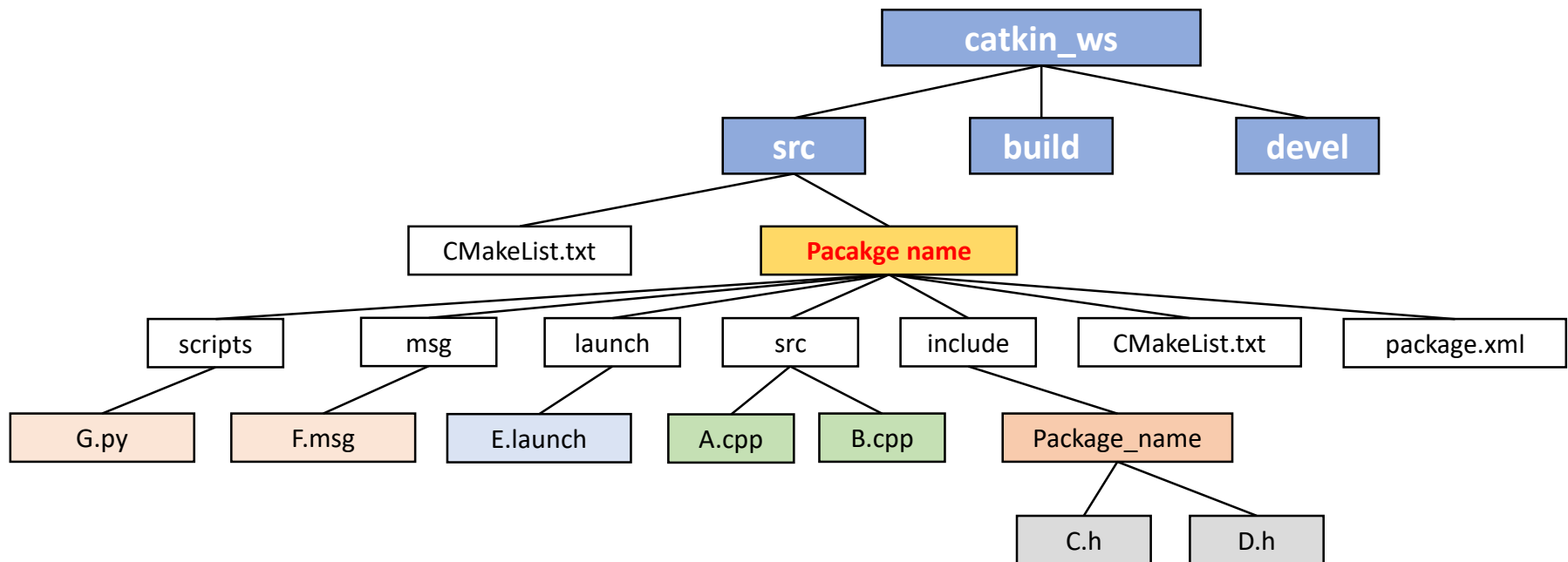
```
> cd ~/catkin_ws/src  
> catkin_create_pkg ros_tutorials roscpp std_msgs  
message_generation
```

```
> cd ros_tutorials  
> ls  
include → header file folder  
src → source code folder  
CMakeLists.txt → build configuration file  
package.xml → package configuration file
```



# ROS 1 basic practice

## ROS package components



# ROS 1 basic practice

## package.xml

```
src > ros_tutorials > package.xml
1  <?xml version="1.0"?>
2  <package format="2">
3    <name>ros_tutorials</name>
4    <version>0.0.0</version>
5    <description>The ros_tutorials package</description>
6
7    <maintainer email="your_email@todo.todo">your_name</maintainer>
8
9    <license>BSD</license>
10
11    <buildtool_depend>catkin</buildtool_depend>
12
13    <build_depend>message_generation</build_depend>
14    <build_depend>roscpp</build_depend>
15    <build_depend>std_msgs</build_depend>
16
17    <build_export_depend>roscpp</build_export_depend>
18    <build_export_depend>std_msgs</build_export_depend>
19
20    <exec_depend>roscpp</exec_depend>
21    <exec_depend>std_msgs</exec_depend>
22    <exec_depend>message_runtime</exec_depend>
23
24    <export>
25
26    </export>
27  </package>
```

# ROS 1 basic practice

## CMakeLists.txt(C++)

```
src > ros_tutorials > M CMakeLists.txt
1  cmake_minimum_required(VERSION 3.0.2)
2  project(ros_tutorials)
3
4  find_package(catkin REQUIRED COMPONENTS
5    message_generation
6    roscpp
7    std_msgs
8  )
9
10 # Generate messages in the 'msg' folder
11 add_message_files(
12   FILES
13   Counts.msg
14 )
15
16 # Generate added messages and services with any dependencies listed here
17 generate_messages(
18   DEPENDENCIES
19   std_msgs
20 )
21
22 catkin_package(
23   LIBRARIES ros_tutorials
24   CATKIN_DEPENDS message_runtime roscpp std_msgs
25 )
26
27 include_directories(
28   # include
29   ${catkin_INCLUDE_DIRS}
30 )
31
32 add_executable(talker src/talker.cpp)
33 target_link_libraries(talker ${catkin_LIBRARIES})
34
35 add_executable(listener src/listener.cpp)
36 target_link_libraries(listener ${catkin_LIBRARIES})
```

# ROS 1 basic practice

## Create message file

- Following this instruction

<http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv>

- create msg file

```
> cd ~/catkin_ws/src/ros_tutorials
> mkdir msg
> cd msg
> code Counts.msg
```

```
src > ros_tutorials > msg > ≡ Counts.msg
1   time stamp
2   int32 count
3   int32 square
```

# ROS 1 basic practice

## Creating the publisher node

- Following this instruction

<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>

- create cpp file

```
> cd ~/catkin_ws/src/ros_tutorials /src
> code talker.cpp
```

- vscode C/C++ configuration setup

```
ctrl + shift + p
> C/C++: Edit Configurations (JSON)
```

```
.vscode > {} c_cpp_properties.json > ...
1  {
2      "configurations": [
3          {
4              "name": "Linux",
5              "includePath": [
6                  "${workspaceFolder}/**",
7                  "~/catkin_ws/devel/include",
8                  "/opt/ros/noetic/include",
9                  "/usr/include/**"
10             ],
11             "defines": [],
12             "compilerPath": "/usr/bin/gcc",
13             "cStandard": "c17",
14             "cppStandard": "gnu++14",
15             "intelliSenseMode": "linux-gcc-x64"
16         }
17     ],
18     "version": 4
19 }
```

# ROS 1 basic practice

## Creating the publisher node

```
src > ros_tutorials > src >  talker.cpp > ...  
1  #include "ros/ros.h"  
2  #include "ros_tutorials/Counts.h"  msg include  
3  #include <cmath>  
4  
5  int main(int argc, char **argv)  
6  {  
7      ros::init(argc, argv, "talker"); node name  
8      ros::NodeHandle nh;  
9  
10     ros::Publisher pub = nh.advertise<ros_tutorials::Counts>("chatter", 100); declare publisher  
11     ros::Rate loop_rate(1); 1Hz  
12  
13     ros_tutorials::Counts msg; declare msg  
14     int count = 0;  
15  
16     while (ros::ok())  
17     {  
18         msg.stamp = ros::Time::now();  
19         msg.count = count;  
20         msg.square = int(pow(count, 2));  
21  
22         ROS_INFO("Send msg");  
23         ROS_INFO("time : %d", msg.stamp.sec);  
24         ROS_INFO("count : %d", msg.count);  
25         ROS_INFO("count square : %d", msg.square);  
26  
27         pub.publish(msg); publish  
28         loop_rate.sleep(); loop rate 1Hz  
29         count++;  
30     }  
31  
32     return 0;  
33 }
```



# ROS 1 basic practice

## Creating the subscriber node

- Put the following options in the CMakeLists.txt

```
add_executable(listener src/listener.cpp)
```

- Create cpp file

```
> cd ~/catkin_ws/src/ros_tutorials/src
> code listener.cpp
```

# ROS 1 basic practice

## Creating the subscriber node

```
src > ros_tutorials > src > listener.cpp > ...
1  #include "ros/ros.h"
2  #include "ros_tutorials/Counts.h" msg include
3  #include <cmath>
4
5  void msgCallback(const ros_tutorials::Counts::ConstPtr& msg)
6  {
7      ROS_INFO("Recieve msg");
8      ROS_INFO("time : %d", msg->stamp.sec);
9      ROS_INFO("count : %d", msg->count);
10     ROS_INFO("count square : %d", msg->square);
11 }
12
13 int main(int argc, char **argv)
14 {
15     ros::init(argc, argv, "listener"); node name
16     ros::NodeHandle nh;
17
18     ros::Subscriber sub = nh.subscribe("chatter", 100, msgCallback);
19                                     declare subscriber
20     ros::spin(); Process callback functions requested in queue
21     return 0;
22 }
```

# ROS 1 basic practice

## Build the nodes

- Build the `cpp_example_pkg` package with the following command

```
> cd ~/catkin_ws  
> catkin_make
```

- or simply use alias

```
> cm
```

```
Scanning dependencies of target ros_tutorials_generate_messages_lisp  
[ 27%] Generating C++ code from ros_tutorials/Counts.msg  
[ 36%] Generating Python from MSG ros_tutorials/Counts  
[ 45%] Generating EusLisp manifest code for ros_tutorials  
[ 54%] Generating EusLisp code from ros_tutorials/Counts.msg  
[ 63%] Generating Javascript code from ros_tutorials/Counts.msg  
[ 72%] Generating Lisp code from ros_tutorials/Counts.msg  
[ 72%] Built target ros_tutorials_generate_messages_nodejs  
[ 72%] Built target ros_tutorials_generate_messages_lisp  
[ 72%] Built target ros_tutorials_generate_messages_cpp  
[ 81%] Generating Python msg __init__.py for ros_tutorials  
[ 81%] Built target ros_tutorials_generate_messages_py  
[ 81%] Built target ros_tutorials_generate_messages_eus  
Scanning dependencies of target ros_tutorials_generate_messages  
[ 81%] Built target ros_tutorials_generate_messages  
[ 90%] Linking CXX executable /home/morin/catkin_ws/devel/lib/ros_tutorials/talker  
[ 90%] Built target talker  
[100%] Linking CXX executable /home/morin/catkin_ws/devel/lib/ros_tutorials/listener  
[100%] Built target listener  
morin@morin:~/catkin_ws$
```

# ROS 1 basic practice

## Execute the publisher node

```
> roscore
```

```
> rosrn ros_tutorials talker
```

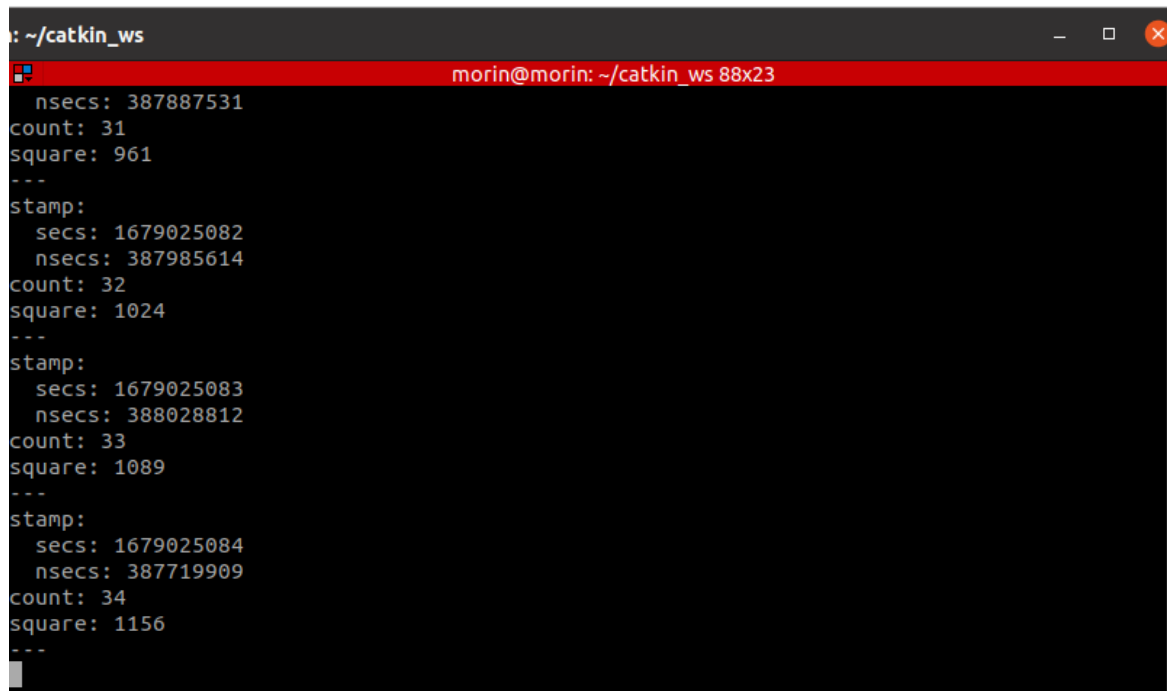
```
morin@morin: ~/catkin_ws 87x23
[ INFO] [1679025079.388322003]: count : 29
[ INFO] [1679025079.388366706]: count square : 841
[ INFO] [1679025080.387857256]: Send msg
[ INFO] [1679025080.387970032]: time : 1679025080
[ INFO] [1679025080.388011333]: count : 30
[ INFO] [1679025080.388044842]: count square : 900
[ INFO] [1679025081.387971887]: Send msg
[ INFO] [1679025081.388109773]: time : 1679025081
[ INFO] [1679025081.388203087]: count : 31
[ INFO] [1679025081.388328571]: count square : 961
[ INFO] [1679025082.388069544]: Send msg
[ INFO] [1679025082.388203816]: time : 1679025082
[ INFO] [1679025082.388252787]: count : 32
[ INFO] [1679025082.388292765]: count square : 1024
[ INFO] [1679025083.388120633]: Send msg
[ INFO] [1679025083.388256427]: time : 1679025083
[ INFO] [1679025083.388304093]: count : 33
[ INFO] [1679025083.388343172]: count square : 1089
[ INFO] [1679025084.387804294]: Send msg
[ INFO] [1679025084.387939367]: time : 1679025084
[ INFO] [1679025084.387986747]: count : 34
[ INFO] [1679025084.388024971]: count square : 1156
```

# ROS 1 basic practice

## rostopic echo

```
> rostopic list
```

```
> rostopic echo /chatter
```



```
~/catkin_ws
morin@morin: ~/catkin_ws 88x23
nsecs: 387887531
count: 31
square: 961
---
stamp:
  secs: 1679025082
  nsecs: 387985614
count: 32
square: 1024
---
stamp:
  secs: 1679025083
  nsecs: 388028812
count: 33
square: 1089
---
stamp:
  secs: 1679025084
  nsecs: 387719909
count: 34
square: 1156
---
```

# ROS 1 basic practice

## Execute the subscriber node

```
> rosrn ros_tutorials listener
```

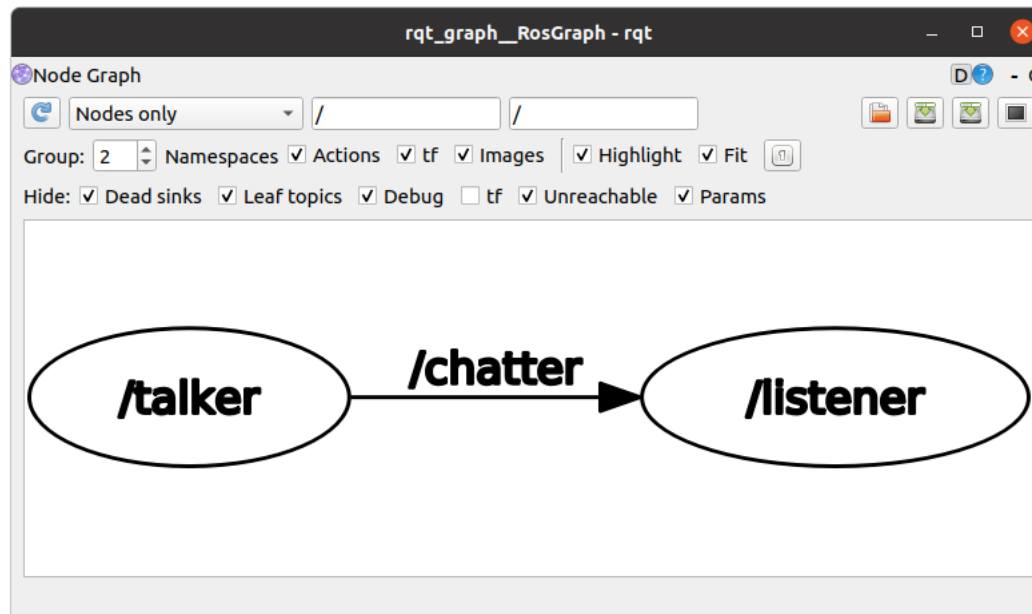
```
morin@morin: ~/catkin_ws 89x23
[ INFO] [1679025079.389439418]: count : 29
[ INFO] [1679025079.389480570]: count square : 841
[ INFO] [1679025080.388678839]: Recieve msg
[ INFO] [1679025080.388775651]: time : 1679025080
[ INFO] [1679025080.388819980]: count : 30
[ INFO] [1679025080.388859230]: count square : 900
[ INFO] [1679025081.389093315]: Recieve msg
[ INFO] [1679025081.389202092]: time : 1679025081
[ INFO] [1679025081.389257220]: count : 31
[ INFO] [1679025081.389295778]: count square : 961
[ INFO] [1679025082.389071669]: Recieve msg
[ INFO] [1679025082.389182022]: time : 1679025082
[ INFO] [1679025082.389229627]: count : 32
[ INFO] [1679025082.389296912]: count square : 1024
[ INFO] [1679025083.389115060]: Recieve msg
[ INFO] [1679025083.389280611]: time : 1679025083
[ INFO] [1679025083.389390551]: count : 33
[ INFO] [1679025083.389467956]: count square : 1089
[ INFO] [1679025084.388842883]: Recieve msg
[ INFO] [1679025084.388974884]: time : 1679025084
[ INFO] [1679025084.389087642]: count : 34
[ INFO] [1679025084.389179440]: count square : 1156
```



# ROS 1 basic practice

## rqt graph

```
> rqt_graph
```



# ROS 1 basic practice


## python version create package & CMakeLists.txt


```
src > ros_tutorials > package.xml
1  <?xml version="1.0"?>
2  <package format="2">
3    <name>ros_tutorials</name>
4    <version>0.0.0</version>
5    <description>The ros_tutorials package</description>
6
7    <maintainer email="your_email@todo.todo">your_name</maintainer>
8
9    <license>BSD</license>
10
11    <buildtool_depend>catkin</buildtool_depend>
12
13    <build_depend>message_generation</build_depend>
14    <build_depend>roscpp</build_depend>
15    <build_depend>rospy</build_depend>
16    <build_depend>std_msgs</build_depend>
17
18    <build_export_depend>roscpp</build_export_depend>
19    <build_export_depend>rospy</build_export_depend>
20    <build_export_depend>std_msgs</build_export_depend>
21
22    <exec_depend>roscpp</exec_depend>
23    <exec_depend>rospy</exec_depend>
24    <exec_depend>std_msgs</exec_depend>
25    <exec_depend>message_runtime</exec_depend>
26
27    <export>
28
29    </export>
30  </package>
```

```
src > ros_tutorials > CMakeLists.txt
1  cmake_minimum_required(VERSION 3.0.2)
2  project(ros_tutorials)
3
4  find_package(catkin REQUIRED COMPONENTS
5    message_generation
6    roscpp
7    std_msgs
8    rospy
9  )
10
11  # Generate messages in the 'msg' folder
12  add_message_files(
13    FILES
14    Counts.msg
15  )
16
17  # Generate added messages and services with any dependencies listed here
18  generate_messages(
19    DEPENDENCIES
20    std_msgs
21  )
22
23  catkin_package(
24    LIBRARIES ros_tutorials
25    CATKIN_DEPENDS message_runtime roscpp std_msgs
26  )
27
28  include_directories(
29    # include
30    ${catkin_INCLUDE_DIRS}
31  )
32
33  add_executable(talker src/talker.cpp)
34  target_link_libraries(talker ${catkin_LIBRARIES})
35
36  add_executable(listener src/listener.cpp)
37  target_link_libraries(listener ${catkin_LIBRARIES})
```

# ROS 1 basic practice

## python version publisher/subscriber

```
src > ros_tutorials > scripts >  talker_py.py
1  #!/usr/bin/env python3
2  import rospy
3  from ros_tutorials.msg import Counts
4
5  def talker():
6      rospy.init_node('talker', anonymous=True)
7
8      pub = rospy.Publisher('chatter', Counts)
9      rate = rospy.Rate(1)
10     msg = Counts()
11
12     count = 0
13
14     while not rospy.is_shutdown():
15         msg.stamp = rospy.Time.now()
16         msg.count = count
17         msg.square = count ** 2
18
19         rospy.loginfo("Send msg")
20         rospy.loginfo("time : %d", msg.stamp.secs)
21         rospy.loginfo("count : %d", msg.count)
22         rospy.loginfo("count square : %d", msg.square)
23
24         pub.publish(msg)
25
26         count += 1
27         rate.sleep()
28
29 if __name__ == '__main__':
30     try:
31         talker()
32     except rospy.ROSInterruptException:
33         pass
```

```
src > ros_tutorials > scripts >  listener_py.py
1  #!/usr/bin/env python3
2  import rospy
3  from ros_tutorials.msg import Counts
4
5  def callback(data):
6      rospy.loginfo("Recieve msg")
7      rospy.loginfo("time : %d", data.stamp.secs)
8      rospy.loginfo("count : %d", data.count)
9      rospy.loginfo("count square : %d", data.square)
10
11 def listener():
12     rospy.init_node('listener', anonymous=True)
13     rospy.Subscriber("chatter", Counts, callback)
14     rospy.spin()
15
16 if __name__ == '__main__':
17     listener()
```

# ROS 1 basic practice

## What is roslaunch?

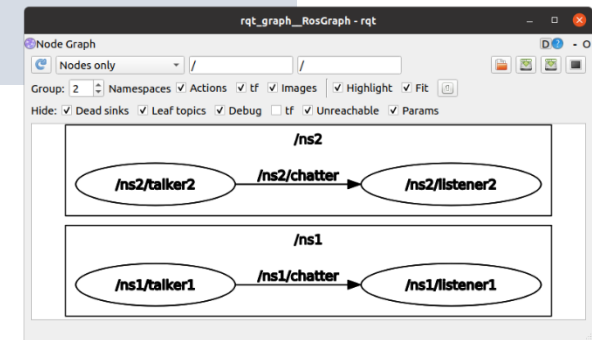
- `roslaunch` is a command to execute a node
- `roslaunch` can run one or more defined nodes
- In addition, `roslaunch` command allows you to **specify options** such as changing package parameters or node names, configuring node namespaces, setting `ROS_ROOT` and `ROS_PACKAGE_PATH`, and changing environment variables when running a node.
- `roslaunch` uses the file `'*.launch'` to set up an executable node, which is XML-based and provides tag-specific options.

# ROS 1 basic practice

## roslaunch

```
> roscd ros_tutorials
> mkdir launch
> cd launch
> code tutorial.launch
```

- **pkg** : Package name
- **type** : The name of the node to actually excute
- **name** : Set the name to be appended



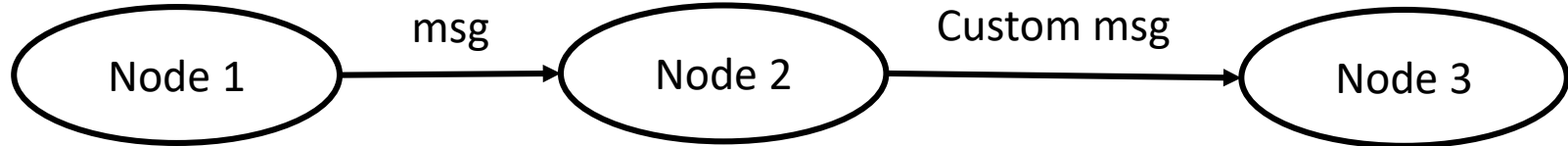
```
src > ros_tutorials > launch > tutorial.launch
1  <launch>
2    <group ns="ns1">
3      <node pkg="ros_tutorials" type="talker" name="talker1" output="screen"/>
4      <node pkg="ros_tutorials" type="listener" name="listener1"/>
5    </group>
6    <group ns="ns2">
7      <node pkg="ros_tutorials" type="talker" name="talker2"/>
8      <node pkg="ros_tutorials" type="listener" name="listener2"/>
9    </group>
10  </launch>
```

# Assignment #1



# Assignment # 1

## Multi node communication(3-6-9 game)



### Create a code that

- **Node 1(Number Generator Node):**  
Publishes sequential integers starting from 1, incrementing by 1 at a frequency of 1 Hz.
- **Node 2(Game Logic Node):**  
Receives numbers from Node 1 and analyzes them. It determines whether each number complies with the 3-6-9 game rules (i.e., if the number contains 3, 6, or 9). Based on this analysis, it publishes a custom message that includes a string indicating "Clap" or "Pass," alongside the respective number.
- **Node 3(Game Output Node):**  
Subscribes to the messages from Node 2. It prints statements like "3 is a clap" or "5 is a pass"
- **Finally, create a Launch file that only the node3 result prints out on terminal**

# Assignment # 1

## Hint for Node2

The following is the skeleton code from last year's assignment.  
Please refer to its structure for guidance.

```
5 //Define class for publisher and subscriber in the same node
6 class SubandPub
7 {
8     public:
9         SubandPub()
10        {
11            /** TODO **/
12            pub = ?? // publisher to node3
13            sub = nh.subscribe(??, ??, &SubandPub::callback, this); // Subscribe from node1
14        }
15
16        void callback(const oddeven::number::ConstPtr& msg) // subscriber callback function
17        {
18            /** TODO **/
19            //Declare message to publish
20            //Check parity(odd or even)
21            //publish to topic
22        }
23
24        private:
25            ros::NodeHandle nh;
26            ros::Publisher pub;
27            ros::Subscriber sub;
28
29 };
30
31 int main(int argc, char **argv)
32 {
33     //Initiate ROS
34     ros::init(argc, argv, "parity_identifier");
35
36     //Create an object of class SubscribeAndPublish that will take care of everything
37     SubandPub SAPObj;
38
39     ros::spin();
40
41     return 0;
42 }
```

# Appendix

# Ubuntu basic command

# Ubuntu basic command practice

## Package management commands

commands		function
<b>sudo</b>		run command as admin
<b>apt</b>		advanced packaging tools
	<b>update</b>	refresh available updates (sudo apt update)
	<b>upgrade</b>	upgrade all packages (sudo apt upgrade)
	<b>install</b>	install package (sudo apt install 'package_name')
	<b>purge</b>	uninstall package (sudo apt purge 'package_name')
	<b>autoremove</b>	remove obsolete packages (sudo apt autoremove)
	<b>search</b>	search the installed packages (apt search 'name')

# Ubuntu basic command practice

## File management commands

commands	function
<b>cd</b>	change directory
<b>ls</b>	directory listing
<b>mkdir</b>	create a directory
<b>touch</b>	create or update a file
<b>mv</b>	move or rename a file
<b>cp</b>	copy a file
<b>rm</b>	delete a file
<b>rmdir</b>	delete an empty directory
<b>pwd</b>	print working directory
<b>cat</b>	show the contents of the file
<b>echo</b>	print the input text
<b>tar</b>	compress and extract the file



# Ubuntu basic command practice

## Searching commands

commands	function
<b>find</b>	search the file
<b>find</b> <i>dir</i> <b>-name</b> <i>file</i>	search the file in some <i>dir</i> name <i>file</i>
<b>grep</b> <i>pattern</i> <i>file</i>	search the <i>pattern</i> in file
<b>grep</b> <b>-r</b> <i>pattern</i> <i>file</i>	search recursively for <i>pattern</i> in <i>dir</i>
<i>command</i>   <b>grep</b> <i>pattern</i>	search for <i>pattern</i> in the output of <i>command</i>

# Ubuntu basic command practice

## Shortcuts

commands	function
<b>Tab</b>	complete the command automatically
<b>ctrl + c</b>	kill the running program in that terminal
<b>ctrl + l</b>	Similar like 'clc' in matlab
<b>ctrl + shift + c</b>	copy from terminal
<b>ctrl + shift + v</b>	paste to terminal

# Ubuntu basic command practice

## Terminal Hotkeys


commands	function
<b>ctrl + alt + t</b>	open the terminal
<b>ctrl + shift + t</b>	open new tab of terminal
<b>ctrl + shift + w</b>	close current tab of terminal
<b>ctrl + shift + q</b>	close all tab of terminal
<b>ctrl + pageup/down</b>	change the tab
<b>ctrl + shift + o</b>	[terminator] split the terminal horizontally
<b>ctrl + shift + e</b>	[terminator] split the terminal vertically

# ROS 2

# ROS basic practice

## ROS 2 installation

<https://docs.ros.org/en/galactic/Installation/Ubuntu-Install-Debians.html>



Search docs

- Installation
  - Ubuntu (Debian)
  - Windows (binary)
  - RHEL (RPM)
- Alternatives
  - Maintain source checkout
  - Testing with pre-release binaries
- DDS implementations
- Distributions
- Tutorials
- How-to Guides
- Concepts
- Contact
- The ROS 2 Project
- Related Projects
- Glossary
- Citations
- Other Versions v: galactic

```

sudo apt update && sudo apt install curl
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros

```

Then add the repository to your sources list.

```

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http

```

### Install ROS 2 packages

Update your apt repository caches after setting up the repositories.

```

sudo apt update

```

ROS 2 packages are built on frequently updated Ubuntu systems. It is always recommended that you ensure your system is up to date before installing new packages.

```

sudo apt upgrade

```

Desktop Install (Recommended): ROS, RViz, demos, tutorials.

```

sudo apt install ros-galactic-desktop

```

ROS-Base Install (Bare Bones): Communication libraries, message packages, command line tools. No GUI tools.

```

sudo apt install ros-galactic-ros-base

```

Development tools: Compilers and other tools to build ROS packages

# ROS 2

## ROS 1 vs. ROS 2



### 개발 당시 컨셉

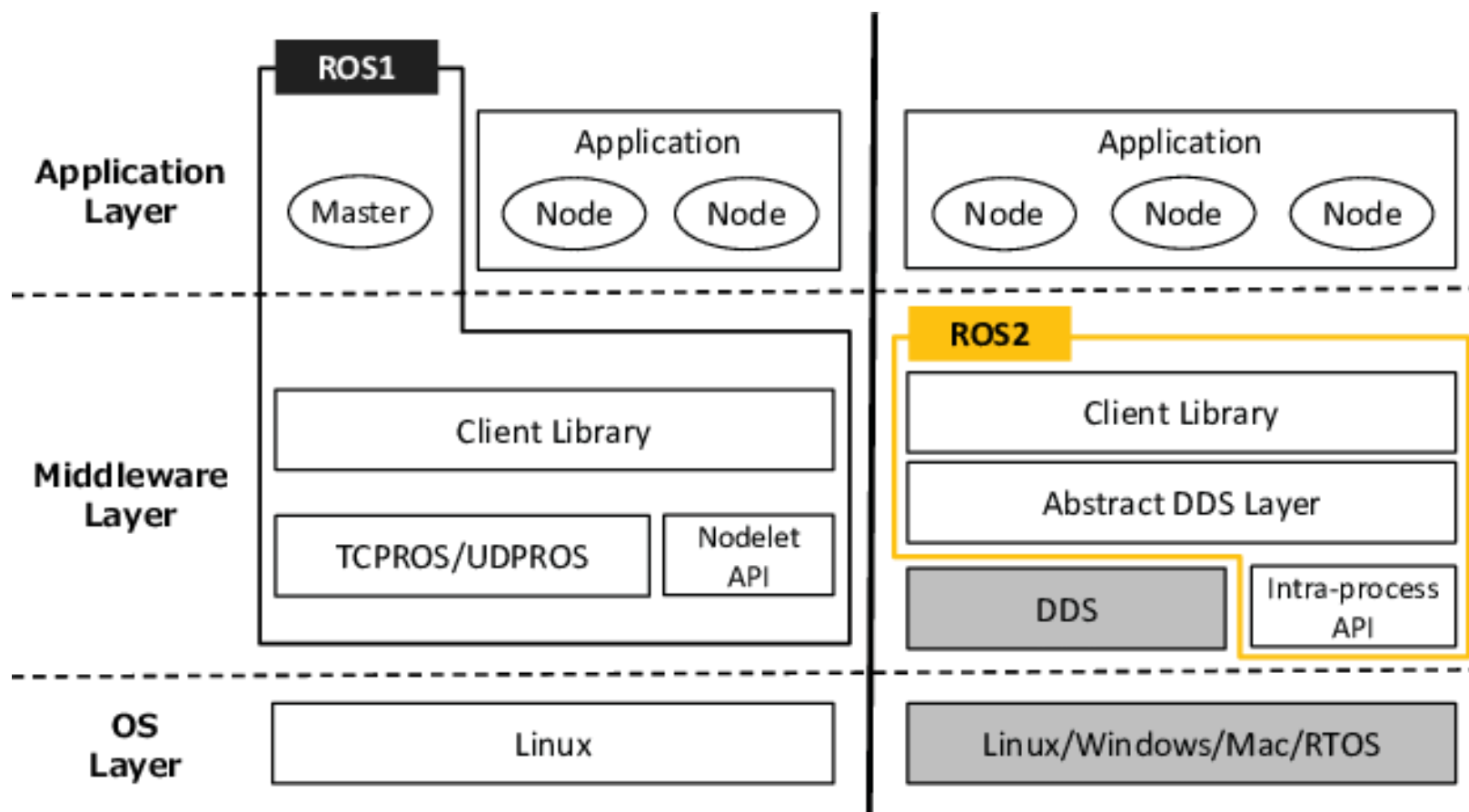
- 단일로봇
- 워크스테이션급 컴퓨터
- Linux 환경
- 실시간 제어 지원 x
- 안정된 네트워크 환경 요구
- 주로 대학이나 연구소와 같은  
아카데미 연구 용도

### 새롭게 요구되는 기능들

- 복수대의 로봇
- 임베디드 시스템에서의 ROS 사용
- 실시간 제어
- 불안정한 네트워크 환경에서도 동작
- 멀티 플랫폼
- 최신기술지원
- 상업용 제품 지원

# ROS 2

## ROS 1 vs. ROS 2





Thank you!

---