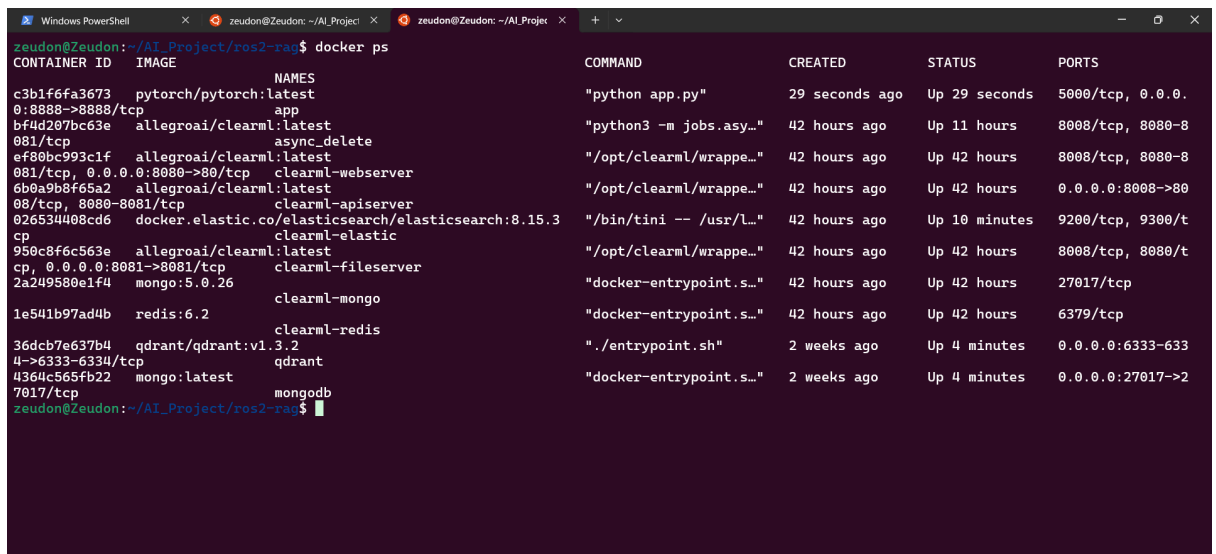


Team:

- 1) Nived Damodaran, Net ID: nd2743, GitHub username: Zeudon, HF username: Zeudon
- 2) Harsha Mupparaju, Net ID: sm12754, GitHub Username: harshamupparaju, HF username: neo2446

## 1. Environment and Tooling Milestone

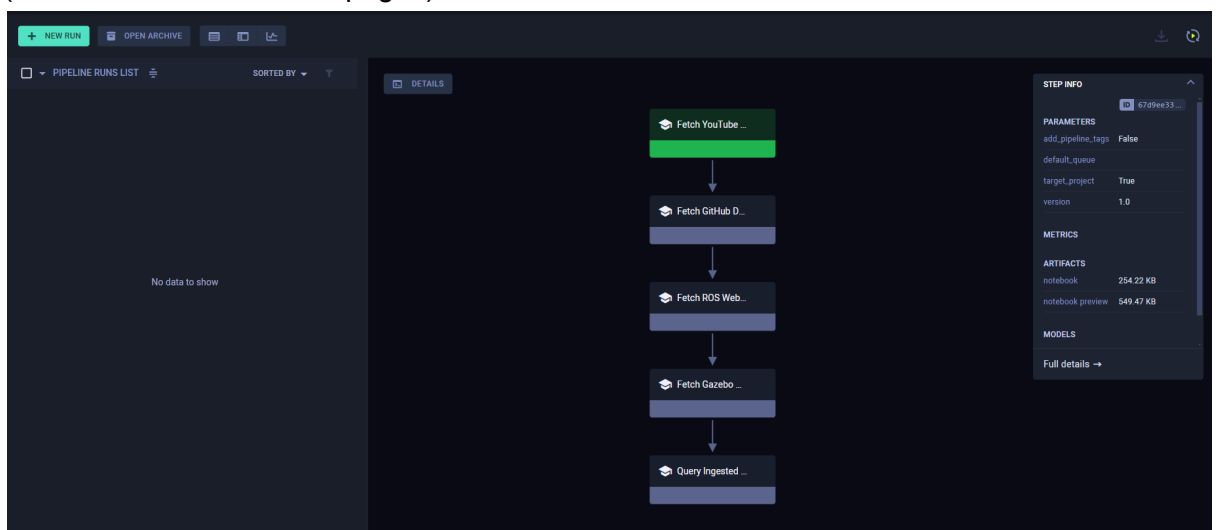
Attaching below the screenshot of the docker containers, There is 1 container each for MongoDB, QDrant and the App. The remaining containers are for clearML, for fileserver, API server and web host.

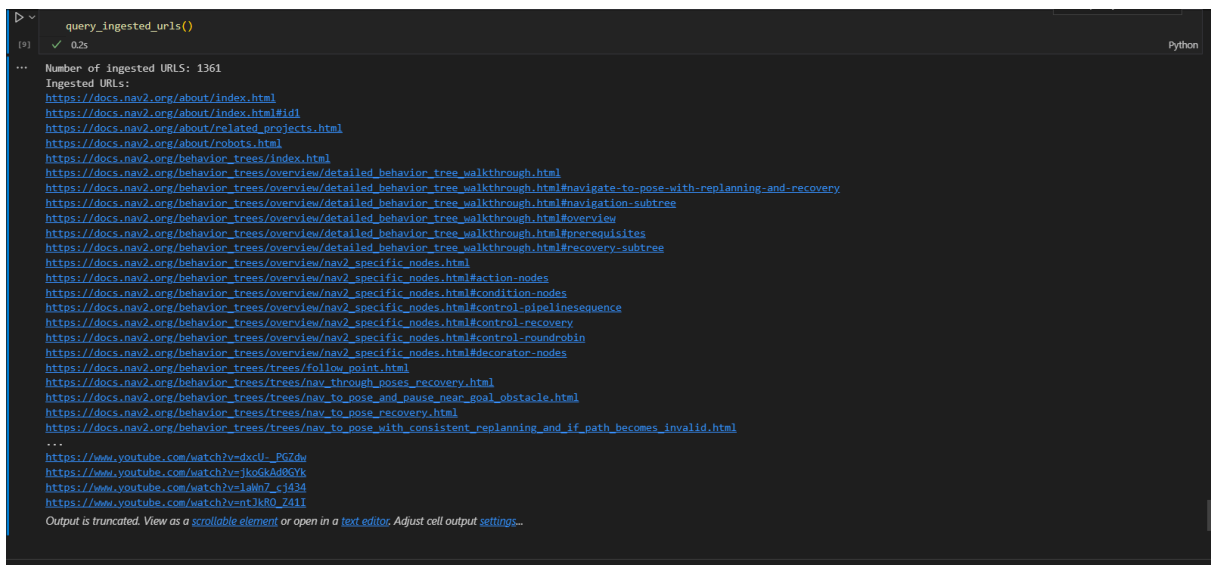
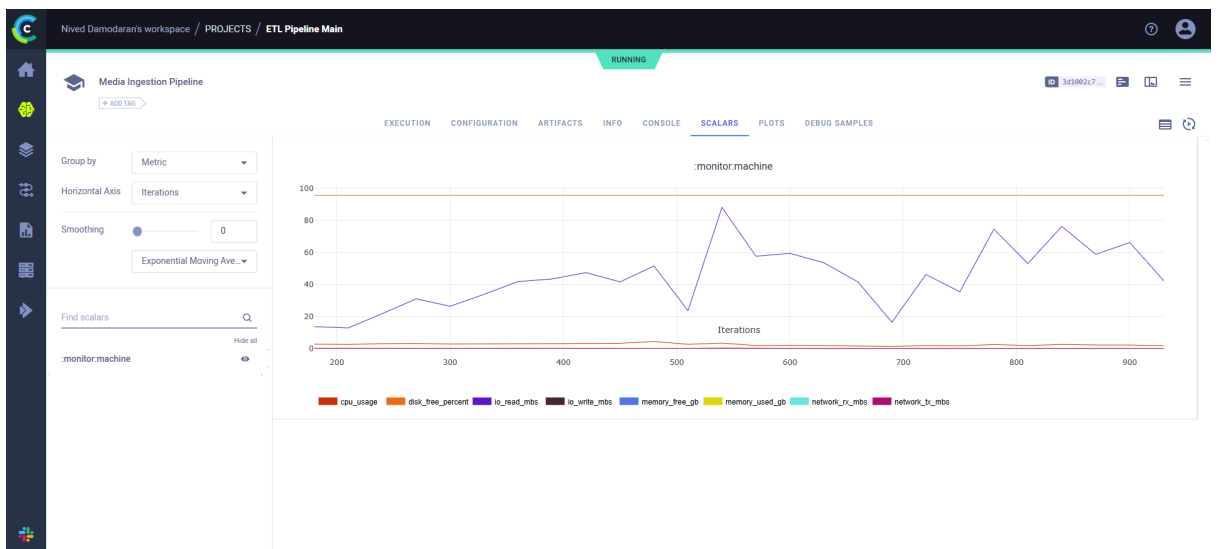
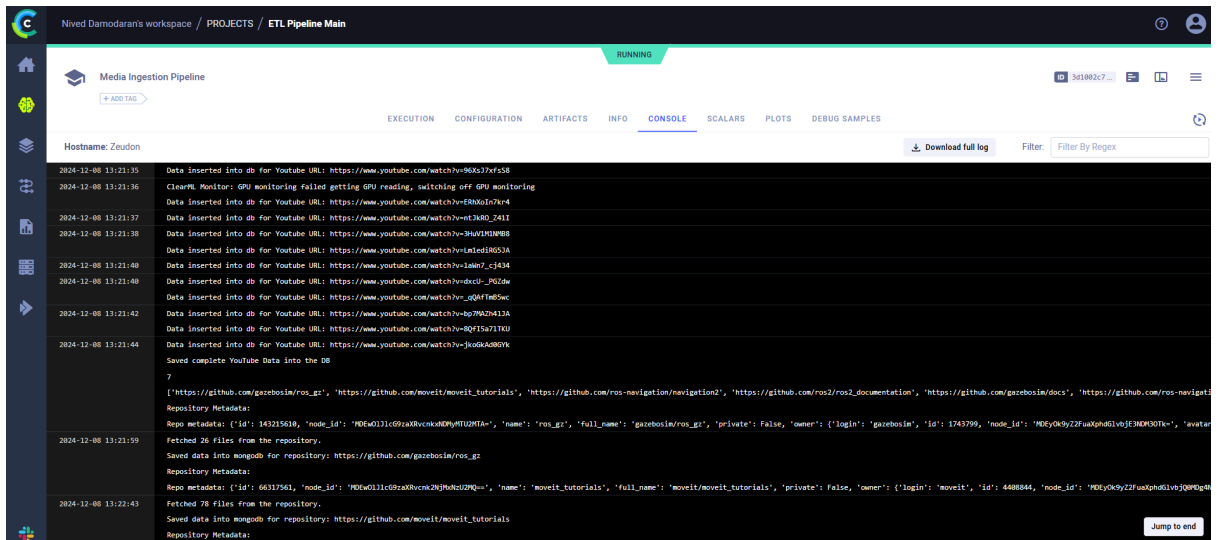


CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
c3b1ff6fa3673	pytorch/pytorch:latest	app	"python app.py"	29 seconds ago	Up 29 seconds	5000/tcp, 0.0.0.
0:8888->8888/tcp						
bfd4207bc63e	allegroai/clearml:latest	clearml-webserver	"python3 -m jobs.asy..."	42 hours ago	Up 11 hours	8008/tcp, 8080-8
081/tcp		clearml-apiserver				
ef80bc993c1f	allegroai/clearml:latest	clearml-fileserver	"/opt/clearml/wrappe..."	42 hours ago	Up 42 hours	8008/tcp, 8080-8
081/tcp, 0.0.0.0:8080->80/tcp						
6b0a9b9f65a2	allegroai/clearml:latest	clearml-mongo	"/opt/clearml/wrappe..."	42 hours ago	Up 42 hours	0.0.0.0:8080->80
08/tcp, 8080-8081/tcp						
026534408cd6	docker.elastic.co/elasticsearch/elasticsearch:8.15.3	clearml-elastic	"/bin/tini -- /usr/L..."	42 hours ago	Up 10 minutes	9200/tcp, 9300/t
cp						
950c8f6c563e	allegroai/clearml:latest	clearml-mongo	"/opt/clearml/wrappe..."	42 hours ago	Up 42 hours	8008/tcp, 8080/t
cp, 0.0.0.0:8081->8081/tcp						
2a249580e1f4	mongo:5.0.26	clearml-mongo	"docker-entrypoint.s..."	42 hours ago	Up 42 hours	27017/tcp
1e541b97ad4b	redis:6.2	clearml-redis	"docker-entrypoint.s..."	42 hours ago	Up 42 hours	6379/tcp
36dcb7e637b4	qdrant/qdrant:v1.3.2	qdrant	". /entrypoint.sh"	2 weeks ago	Up 4 minutes	0.0.0.0:6333-633
4->6333-6334/tcp						
4364c565fb22	mongo:latest	mongodb	"docker-entrypoint.s..."	2 weeks ago	Up 4 minutes	0.0.0.0:27017->2
7017/tcp						

## 2. ETL Pipeline

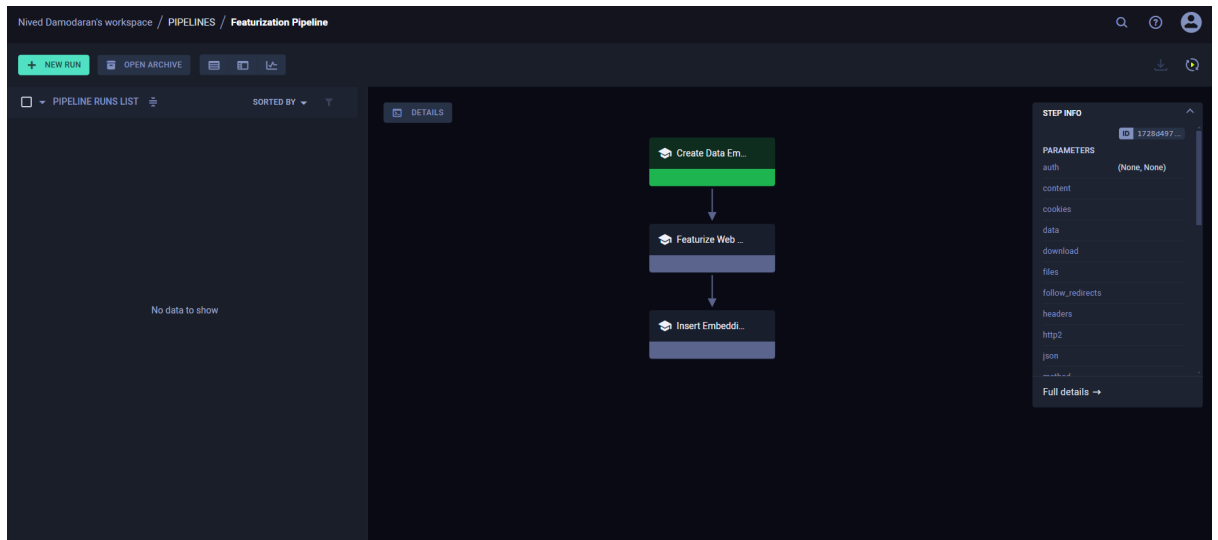
Attached below are screenshots of the ClearML dashboard and pipeline. The last image shows the output when we query all the unique URLs that was ingested (GitHub + Youtube + Web pages)





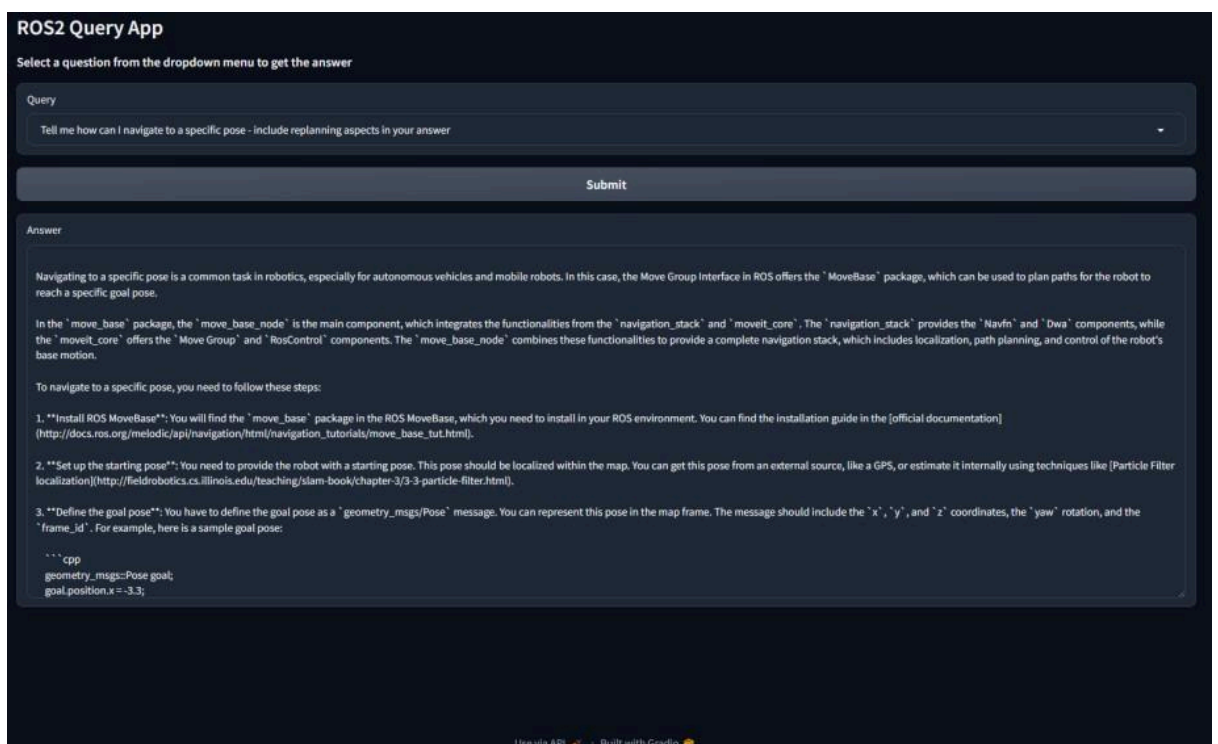
As seen from the image, we have scraped 1361 URLs, that include the unique youtube videos, github repository files and the web pages.

### 3. Featurization Pipeline



#### 4. Screenshots of Output

Question 1 - Tell me how can I navigate to a specific pose - include replanning aspects in your answer



## ROS2 Query App

Select a question from the dropdown menu to get the answer

Query

Tell me how can I navigate to a specific pose - include replanning aspects in your answer

Submit

Answer

3. **"Define the goal pose"**: You have to define the goal pose as a `geometry_msgs/Pose` message. You can represent this pose in the map frame. The message should include the `x`, `y`, and `z` coordinates, the `yaw` rotation, and the `frame_id`. For example, here is a sample goal pose:

```
'''cpp
geometry_msgs::Pose goal;
goal.position.x = -3.3;
goal.position.y = 2.6;
goal.position.z = 0.0;
goal.orientation.w = 1.0;
goal.header.frame_id = "map";
'''
```

4. **"Send the goal pose to the robot"**: Use the ROS service `move_base/MoveBase/clear_navigation_goals` to clear the current navigation goals and then use the `move_base/MoveBase/AddNavigationGoal` service to add the new goal.

5. **"Replanning"**: The `move_base` node replans by default every 1 second. If you want to change this, you can do it by launching the `move_base` node with a specific frequency, like this:

```
'''bash
roslaunch move_base move_base.launch frequency:=1.0
'''
```

You can also set the frequency in the launch file.

Use via API  - Built with Gradio 

## ROS2 Query App

Select a question from the dropdown menu to get the answer

Query

Tell me how can I navigate to a specific pose - include replanning aspects in your answer

Submit

Answer

4. **"Send the goal pose to the robot"**: Use the ROS service `move_base/MoveBase/clear_navigation_goals` to clear the current navigation goals and then use the `move_base/MoveBase/AddNavigationGoal` service to add the new goal.

5. **"Replanning"**: The `move_base` node replans by default every 1 second. If you want to change this, you can do it by launching the `move_base` node with a specific frequency, like this:

```
'''bash
roslaunch move_base move_base.launch frequency:=1.0
'''
```

You can also set the frequency in the launch file.



6. **"Replanning with a new path"**: You can also replan with a new path at any time by calling the `clear_navigation_goals` service.

In the Move Group interface, replanning happens at a specific rate (1 Hz by default). If you want to change this rate, you can do it by launching the `move_group` node with a specific frequency, like this:

```
'''bash
roslaunch move_group move_group.launch frequency:=1.0
'''
```

You can also set the frequency in the launch file.

To navigate to a specific pose, you need to provide the robot with a starting pose and a goal pose, and then use the ROS services and nodes to plan and execute the navigation.

Use via API  - Built with Gradio 

Question 2 - Can you provide me with code for this task?

## ROS2 Query App

Select a question from the dropdown menu to get the answer

Query

Can you provide me with code for this task? Task: Tell me how can I navigate to a specific pose

Submit

Answer

To navigate to a specific pose, you can use the Navigation stack in ROS. Here is an example code snippet in Python using the 'move\_group\_interface' to navigate to a specific pose.

```
''' python
from moveit_tutorials.tutorials.navigation_py import *

def main():
    # create a MoveGroupInterface object for the robot's left arm.
    # We will execute one operation, planning and then executing a move!
    planning_group = MoveGroupInterface(
        "left_arm", # robot_description, DH parameters, SRDF, collision_object, src
        "/move_groups_node", # node, planning_scene, move_server, planning_interface, moveit_tutorials
    )

    # Set our planning to use joint values as the method to specify the pose
    planning_group.set_planner_id("RRTstar")
    # Set the goal pose for the robot
    goal_pose = Pose(
        point=Point(x=1.0, y=1.0, z=1.0), # this is the position part
        orientation=Quaternion(
            roll=0,
            pitch=0,
```

## ROS2 Query App

Select a question from the dropdown menu to get the answer

Query

Can you provide me with code for this task? Task: Tell me how can I navigate to a specific pose



Submit

Answer

```
}

# Set our planning to use joint values as the method to specify the pose
planning_group.set_planner_id("RRTstar")
# Set the goal pose for the robot
goal_pose = Pose(
    point=Point(x=1.0, y=1.0, z=1.0), # this is the position part
    orientation=Quaternion(
        roll=0,
        pitch=0,
        yaw=0.7853981699425529, # this is the orientation part
    ),
)
# Execute the move.
planning_group.go(goal_pose)
rospy.sleep(10)
planning_group.stop()
planning_group.clear_pose_targets()

if __name__ == "__main__":
    main()
'''
```

Use via API  · Built with Gradio 

Question 3 - Tell me the difference between ROS1 and ROS2

### ROS2 Query App

Select a question from the dropdown menu to get the answer

Query

What are the main differences between ROS 1 and ROS 2?

Submit



Answer

ROSTwoMainDifferencesROS1ROS2  
ROS1ROS2Differences

ROS 1 and ROS 2 are both open-source robotics middleware projects. ROS 2 is the successor to ROS 1 and it's been designed to address the shortcomings of the previous version, as well as to take advantage of new technologies. Here are some of the main differences between ROS 1 and ROS 2:

- Underlying Technology: ROS 2 is built on top of the Data Distribution Service (DDS) standard, which is a middleware standard for real-time systems, whereas ROS 1 was built on top of the Robotic Operating System communication infrastructure (ROScomm). DDS has better performance, scalability, and portability compared to ROScomm.
- Language Support: ROS 2 supports multiple programming languages (C++, Python, Java, etc.) out of the box, whereas ROS 1 primarily focused on C++. This makes ROS 2 more accessible to developers who prefer other languages.
- Cross-Platform Capability: ROS 2 is more portable across different platforms compared to ROS 1. It can run on Linux, Windows, macOS, and embedded systems.
- Improved Performance: ROS 2 offers improved performance compared to ROS 1 due to the use of DDS. It has lower latency, higher throughput, and can handle more nodes and messages.
- Interoperability: ROS 2 is designed to be interoperable with other standards, such as OPC-UA, which is not the case with ROS 1.
- Quality of Service (QoS): ROS 2 offers more fine-grained control over Quality of Service features compared to ROS 1. It provides more options for reliability, latency, throughput, etc.
- Better Code Organization: ROS 2 has a better structure and organization of code compared to ROS 1. It uses the Ament tool for managing dependencies, building packages, and launching nodes.
- Security: ROS 2 has built-in security features that were not present in ROS 1. It allows for encryption, authentication, and access control between nodes, providing better security for the system.
- Rosbag: ROS 2 comes with Rosbag, a tool for recording and playing back ROS 2 messages. This is not available in ROS 1.

These are just some of the main differences between ROS 1 and ROS 2. ROS 2 is a major step forward in robotics middleware, providing better performance, more features, and better compatibility with other systems.

Use via API  · Built with Gradle 

Question 4 - What is the purpose of rqt\_graph and how can it help in debugging a ROS system?

### ROS2 Query App

Select a question from the dropdown menu to get the answer

Query

What is the purpose of rqt\_graph, and how can it help in debugging a ROS system?

Submit



Answer

rqt\_graph is a graphical introspection tool in ROS (Robot Operating System) that helps visualize the changing nodes and topics, as well as the connections between them. It provides a graphical representation of the ROS system, which can be helpful in debugging a ROS system.

rqt\_graph can assist in debugging a ROS system in several ways:

1. **"Identifying Issues"**: rqt\_graph allows you to visually inspect the relationships between nodes and topics in your ROS system. This can help you identify any abnormalities in your system, such as nodes that are not communicating as expected or topics that are not being published or subscribed to.
2. **"Debugging Complex Systems"**: In complex systems with many nodes and topics connected in various ways, rqt\_graph's highlighting feature can be very helpful. It allows you to easily focus on specific nodes or topics, making it easier to trace data flow and identify any issues.
3. **"Understanding System Behavior"**: By visualizing the system, you can gain a better understanding of how data is flowing through your ROS system. This can help you identify any bottlenecks, inefficiencies, or unexpected behavior in your system.
4. **"Visualizing Changes"**: rqt\_graph updates in real-time as the system changes. This allows you to monitor the effects of modifications to your system, such as adding or removing nodes, in real-time, helping you debug more efficiently.
5. **"Efficient Troubleshooting"**: With rqt\_graph, you can focus on the most relevant parts of your system during debugging, quickly drilling down to the root of the problem.

Overall, rqt\_graph provides a visual representation of your ROS system that can greatly assist in debugging. By enabling you to visualize the system, identify issues, and monitor changes in real-time, it can help you quickly diagnose and resolve problems in your ROS system.

Use via API  · Built with Gradle 

Question 5 - What is the function of rviz in ROS?

## ROS2 Query App

Select a question from the dropdown menu to get the answer

Query

What is the function of rviz in ROS?

Submit

Answer

ROS (Robot Operating System) is an open-source robot software framework that is primarily used for controlling and interacting with robots. RViz is a part of the ROS framework and is a graphical interface for displaying, browsing, and editing the ROS topics, services, and parameters.

The primary functions of RViz include:

1. Visualization: RViz provides a visual representation of the robot's environment. It can display point clouds, laser scans, 3D maps, and model objects. It is useful for visualizing the robot's position and orientation, as well as other sensor data.
2. Interactive Display: RViz allows you to interact with your robot's environment and sensors in real-time. You can select, filter, and change the visualization of your data. You can also change the view of your robot's environment, zoom in/out, and rotate the display.
3. Editing: RViz allows you to edit the ROS topics, services, and parameters. You can add, remove, and modify the data, which can be useful for debugging and testing.
4. Configuration: RViz provides a user interface for configuring the ROS parameters and visualizing the robot's configuration.
5. Simulation: RViz can display the data from the ROS simulations, which makes it useful for debugging and testing the robot's behavior in a simulated environment.

In summary, RViz is an essential tool for visualizing, interacting with, and configuring your robot's environment and sensors within the ROS framework. It helps developers and researchers understand and control the robot's behavior.