What is ROS?

ROS (Robot Operating System) is designed to manage robotic components via a computer. In the ROS framework, various standalone nodes interact through a publish/subscribe messaging model. To illustrate, a node could represent the driver for a specific sensor, publishing its data through a stream of messages. These messages can then be utilized by numerous other nodes, encompassing filters, loggers, and more sophisticated systems like guidance and pathfinding.

Why ROS?

It's crucial to recognize that within ROS, nodes are not constrained to reside on the same system; they can span across multiple computers and even differ in architecture. Picture an Arduino dispatching messages, a laptop receiving them, and an Android phone maneuvering motors—all seamlessly orchestrated within ROS. This characteristic endows ROS with remarkable flexibility, tailoring itself adeptly to diverse user requirements. Additionally, ROS operates as an open-source system, upheld by a collaborative community of contributors.

Follow these instructions to install ROS: https://wiki.ros.org/noetic/Installation/Ubuntu

All tutorials can be found here: https://wiki.ros.org/ROS/Tutorials

It is highly recommended that you go through tutorials numbered 1 to 13 and 17.

Following is a brief summary of the most essential tutorials.

Step 1: source ROS

```
$ source /opt/ros/kinetic/setup.bash
```

You will need to run this command on every new shell you open to have access to the ROS commands, unless you add this line to your .bashrc.

Step 2: Create your workspace

A catkin workspace refers to a directory where you store and develop your ROS-related packages. A workspace typically contains multiple packages, each representing a distinct unit of functionality or a robotic component. These packages can include libraries, executables, configuration files, and more..

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

```
$ source devel/setup.bash
```

To make sure your workspace is properly overlayed by the setup script, make sure ROS_PACKAGE_PATH environment variable includes the directory you're in.

```
$ echo $ROS_PACKAGE_PATH
```

You would see the output of the above command as follows:

```
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```

Step 3: Create a ROS Package

1. Install ros tutorials package

```
$ sudo apt-get install ros-kinetic-ros-tutorials
```

2. Go to source space directory of the catkin workspace you created earlier.

```
$ cd ~/catkin_ws/src
```

3. Now use the catkin_create_pkg script to create a new package called 'beginner_tutorials' which depends on std_msgs, roscpp, and rospy.

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

ROS packages sometimes require external libraries and tools that must be provided by the operating system. These required libraries and tools are commonly referred to as *system dependencies*. In some cases these *system dependencies* are not installed by default.

Std_msgs: Standard ROS Messages including common message types representing primitive data types and other basic message constructs, such as multiarrays.

roscpp, rospy: c++ and python libraries

4. Now you need to build the packages in the catkin workspace:.

```
$ cd ~/catkin_ws
$ catkin_make
```

5. To add the workspace to your ROS environment you need to source the generated setup file:

```
$ . ~/catkin_ws/devel/setup.bash
```

6. Build your ROS package

```
$ cd ~/catkin_ws/

$ catkin_make
```

Building a ROS (Robot Operating System) package involves compiling the source code and generating executable binaries from the package's codebase.

Step 4: Writing a publisher node

Important terms:

- Nodes: A node is an executable that uses ROS to communicate with other nodes.

- Messages: ROS data type used when subscribing or publishing to a topic.
- Topics: Nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages.

Also checkout

https://wiki.ros.org/ROS/Tutorials/UnderstandingNodes
https://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

We will now see an example of a node that we will write in our beginner_tutorials package. The publisher node will continuously broadcast/publish a message to a specific topic named.

```
$ roscd beginner_tutorials
```

Download the example script talker.py to your new `scripts` directory and make it executable:

```
$ wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials/001_talker_listener/talker.py
```

```
$ chmod +x talker.py
```

Alternately you can create a text file named talker.py and copy paste the code.

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

The code publishes a string to the topic named “chatter”. The queue size of this topic is 10 which means the topic will hold the 10 latest messages published on the topic.

We will now create a subscriber node that will subscribe to the topic “chatter” and we can then access the messages that the publisher is broadcasting.

Step 5: Writing a subscriber node (listener.py)

```
$ roscd beginner_tutorials/scripts/
$ wget https://raw.github.com/ros/ros_tutorials/kinetic-
devel/rospy_tutorials/001_talker_listener/listener.py
$ chmod +x listener.py
```

Alternately you can create a text file named subscriber.py and copy paste the code.

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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

The above code subscribes to the topic named “chatter”. Everytime it sees a message on “chatter” it will call the function callback, which prints the message.

Then, edit the catkin_install_python() call in your CMakeLists.txt so it looks like the following:

```
catkin_install_python(PROGRAMS scripts/talker.py scripts/listener.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

Step 6: Build your nodes

```
$ cd ~/catkin_ws
$ catkin_make
```

Step 7: Running your program!

Now that you have built the publisher and subscriber nodes, its time to run it and see what happens.

The very first thing you need to do when run ROS is the roscore command. In a new terminal:

```
$ roscore
```

Make sure you have sourced your workspace's setup.sh file after calling catkin_make but before trying to use your applications.

```
$ cd ~/catkin_ws
$ source ./devel/setup.bash
```

Run the talker.

```
$ rosrun beginner_tutorials talker.py
```

In another terminal run the listerner.

```
$ rosrun beginner_tutorials listener.py
```

Observe the output being printed out for both codes.

You have now successfully implemented a publisher/subscriber

Step 7: Rostopic

In a normal ROS program, you wont be printing the data that is being published or subscribed. For example if a sensor node is publishing sensor data it will be redundant to print the data in the code. The easiest way to see what messages are being published to a topic is by using the rostopic command. In our talker listerner code, you can do the following:

In a new terminal:

```
$ rostopic echo /chatter
```