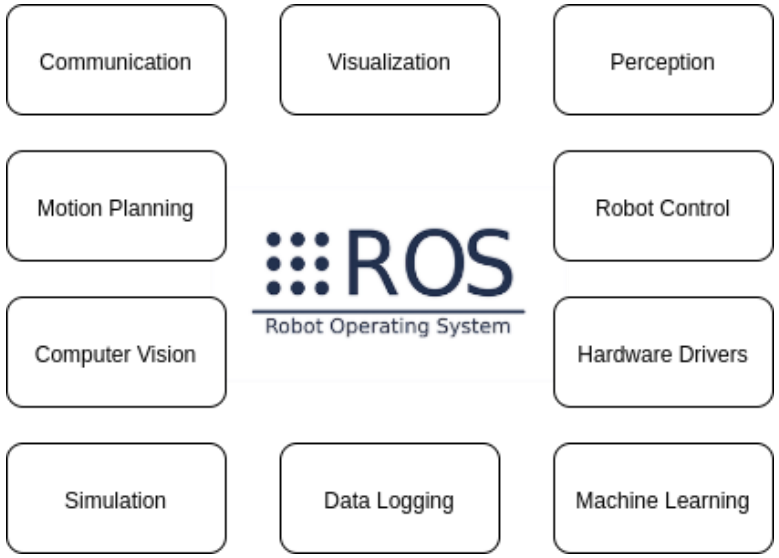
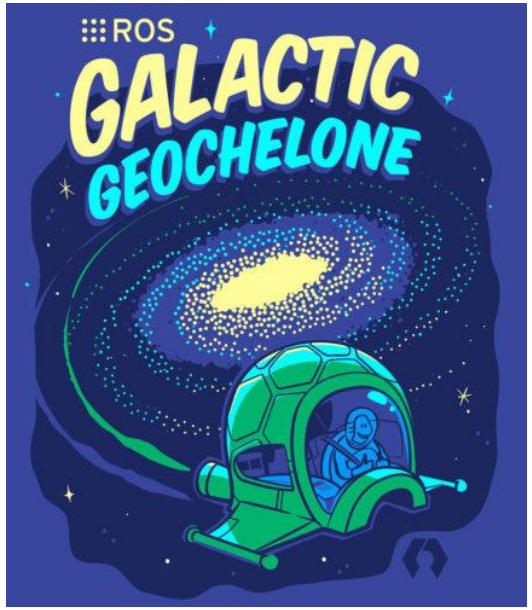
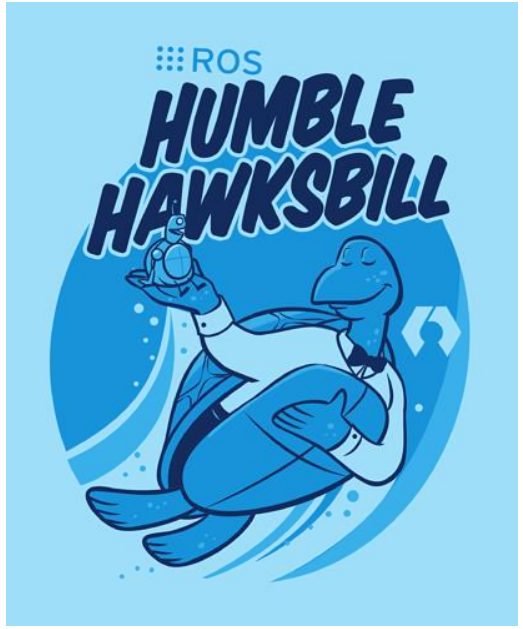


ROS2





http://design.ros2.org/

https://navigation.ros.org/roadmap/roadmap.html

ALWAYS MATCH ROS DISTRIBUTION WITH UBUNTU (OR LINUX) DISTRIBUTION

- **Humble Roadmap¶**
- This is the list of major issues and features the Nav2 maintainers are commiting for completion for the ROS 2 Humble Release in 2022. This is *not* an exhaustive list of planned features or what changes may be found in the new distribution. It represents only the items of direct commitment to give insight into commitments for REP-2005 repositories in the ROS 2 Roadmap. For a full list of important completed changes in the project, see the Migration Guide Galactic to Humble.

WHY ROS2

Of specific interest to us for the ongoing and future growth of the ROS community are the following use cases, which we did not have in mind at the beginning of the project:

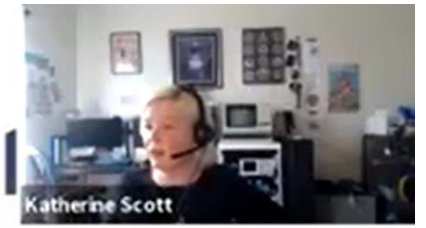
- **Teams of multiple robots**: while it is possible to build multi-robot systems using ROS today, there is no standard approach, and they are all somewhat of a hack on top of the single-master structure of ROS.
- Small embedded platforms: we want small computers, including “bare-metal” micro controllers, to be first-class participants in the ROS environment, instead of being segregated from ROS by a device driver.
- **Real-time systems**: we want to support real-time control directly in ROS, including inter-process and inter-machine communication (assuming appropriate operating system and/or hardware support).
- **Non-ideal networks**: we want ROS to behave as well as is possible when network connectivity degrades due to loss and/or delay, from poor-quality WiFi to ground-to-space communication links.
- **Production environments**: while it is vital that ROS continue to be the platform of choice in the research lab, we want to ensure that ROS-based lab prototypes can evolve into ROS-based products suitable for use in real-world applications.
- Prescribed patterns for building and structuring systems: while we will maintain the underlying flexibility that is the hallmark of ROS, we want to provide clear patterns and supporting tools for features such as life cycle management and static configurations for deployment.

https://design.ros2.org/articles/why_ros2.html

Technical Steering Committee

open robotics

Why ROS 2? Robotics, Autonomy, Industrial



ROS 2 TSC

ADLINK Apex.AI

BOSCH TOYOTA RESEARCH INSTITUTE

Microsoft amazon

ubuntu intel SAMSUNG

PICKNIK open robotics

DEVCOM LG Electronics

ROVER ROBOTICS eProsima The Middleware Experts

iRobot SONY

ROBOTIS WIND

ROS Industrial

78 Members

bit.ly/ROSIMembers

Government

- NASA
- NHSTA / USDOT
- DARPA
- Army / Navy / AF
- NIST
- Dozens of Universities
- Singapore Hospital System

Want to work at one of these places? They all use ROS

EXAMPLE ROS2 LINE COMMANDS AND USEFUL PACKAGES

- ros2 run turtlesim turtlesim_node
- ros2 node info /your_node_name
- ros2 topic type /your_topic
- ros2 interface show geometry_msgs/msg/Twist

rviz – Provides a 3D visualizer for ROS messages.
rqt_console – Provides a GUI for displaying and filtering ROS messages.
rqt_graph – Visualizes the ROS computation graph – i.e. nodes and the topics they publish and subscribe to.
rqt_image_view – Displays camera images using image_transport.
rqt_plot – Plots numeric values on a 2D chart.

https://foxglove.dev/blog/the-building-blocks-of-ros2

https://github.com/ubuntu-robotics/ros2_cheats_sheet/blob/master/cli/cli_cheats_sheet.pdf

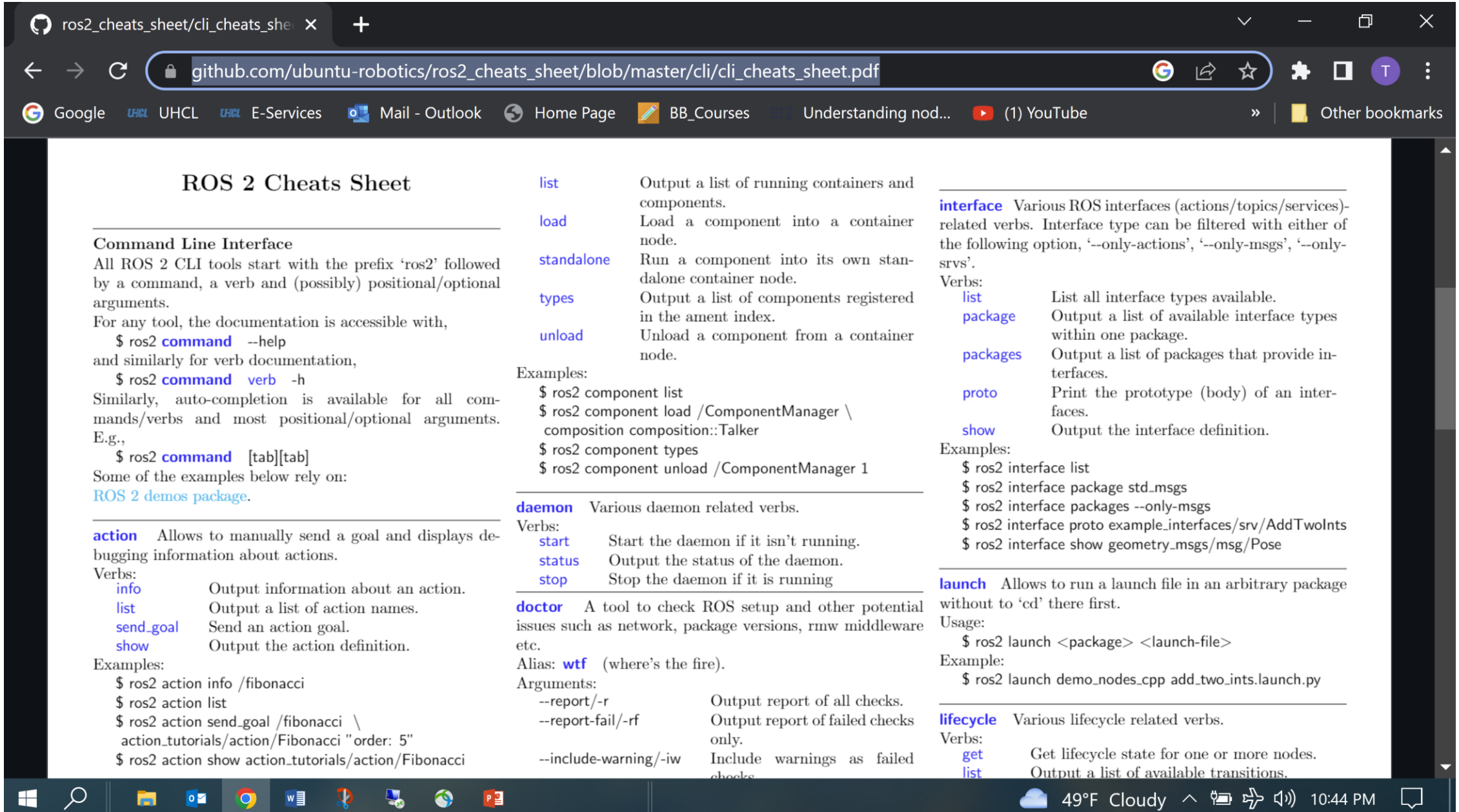
Alias foxy or noetic
harman@harman-VirtualBox:~$ foxy
harman@harman-VirtualBox:~$ **ros2 run turtlesim turtlesim_node**
[INFO] [1668991198.146164317] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1668991198.151565206] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]

Alias foxy or noetic TERMINAL 2
harman@harman-VirtualBox:~$ foxy
harman@harman-VirtualBox:~$ **ros2 node info /turtlesim**
LIST OF
Subscribers: Publishers: Service Servers: Action Servers:

harman@harman-VirtualBox:~$ **ros2 topic type /turtle1/cmd_vel**
geometry_msgs/msg/Twist

harman@harman-VirtualBox:~$ **ros2 interface show geometry_msgs/msg/Twist**
Vector3 linear
Vector3 angular

https://github.com/ubuntu-robotics/ros2_cheats_sheet/blob/master/cli/cli_cheats_sheet.pdf



ROS 2 Cheats Sheet

Command Line Interface

All ROS 2 CLI tools start with the prefix 'ros2' followed by a command, a verb and (possibly) positional/optional arguments.
For any tool, the documentation is accessible with,

```
$ ros2 command --help
```

and similarly for verb documentation,

```
$ ros2 command verb -h
```

Similarly, auto-completion is available for all commands/verbs and most positional/optional arguments. E.g.,

```
$ ros2 command [tab][tab]
```

Some of the examples below rely on:
ROS 2 demos package.

action Allows to manually send a goal and displays debugging information about actions.
Verbs:

info	Output information about an action.
list	Output a list of action names.
send_goal	Send an action goal.
show	Output the action definition.

Examples:

```
$ ros2 action info /fibonacci
$ ros2 action list
$ ros2 action send_goal /fibonacci \
 action_tutorials/action/Fibonacci "order: 5"
$ ros2 action show action_tutorials/action/Fibonacci
```

list	Output a list of running containers and components.
load	Load a component into a container node.
standalone	Run a component into its own standalone container node.
types	Output a list of components registered in the ament index.
unload	Unload a component from a container node.

Examples:

```
$ ros2 component list
$ ros2 component load /ComponentManager \
 composition composition::Talker
$ ros2 component types
$ ros2 component unload /ComponentManager 1
```

daemon Various daemon related verbs.
Verbs:

start	Start the daemon if it isn't running.
status	Output the status of the daemon.
stop	Stop the daemon if it is running

doctor A tool to check ROS setup and other potential issues such as network, package versions, rmw middleware etc.
Alias: **wtf** (where's the fire).
Arguments:

--report/-r	Output report of all checks.
--report-fail/-rf	Output report of failed checks only.
--include-warning/-iw	Include warnings as failed checks

interface Various ROS interfaces (actions/topics/services)-related verbs. Interface type can be filtered with either of the following option, '--only-actions', '--only-msgs', '--only-srvs'.
Verbs:

list	List all interface types available.
package	Output a list of available interface types within one package.
packages	Output a list of packages that provide interfaces.
proto	Print the prototype (body) of an interfaces.
show	Output the interface definition.

Examples:

```
$ ros2 interface list
$ ros2 interface package std_msgs
$ ros2 interface packages --only-msgs
$ ros2 interface proto example_interfaces/srv/AddTwoInts
$ ros2 interface show geometry_msgs/msg/Pose
```

launch Allows to run a launch file in an arbitrary package without to 'cd' there first.
Usage:

```
$ ros2 launch <package> <launch-file>
```

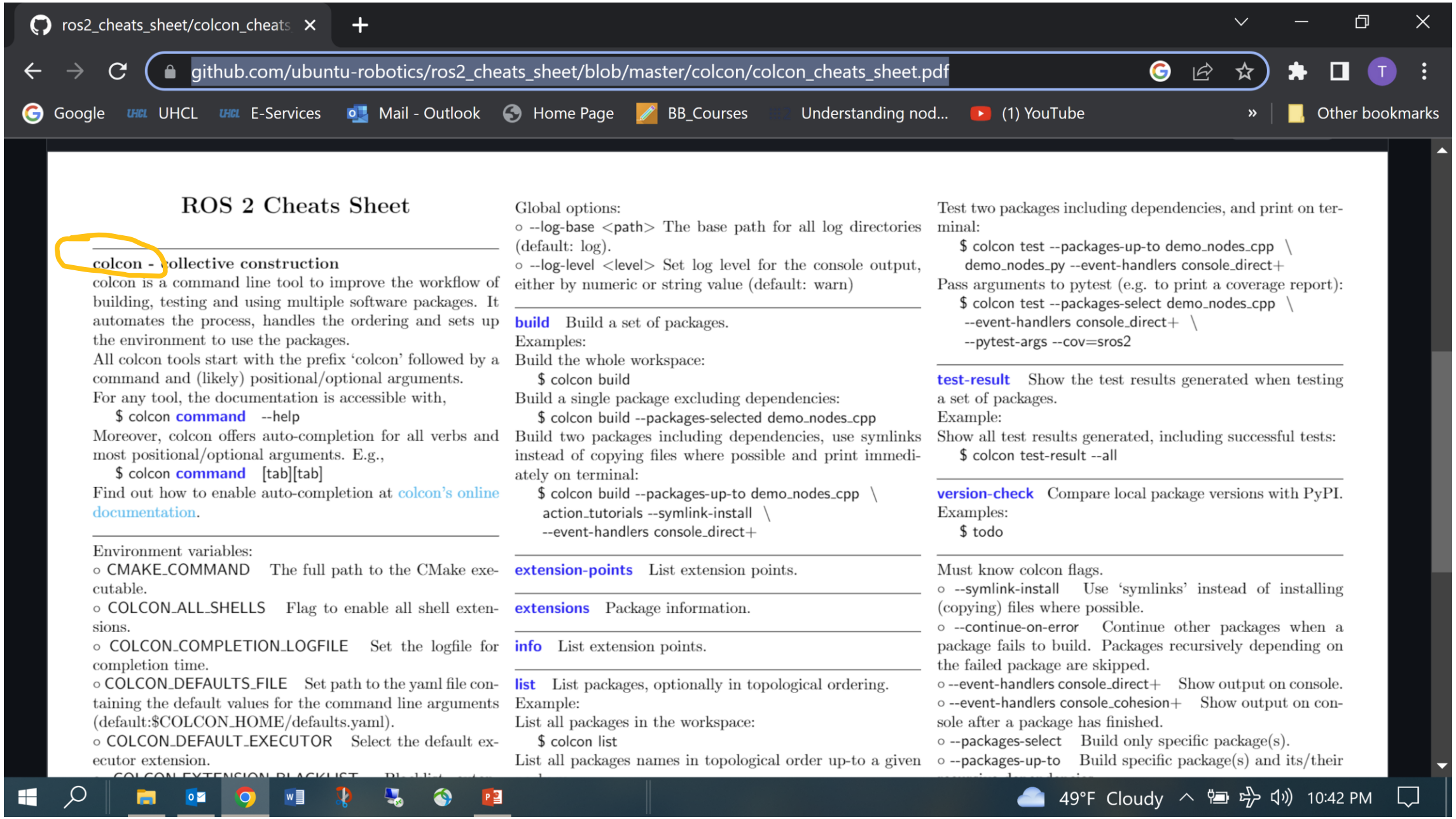
Example:

```
$ ros2 launch demo_nodes_cpp add_two_ints.launch.py
```

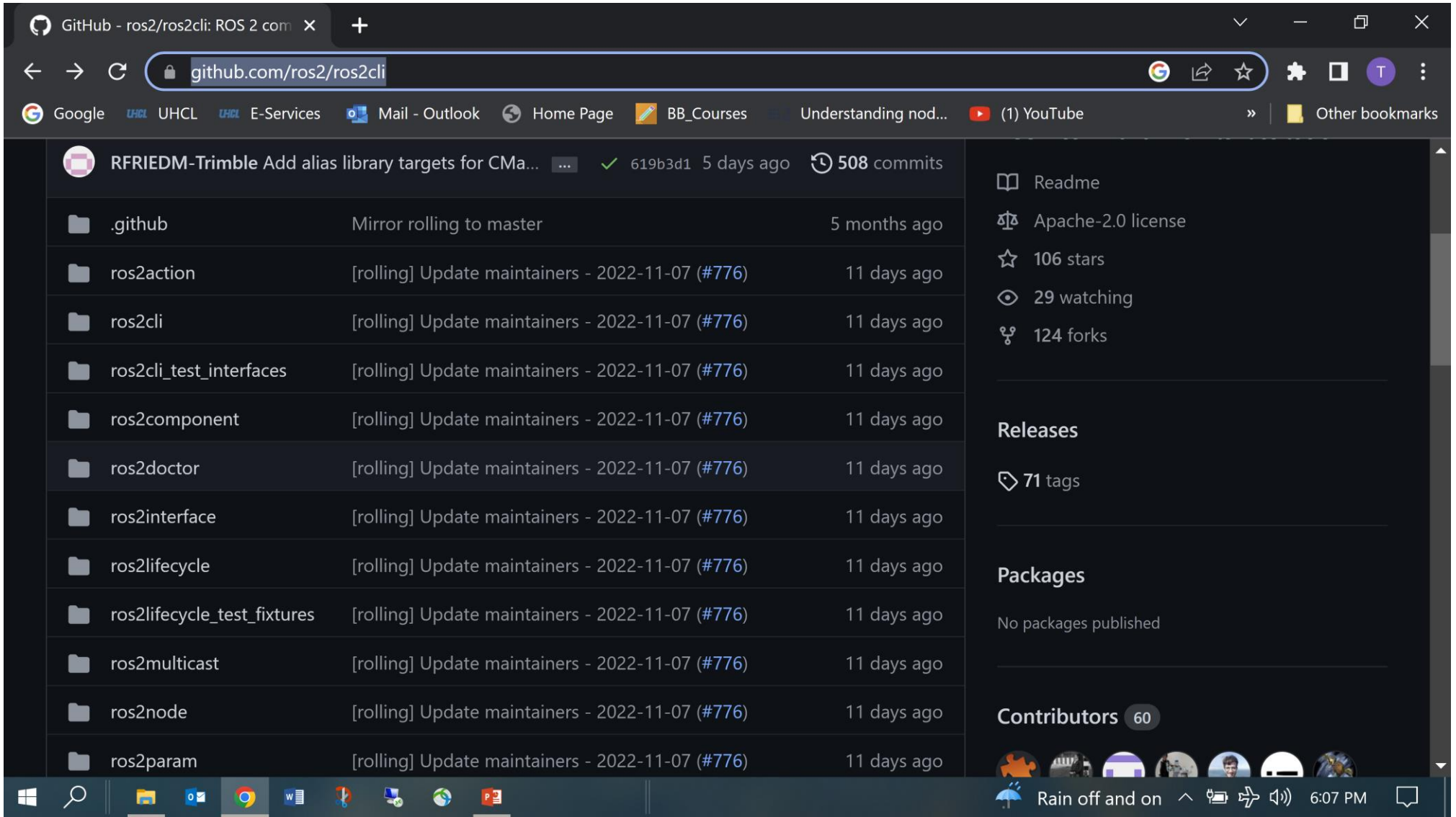
lifecycle Various lifecycle related verbs.
Verbs:

get	Get lifecycle state for one or more nodes.
list	Output a list of available transitions.

https://github.com/ubuntu-robotics/ros2_cheats_sheet/blob/master/colcon/colcon_cheats_sheet.pdf



https://github.com/ros2/ros2cli



What Is ROS2? - Framework Overview
LET'S WATCH 8:21

https://www.youtube.com/watch?v=7TVWlADXwRw

17,539 views Jun 18, 2021
Let's look at what ROS2 has to offer when programming robotics systems. We will take a dive into ROS version 2 features, and go over some points on why it succeeded ROS version 1.

Check out my related Udemy Courses to learn more:
- ROS2 In Python: https://rebrand.ly/udemy-ros2-python-p
- ROS2 In C++: https://rebrand.ly/udemy-ros2-cpp-u
- ROS1 In Python: https://rebrand.ly/udemy-ros-python-p
- ROS1 In C++: https://rebrand.ly/udemy-ros-cpp-p
- MongoDB: https://rebrand.ly/udemy-mongodb-pyth...

Raymond Andrade

ROS2 SUPPORT & SOFTWARE

https://classic.gazebosim.org/tutorials?tut=ros_wrapper_versions&cat=connect_ros

- ***packages.ros.org***
  - ROS Melodic: Gazebo 9.x
  - ROS Noetic: Gazebo 11.x
  - ROS2 Foxy: Gazebo 11.x
  - ROS2 Rolling: Gazebo 11.x

ALWAYS CHECK

- RVIZ2 for Foxy and later
- Gazebo 11 +
- Python 3 +
- C++ Foxy targets C++14.

https://docs.ros.org/en/foxy/The-ROS2-Project/Contributing/Code-Style-Language-Versions.html

ROS2 CLIENT LIBRARIES

https://docs.ros.org/en/foxy/Concepts/About-ROS-2-Client-Libraries.html

The C++ client library (rclcpp) and the Python client library (rclpy) are both client libraries which utilize common functionality in the RCL.

The rclpy repository is located on GitHub at ros2/rclpy and contains the package rclpy. The generated API documentation is here:

https://github.com/ros2/rclpy

https://docs.ros2.org/foxy/api/rclpy/index.html

Node

class rclpy.node.Node(node_name, *, context=None, cli_args=None, namespace=None, use_global_arguments=True, enable_rosout=True, start_parameter_services=True, parameter_overrides=None, allow_undeclared_parameters=False, automatically_declare_parameters_from_overrides=False)

Create a Node.

Parameters

node_name (str) – A name to give to this node. Validated by validate_node_name().

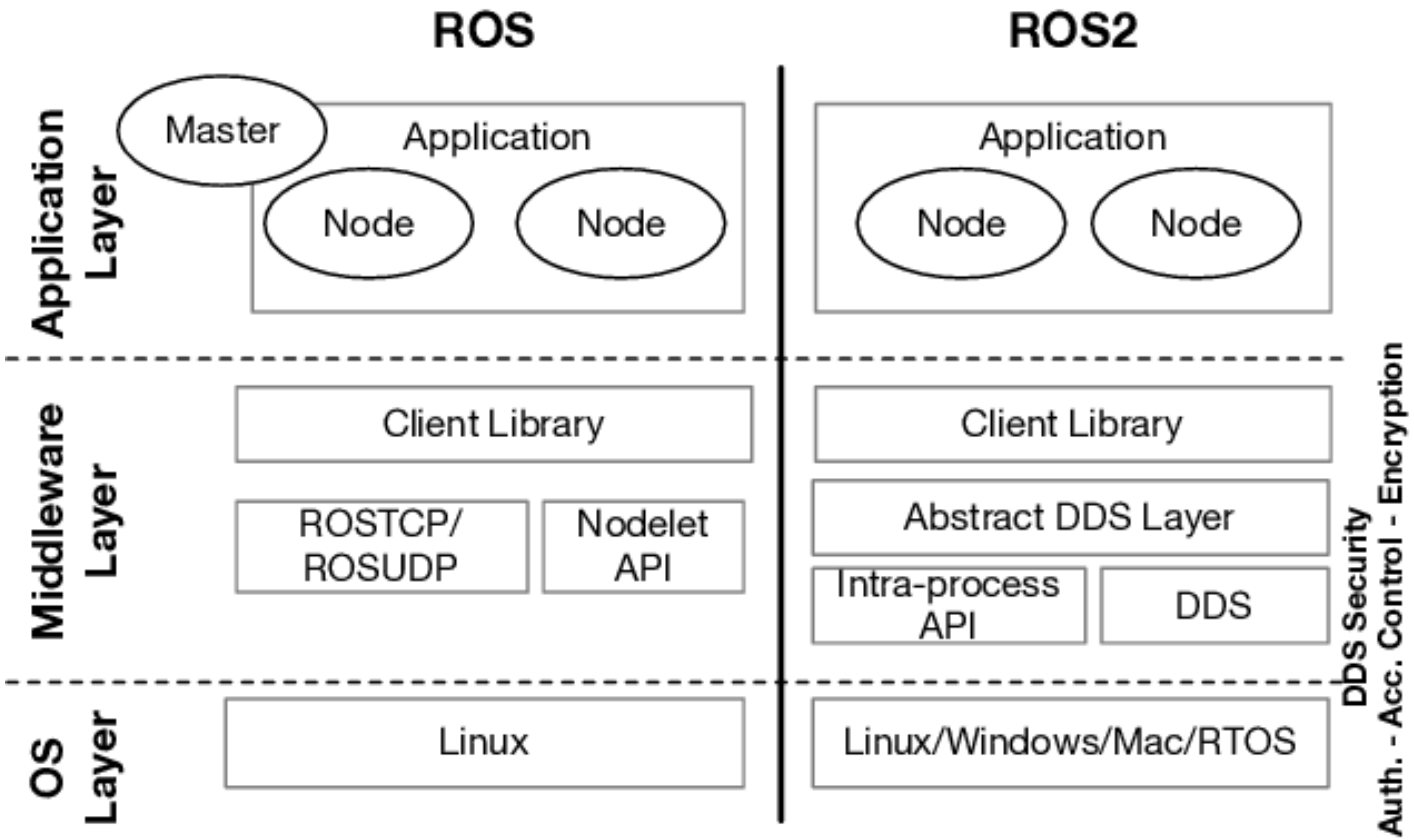
https://docs.ros2.org/foxy/api/rclpy/api/node.html

Once you have created the node, you can use it to start ROS2 publishers, subscribers, services, get parameters, etc.

```
import rclpy
from rclpy.node import Node
def main(args=None):
    rclpy.init(args=args)          # To instantiate a node
    node = Node('my_node_name')
    rclpy.spin(node)
    rclpy.shutdown()
if __name__ == '__main__':
   main()
```

https://roboticsbackend.com/write-minimal-ros2-python-node/

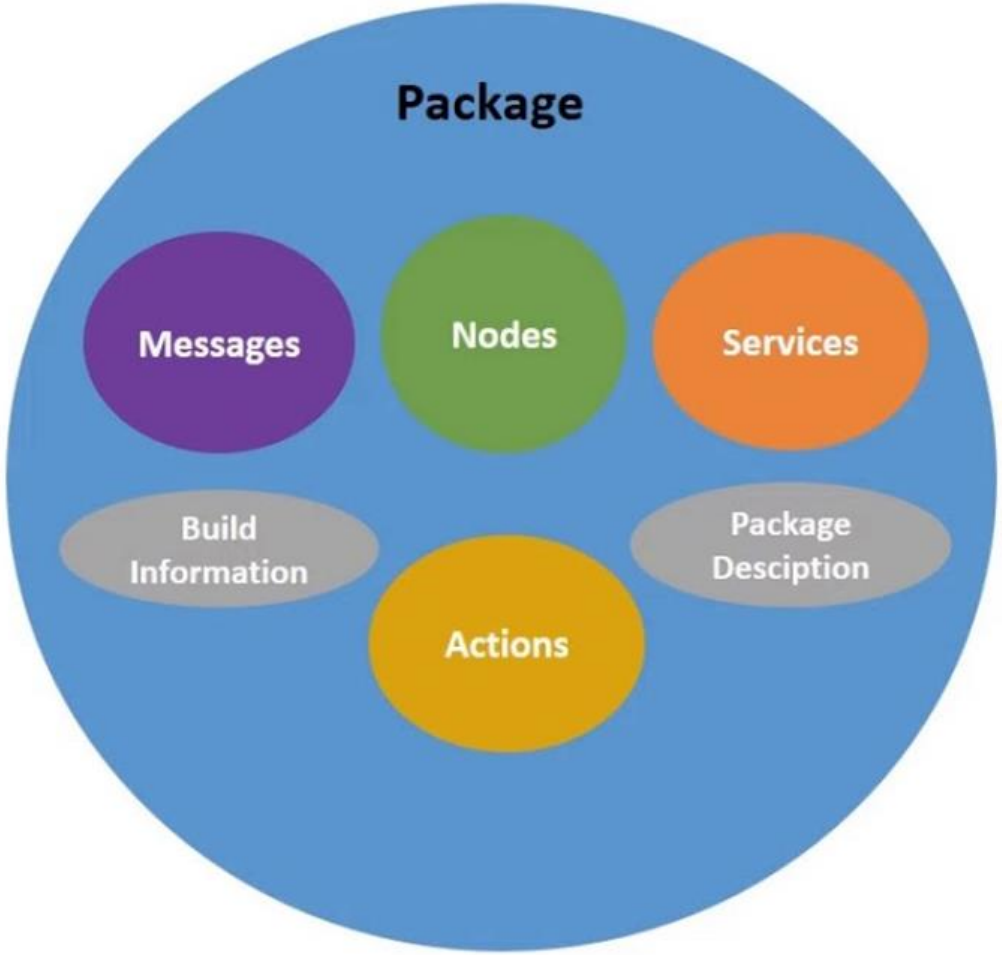
ROS2 STRUCTURE




```
workspace_folder/
    src/
        package_1/
            CMakeLists.txt
            package.xml
        package_2/
            setup.py
            package.xml
            Resource/package_2

        package_n/
            CMakeLists.txt
            package.xml
```

Ros2 Workspace and Package



1 - Ros2 Workspace and Clone Sample Package Foxy

https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html#new-directory

$ source /opt/ros/foxy/setup.bash

$mkdir -p ~/ros2_ws/src
$cd ~/ros2_ws/src

CLONE Ensure you're still in the ros2_ws/src directory before you clone.

$ git clone https://github.com/ros/ros_tutorials.git -b foxy-devel

NOTE: THIS IS SEPARATE FROM THE TURTLESIM TUTORIALS LOADED WITH ROS FOXY – DESKTOP FULL

See Section 7 in the document - Modify the overlay

https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html#new-directory

From the root of your workspace (ros2_ws), you can now build your packages using the command:

$ colcon build

$ source /opt/ros/foxy/setup.bash (Called the Underlay – ros files)

6 Source the overlay – your ws

$ cd ~/ros2_ws

Note: In the root of ws, source your overlay:

$. install/local_setup.bash

Note: **. = source**

Sourcing the local_setup of the overlay will only add the packages available in the overlay to your environment. setup sources the overlay as well as the underlay it was created in, allowing you to **utilize both workspaces.**

Make_ros_ws22 **Make ws and /src**

Alias foxy or noetic
harman@harman-VirtualBox:~$ **foxy**

harman@harman-VirtualBox:~$ **mkdir -p ~/ros2_ws22/src**

harman@harman-VirtualBox:~$ **cd ~/ros2_ws22**
harman@harman-VirtualBox:~/ros_ws22$ **ls**
src

https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

Type this command to create a C++ package:

$ ros2 pkg create --build-type ament_cmake my_package

Type this command to create a Python package:

$ ros2 pkg create --build-type ament_python my_package

https://automaticaddison.com/how-to-create-a-package-ros-2-foxy-fitzroy/

EXAMPLE: ADD DEPENDENCIES

$ ros2 pkg create --build-type ament_python my_package --dependencies **rclpy** image_transport cv_bridge sensor_msgs std_msgs opencv2

https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

package.xml file containing meta information about the package

setup.py containing instructions for how to install the package

setup.cfg is required when a package has executables, so ros2 run can find them

/<package_name> - a directory with the same name as your package, used by ROS 2 tools to find your package, contains __init__.py

Foxy sourced in every terminal

harman@harman-VirtualBox:~/ros2_ws22/src$ **ros2 pkg create --build-type ament_cmake tlh_package**
going to create a new package
package name: tlh_package
destination directory: /home/harman/ros2_ws22/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['harman <harman@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: []
creating folder ./tlh_package
creating ./tlh_package/package.xml
creating source and include folder
creating folder ./tlh_package/src
creating folder ./tlh_package/include/tlh_package
creating ./tlh_package/CMakeLists.txt

```
harman@harman-VirtualBox:~/ros2_ws22/src$ ls
  tlh_package
harman@harman-VirtualBox:~/ros2_ws22/src$ cd tlh_package
harman@harman-VirtualBox:~/ros2_ws22/src/tlh_package$ ls
CMakeLists.txt  include  package.xml  src
```

Source install/local_setup.bash for Workspace

```
harman@harman-VirtualBox:~/ros2_ws22$ colcon build
Starting >>> tlh_package
Finished <<< tlh_package [1.34s]

Summary: 1 package finished [1.64s]
```

```
harman@harman-VirtualBox:~/ros2_ws22$ tree -L 1
.
├── build
├── install
├── log
└── src
```

```
harman@harman-VirtualBox:~/ros2_ws22$ tree -L 2
.
├── build
│   ├── COLCON_IGNORE
│   └── tlh_package
├── install
│   ├── COLCON_IGNORE
│   ├── local_setup.bash
│   ├── local_setup.ps1
│   ├── local_setup.sh
│   ├── _local_setup_util_ps1.py
│   ├── _local_setup_util_sh.py
│   ├── local_setup.zsh
│   ├── setup.bash
│   ├── setup.ps1
│   ├── setup.sh
│   ├── setup.zsh
│   └── tlh_package
├── log
│   ├── build_2022-11-21_12-44-43
│   ├── COLCON_IGNORE
│   ├── latest -> latest_build
│   └── latest_build -> build_2022-11-21_12-44-43
└── src
    └── tlh_package

10 directories, 13 files
```

2- MAKE A WORKSPACE AND OUR OWN PACKAGE

ros2_ws2

$ mkdir -p ~/ros2_ws2/src && cd ros2_ws2/ && colcon build

Alias foxy or noetic

harman@harman-VirtualBox:~$ foxy (Source foxy)

harman@harman-VirtualBox:~$ **mkdir -p ~/ros2_ws2/src && cd ros2_ws2/ && colcon** build

Summary: 0 packages finished [0.36s]

harman@harman-VirtualBox:~/ros2_ws2$ tree -L 1

```
.
├── build
├── install
├── log
└── src
```

tlh_python_pkg and tlh_python_node.py

- In ros2_ws2 CREATE A PACKAGE **tlh_python_pkg** FOR A PYTHON SCRIPT
- **Colcon Build**
- Go to the package directory and create a **python node**

MODIFY THE SETUP.PY AND PACKAGE.XML FILES

- **Add entry point** to **setup.py**
- **Add** <**buildtool**_depend> and **<depend>rclpy</depend>** to **package.xml**
- **Colcon Build Again**
- **ros2 run tlh_python_pkg test**

https://roboticsbackend.com/create-a-ros2-python-package/

Table of Contents

tlh_python_pkg

harman@harman-VirtualBox:~/ros2_ws2/src$ **ros2 pkg create tlh_python_pkg --build-type ament_python**
going to create a new package
package name: tlh_python_pkg
destination directory: /home/harman/ros2_ws2/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['harman <harman@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
creating folder ./tlh_python_pkg
creating ./tlh_python_pkg/package.xml
creating source folder
creating folder ./tlh_python_pkg/tlh_python_pkg
creating ./tlh_python_pkg/setup.py
creating ./tlh_python_pkg/setup.cfg
creating folder ./tlh_python_pkg/resource
creating ./tlh_python_pkg/resource/tlh_python_pkg
creating ./tlh_python_pkg/tlh_python_pkg/__init__.py
creating folder ./tlh_python_pkg/test
creating ./tlh_python_pkg/test/test_copyright.py
creating ./tlh_python_pkg/test/test_flake8.py
creating ./tlh_python_pkg/test/test_pep257.py

harman@harman-VirtualBox:~/ros2_ws2/src$ **tree -L 2**

```
.
└── tlh_python_pkg
    ├── package.xml
    ├── resource
    ├── setup.cfg
    ├── setup.py
    ├── test
    └── tlh_python_pkg
```

harman@harman-VirtualBox:~/ros2_ws2/src/tlh_python_pkg$ **tree -L 2**

```
.
├── package.xml
├── resource
│   └── tlh_python_pkg
├── setup.cfg
├── setup.py
├── test
│   ├── test_copyright.py
│   ├── test_flake8.py
│   └── test_pep257.py
└── tlh_python_pkg
    └── __init__.py

3 directories, 8 files
```

harman@harman-VirtualBox:~/ros2_ws2/src/tlh_python_pkg$ **gedit package.xml**

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>tlh_python_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="harman@todo.todo">harman</maintainer>
  <license>TODO: License declaration</license>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

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

If you know what a CMakeLists.txt file is, well the setup.py is basically the same but for Python. When you compile your package it will tell what to install, where to install it, how to link dependencies, etc.

```
from setuptools import setup

package_name = 'tlh_python_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='harman',
    maintainer_email='harman@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        ],
    },
```

JUST LOOKING

harman@harman-VirtualBox:~/ros2_ws2/src/tlh_python_pkg$ **gedit setup.cfg**

```
[develop]
script-dir=$base/lib/tlh_python_pkg
[install]
install-scripts=$base/lib/tlh_python_pkg
```

harman@harman-VirtualBox:~/ros2_ws2/src/tlh_python_pkg/tlh_python_pkg$ **ls**
__init__.py

BUILD AGAIN – JUST ONE PACKAGE

harman@harman-VirtualBox:~/ros2_ws2$ **colcon build --packages-select tlh_python_pkg**
Starting >>> tlh_python_pkg
Finished <<< tlh_python_pkg [1.02s]

Summary: 1 package finished [1.18s]

```
import rclpy
from rclpy.node import Node
class MyPythonNode(Node):
    def __init__(self):
        super().__init__("my_node_name")
        self.get_logger().info("This node just says 'Hello'")
def main(args=None):
    rclpy.init(args=args)
    node = MyPythonNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
if __name__ == "__main__":
    main()
```

Add entry point to setup.py

```
from setuptools import setup
package_name = 'tlh_python_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='harman',
    maintainer_email='harman@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
```

Note test

```
    entry_points={
        'console_scripts': [
        'test = tlh_python_pkg.tlh_python_node:main'
        ],
    },
)
```

Add <buildtool_depend> and <depend>rclpy</depend> to package.xml

```
<?xml version="1.0"?>
<?xml-model
href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>tlh_python_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer
email="harman@todo.todo">harman</maintainer>
  <license>TODO: License declaration</license>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>
   <buildtool_depend>ament_python</buildtool_depend>
    <depend>rclpy</depend>
  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

harman@harman-VirtualBox:~/ros2_ws2/src/tlh_python_pkg$ **colcon build --packages-select tlh_python_pkg**
Starting >>> tlh_python_pkg
Finished <<< tlh_python_pkg [0.81s]

harman@harman-VirtualBox:~/ros2_ws2$ **ros2 run**
tlh_python_pkg test
No executable found

harman@harman-VirtualBox:~/ros2_ws2/src/tlh_python_pkg/tlh_python_pkg$ ls
__init__.py tlh_python_node.py
harman@harman-VirtualBox:~/ros2_ws2/src/tlh_python_pkg/tlh_python_pkg$ **chmod +x tlh_python_node.py**
harman@harman-VirtualBox:~/ros2_ws2/src/tlh_python_pkg/tlh_python_pkg$ ls -la
total 12
drwxrwxr-x 2 harman harman 4096 Nov 21 14:21 .
drwxrwxr-x 8 harman harman 4096 Nov 21 14:28 ..
-rw-rw-r-- 1 harman harman   0 Nov 21 13:31 __init__.py
-rwxrwxr-x 1 harman harman 371 Nov 21 14:21 tlh_python_node.py

This will let script run with just $ python3 <script> WE WANT PACKAGE!!

```
#!/usr/bin/env python
import rclpy
from rclpy.node import Node
class MyPythonNode(Node):
    def __init__(self):
        super().__init__("my_node_name")
        self.get_logger().info("This node just says 'Hello'")
def main(args=None):
    rclpy.init(args=args)
    node = MyPythonNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
if __name__ == "__main__":
    main()
```

RUN AS EXECUTABLE PYTHON

harman@harman-VirtualBox:~/ros2_ws2/src/tlh_python_pkg/tlh_python_pkg$ **python3 tlh_python_node.py**
[INFO] [1669064036.934436163] [my_node_name]: **This node just says 'Hello**'

RUN AS PYTHON SCRIPT IN A PACKAGE

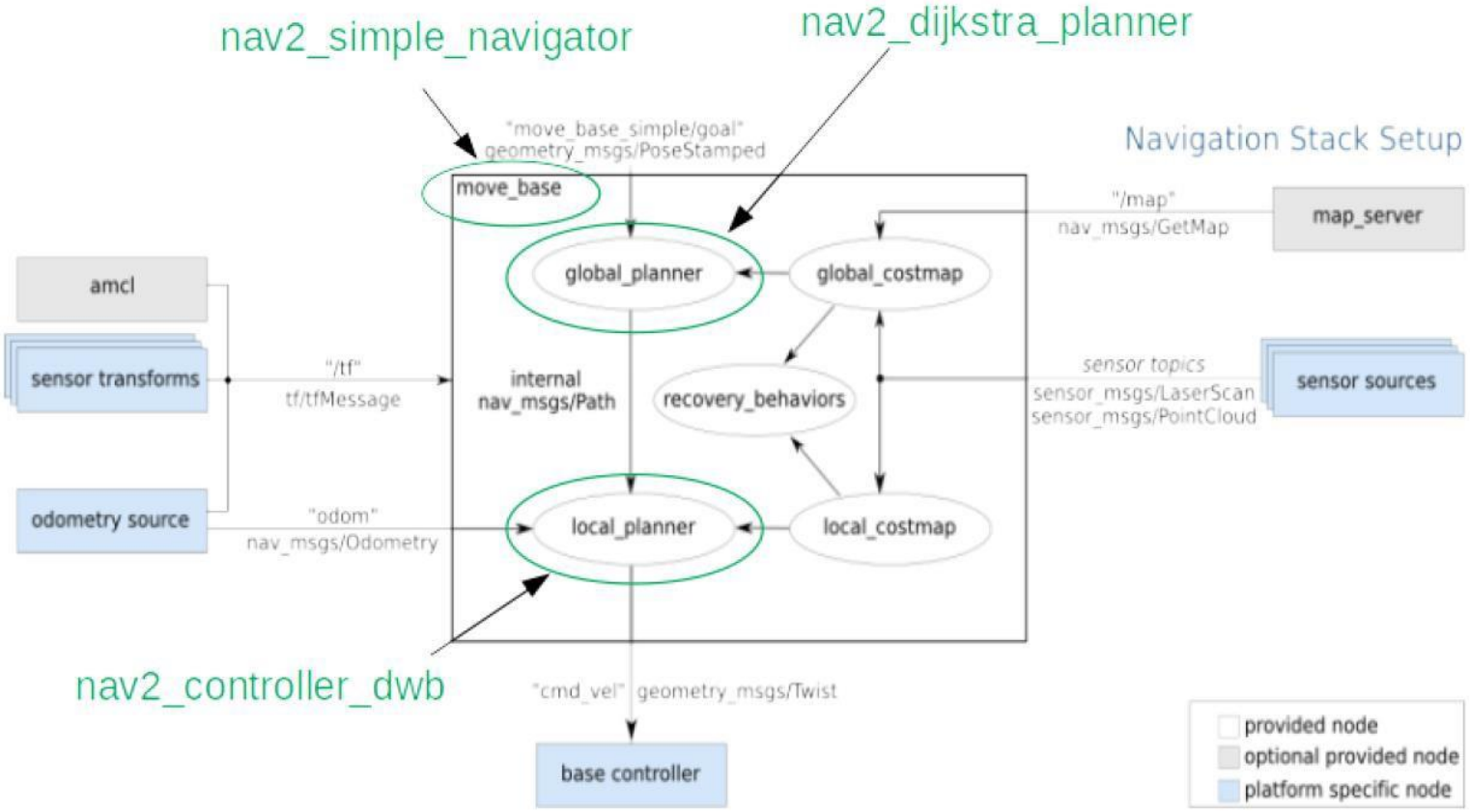
harman@harman-VirtualBox:~$ **cd ~/ros2_ws2**
harman@harman-VirtualBox:~/ros2_ws2$ **colcon build**
Starting >>> tlh_python_pkg
Finished <<< tlh_python_pkg [0.84s]

Summary: 1 package finished [1.00s]
harman@harman-VirtualBox:~/ros2_ws2$ **ros2 run tlh_python_pkg test**

[INFO] [1669064287.191807583] [my_node_name]: This node just says 'Hello'

ROS2 NAVIGATION

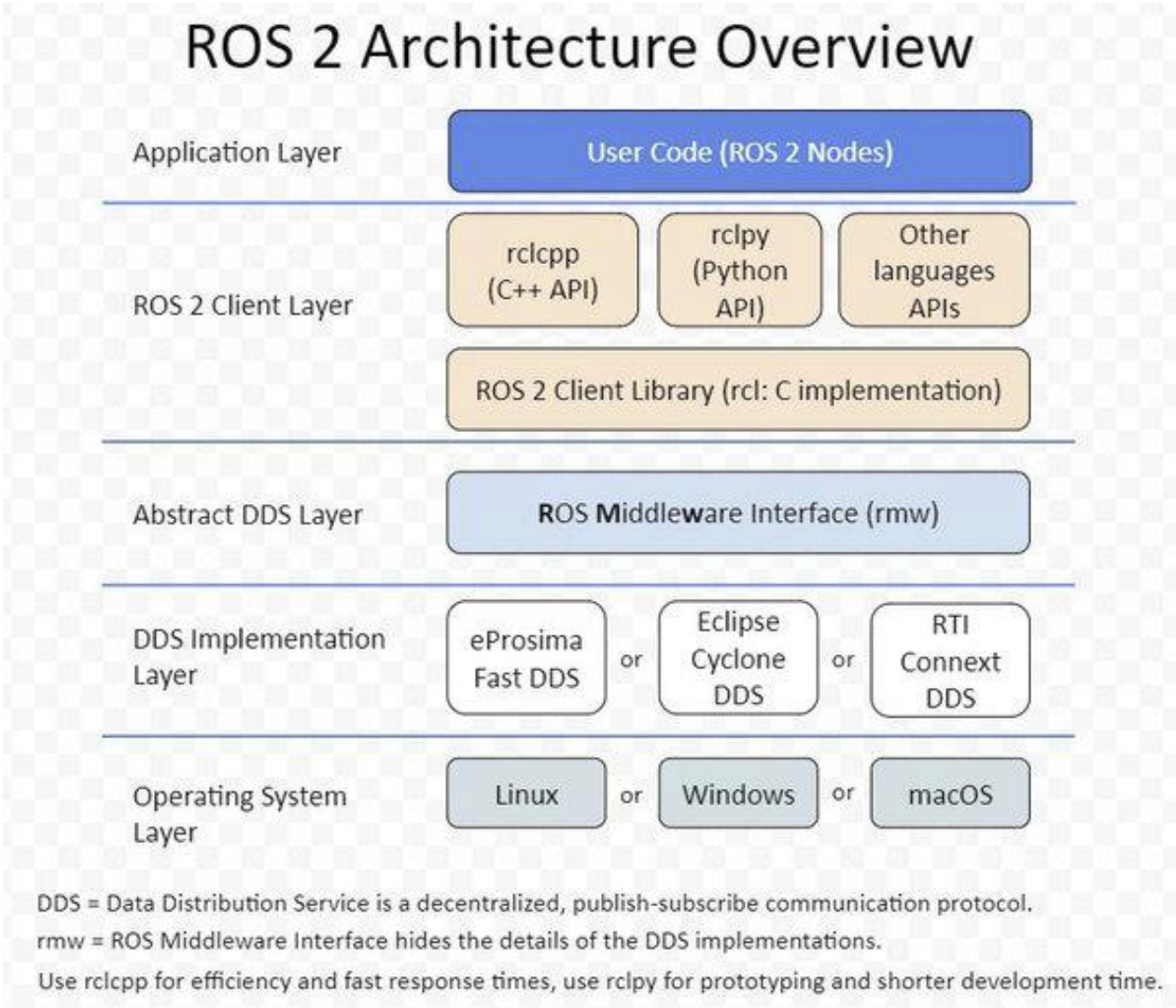
https://navigation.ros.org/about/ros1_comparison.html#ros1-comparison



Note: `nav2_simple_navigator` no longer exists, it has been replaced by `nav2_bt_navigator`.

https://automaticaddison.com/ros-2-architecture-overview/

COMMUNICATION DDS



MicroROS, combining advanced robotics and low-cost embedded systems

https://www.youtube.com/watch?v=aOktdgwbm_M

6,070 views Jan 15, 2022
(Brett Downing) ROS is the open-source robotics toolkit that has powered robots in academia for the last decade;
Micro-ROS allows nearly seamless integration of ROS tools into micro-controller frameworks including Arduino.
In this talk, Brett will demonstrate some of the capabilities of Micro-ROS, linking state-of-the-art software to some of the cheapest hardware on the marke

12,274 views Oct 8, 2020 1:03:13

DDS VIDEO

Learn how ROS 2 and DDS can be used together to develop autonomous vehicles / robots.

https://www.youtube.com/watch?v=GGqcrccWfeE&t=3358s

NEED HELP OR INFORMATION

5 _ROS_DATA&INFORMATION_F22.pdf

6_Ros1_ROS2FeaturesPowerPoint Presentation.pdf