Micro-Ros Setup :

micro-ROS provides two ways of building a micro-ROS application for embedded platforms:

1. **micro_ros_setup:** micro_ros_setup provides a standalone build system in the form of a ROS 2 package for use in any normal ROS 2 workspace. This tool is available in the micro-ROS/micro_ros_setup repository. Click here to learn more .
2. **Platform-specific integrations:** Developers have integrated micro-ROS with several platforms build tools. Click here to learn more.

Note: We are opting for a platform-specific integration,since this approach is simpler.

micro-ROS for Arduino :

The micro-ROS for Arduino support package is a special port of micro-ROS provided as a set of precompiled libraries for specific platforms. The main reason for this approach is that Arduino does not allow the build of a complex library such as micro-ROS, so by using this approach a ready-to-use solution is provided to the Arduino users. You can explore the **micro_ros_arduino** repository.

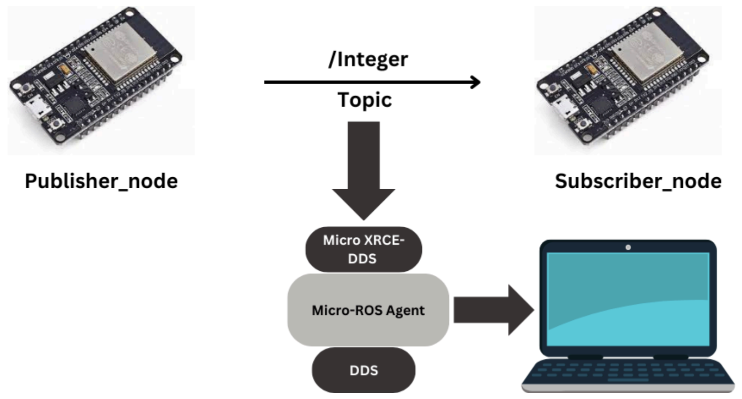
Now, we are moving to set up the **micro_ros_arduino** package in your Arduino IDE.

- **Step_1 :** Download the zip file from the **micro_ros_arduino** repository.
- **Step_2 :** Open Arduino IDE and include the downloaded zip file as library.
- **Step_3 :** After successfully including it, you will be able to see examples of **micro_ros_arduino** in the examples section of the Arduino IDE.

Programming with rcl and rclc:

The rclc packages offer convenient functions for easily programming a ROS application in the C programming language. As we will be utilizing the micro-ROS Arduino library. This library is specifically designed for baremetal projects and is intended to be used with the Arduino IDE .To simplify the integration process, the library is provided in a precompiled version.

The programming interface is demonstrated by a simple example: **micro_ros_publisher_wifi.ino** it shall publish a message every second, and a **micro_ros_subscriber_wifi.ino** to print out this message. You can download both nodes in this **link**.



The above image shows the block diagram of the example we are going to do.

1. Publisher node

When you look at the example of **micro_ros_publisher_wifi.ino** we will see the instances of structures used from rclc.

```
rcl_publisher_t publisher;
std_msgs__msg__Int32 msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;
```

Further more, memory allocation is organized by an allocator and the ROS context is initialized by using a support object.

`set_microros_wifi_transports` function is used to configure Micro-ROS to use specific WiFi transports for communication.

```
set_microros_wifi_transports("WIFI SSID", "WIFI PASS", "laptop_ip_address", 8888);
```

A node is created by below functions.

```
  allocator = rcl_get_default_allocator();

 //create init_options
   RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

 // create node
   RCCHECK(rclc_node_init_default(&node, "micro_ros_arduino_wifi_node", "", &support));
```

A publisher is created with three parameters: node object, message type, and topic name. In this example, a Int32 publisher with topic name /Integer is created.

```
  // create publisher
  RCCHECK(rclc_publisher_init_best_effort(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "/Integer"));
```

To publish this message, a timer is setup to call rcl_publish every second:

```
void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
  RCLC_UNUSED(last_call_time);
  if (timer != NULL) {
    RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
    msg.data++;
  }
}
```

2. Subscriber node

The subscriber node will be the same as the publisher node, but with some changes. Subscriber instances are called instead of the publisher

```
rcl_subscription_t subscriber;
```

Since a subscriber needs a callback, it has also been defined:

```
void subscription_callback(const void * msgin)
{
  const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;

}
```

The following command is used to initiate the subscriber:

```
 RCCHECK(rclc_subscription_init_default(
    &subscriber,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "topic_name"));
```

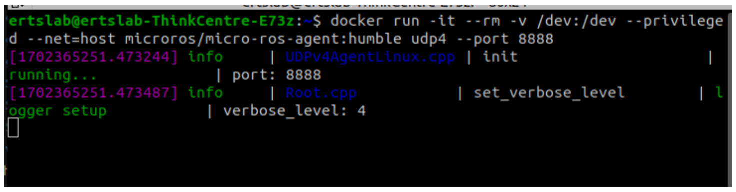
Now, upload the **micro-ros_publisher_wifi.ino** code to one ESP32.And **micro-ros_subcriber_wifi.ino** to another Esp32.

Micro-ROS Agent :

As explained in Part 1 of let's Micro-ROS ,the Agent bridges between DDS-XRCE and normal DDS. This implies that it is a crucial tool for facilitating communication between ROS 2 and microcontrollers. Therefore, you have to use it every time you intend to communicate between the microcontroller and ROS 2 nodes. To run the microros agent you have to run the following command in your system.

```
docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:humble
```

Note: Before running the above command, ensure that your system and ESP32 are connected to the same hotspot. It is recommended to use the personal hotspot feature from your mobile phone.



```
ertslab@ertslab-ThinkCentre-E73z:~$ docker run -it --rm -v /dev:/dev --privilege
d --net=host microros/micro-ros-agent:humble udp4 --port 8888
[1702365251.473244] info     | UDPv4AgentLinux.cpp | init                   |
running...             | port: 8888
[1702365251.473487] info     | Root.cpp             | set_verbose_level     | l
ogger setup            | verbose_level: 4
```

You will be able to see this output. Press the EN button on the ESP32, and you will see that it has been connected with your system.

If any error occurs or the agent is not connecting with Esp32 ,try to change the port number both in code and in agent.

Open new terminal and type below command

```
ros2 topic list
```

If you are able to see the topic `/Integer` it means your system is connected with an ESP32. Data has been published from one ESP32 and subscribed to by another. With the help of the Micro-ROS agent, ROS2 can view this data in your system.

You can explore more examples, such as publishing data from a ROS2 node in your system and subscribing to it with an ESP32 controller.