ROS 2 QoS Lab: Step-by-Step Instructions

This guide covers how to create the ROS 2 package qos_demo on a Raspberry Pi 4 (Bookworm) running ROS 2 Jazzy.

1. Environment Setup

Open a terminal on the Raspberry Pi and ensure ROS 2 is sourced.

```
source /opt/ros/jazzy/setup.bash
```

2. Create the Workspace & Package

If you don't have a workspace yet, create one. Then create the Python package with the necessary dependencies.

```
# 1. Create workspace directories
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src

# 2. Create the package 'qos_demo'
# Dependencies: rclpy (Python client), sensor_msgs (Image msg), cv_bridge (OpenCV helper)
ros2 pkg create --build-type ament_python qos_demo --dependencies rclpy sensor_msgs
cv_bridge
```

3. Create the Node Scripts

Navigate to the package folder where Python scripts live.

```
cd ~/ros2_ws/src/qos_demo/qos_demo
```

A. Create the Publisher (cam_pub.py)

Create the file:

```
touch cam_pub.py
```

Open it (e.g., nano cam_pub.py or using your editor) and paste the **Publisher Code** provided in the lecture/previous steps.

B. Create the Subscriber (cam_sub.py)

Create the file:

```
touch cam_sub.py
```

Open it and paste the **Subscriber Code** provided in the lecture/previous steps.

4. Configure setup.py (Crucial Step)

You must tell ROS 2 that these scripts are executable nodes.

1. Navigate back one folder to find setup.py.
   ```
   cd ~/ros2_ws/src/qos_demo
   ```
2. Open setup.py and modify the entry_points section. It should look exactly like this:

```
entry_points={
    'console_scripts': [
        'cam_pub = qos_demo.cam_pub:main',
        'cam_sub = qos_demo.cam_sub:main',
    ],
},
```

5. Build the Package

Now compile the workspace to install the new nodes.

```
# 1. Go to the workspace root
cd ~/ros2_ws

# 2. Build the package
colcon build --packages-select qos_demo

# 3. Source the install environment (This adds your new node to the path)
source install/setup.bash
```

6. Running the Nodes

The nodes require command-line arguments to set their QoS policy.

Syntax: ros2 run qos_demo <node_name> <reliability> <durability>

Experiment A: Standard Streaming (Best Effort)

Use this for low-latency video (UDP-like).

Publisher (Instructor/Sender):

```
ros2 run qos_demo cam_pub best_effort volatile
```

Subscriber (Student/Receiver):

```
ros2 run qos_demo cam_sub best_effort volatile
```

Experiment B: Reliable Streaming (TCP-like)

Use this to demonstrate lag/buffering when the network is busy.

Publisher:

```
ros2 run qos_demo cam_pub reliable volatile
```

Subscriber:

```
ros2 run qos_demo cam_sub reliable volatile
```

Experiment C: Late Joining (Transient Local)

Use this to show a new subscriber receiving "old" data immediately upon joining.

Publisher:

```
ros2 run qos_demo cam_pub reliable transient
```

(Wait 10 seconds...)

Subscriber:

```
ros2 run qos_demo cam_sub reliable transient
```