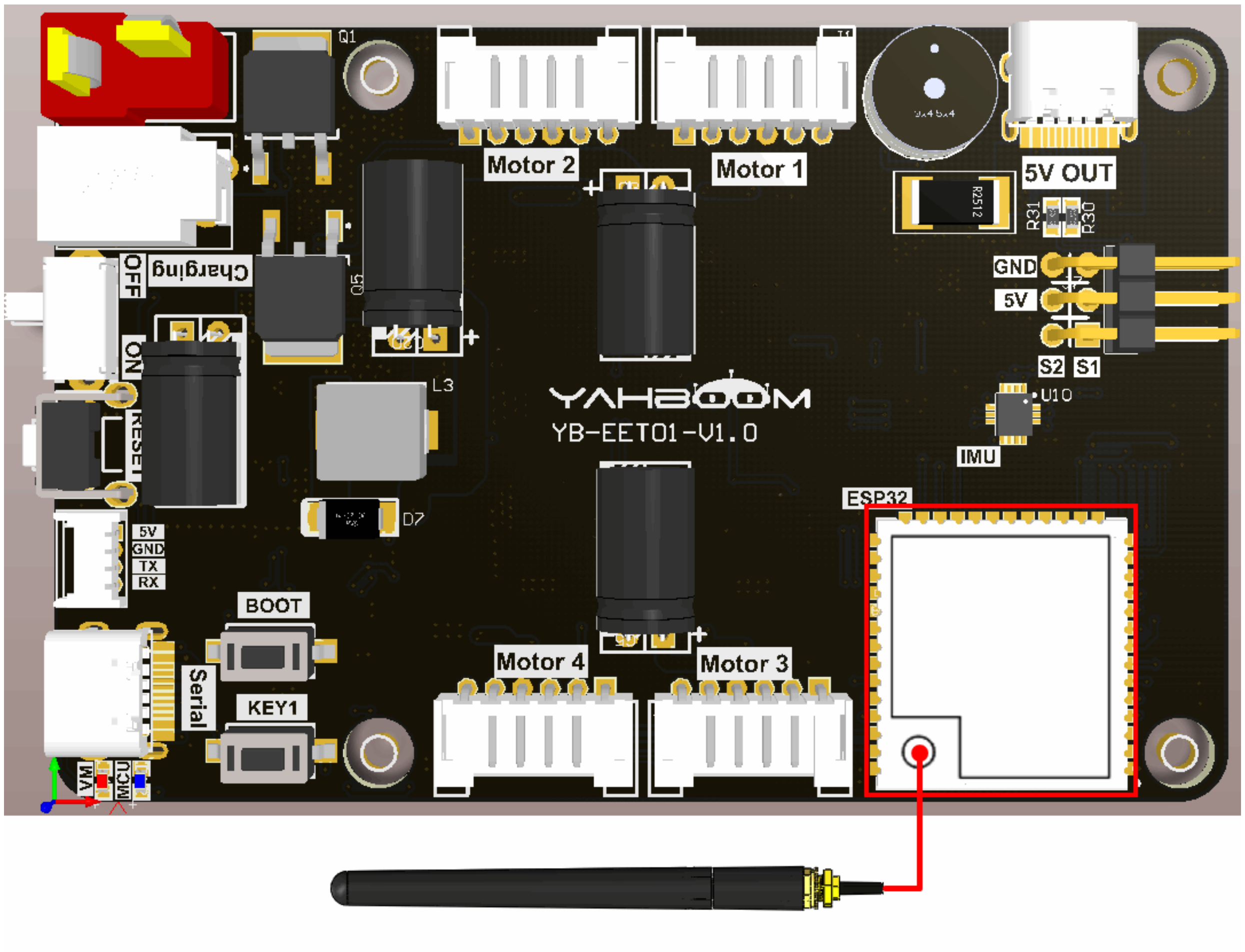
Multi-topic subscribe and publish

1. Experimental purpose

Learn ESP32-microROS components, access the ROS2 environment, subscribe to and publish multiple int32 topics.

2. Hardware connection

As shown in the figure below, the microROS control board integrates the ESP32-S3-WROOM core module, which has its own wireless WiFi function. The ESP32-S3 core module needs to be connected to an antenna, and a type-C data cable needs to be connected to the computer and the microROS control board as Burn firmware function.



3. Core code analysis

The virtual machine path corresponding to the program source code is as follows

```
~/esp/Samples/microros_samples/publisher_subscriber
```

First, get the WiFi name and password to connect from the IDF configuration tool.

```
#define ESP_WIFI_SSID      CONFIG_ESP_WIFI_SSID
#define ESP_WIFI_PASS      CONFIG_ESP_WIFI_PASSWORD
#define ESP_MAXIMUM_RETRY  CONFIG_ESP_MAXIMUM_RETRY
```

The uros_network_interface_initialize function will connect to WiFi hotspots based on the WiFi configuration in IDF.

```
ESP_ERROR_CHECK(uros_network_interface_initialize());
```

Then obtain ROS_NAMESPACE, ROS_DOMAIN_ID, ROS_AGENT_IP and ROS_AGENT_PORT from the IDF configuration tool.

```
#define ROS_NAMESPACE      CONFIG_MICRO_ROS_NAMESPACE
#define ROS_DOMAIN_ID      CONFIG_MICRO_ROS_DOMAIN_ID
#define ROS_AGENT_IP       CONFIG_MICRO_ROS_AGENT_IP
#define ROS_AGENT_PORT     CONFIG_MICRO_ROS_AGENT_PORT
```

Initialize the configuration of microROS, in which ROS_DOMAIN_ID, ROS_AGENT_IP and ROS_AGENT_PORT are modified in the IDF configuration tool according to actual needs.

```
    rcl_allocator_t allocator = rcl_get_default_allocator();
    rclc_support_t support;

    // 创建rcl初始化选项
    // Create init_options.
    rcl_init_options_t init_options = rcl_get_zero_initialized_init_options();
    RCCHECK(rcl_init_options_init(&init_options, allocator));
    // 修改ROS域ID
    // change ros domain id
    RCCHECK(rcl_init_options_set_domain_id(&init_options, ROS_DOMAIN_ID));

    // 初始化rmw选项
    // Initialize the rmw options
    rmw_init_options_t *rmw_options =
rcl_init_options_get_rmw_init_options(&init_options);

    // 设置静态代理IP和端口
    // Setup static agent IP and port
    RCCHECK(rmw_uros_options_set_udp_address(ROS_AGENT_IP, ROS_AGENT_PORT,
rmw_options));
```

Try to connect to the proxy. If the connection is successful, go to the next step. If the connection to the proxy is unsuccessful, you will always be in the connected state.

```
    while (1)
    {
        ESP_LOGI(TAG, "Connecting agent: %s:%s", ROS_AGENT_IP, ROS_AGENT_PORT);
        state_agent = rclc_support_init_with_options(&support, 0, NULL,
&init_options, &allocator);
        if (state_agent == ESP_OK)
        {
            ESP_LOGI(TAG, "Connected agent: %s:%s", ROS_AGENT_IP, ROS_AGENT_PORT);
            break;
        }
        vTaskDelay(pdMS_TO_TICKS(500));
    }
```

Create the node "esp32_pub_sub", in which ROS_NAMESPACE is empty by default and can be modified in the IDF configuration tool according to actual conditions.

```
    rcl_node_t node;
    RCCHECK(rclc_node_init_default(&node, "esp32_pub_sub", ROS_NAMESPACE,
&support));
```

Create three publishers "publisher_1", "publisher_2" and "publisher_3", and specify the ROS topic information as std_msgs/msg/Int32 type.

```
    RCCHECK(rclc_publisher_init_default(
        &publisher_1,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "publisher_1"));
    RCCHECK(rclc_publisher_init_default(
        &publisher_2,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "publisher_2"));
    RCCHECK(rclc_publisher_init_default(
        &publisher_3,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "publisher_3"));
```

Create three subscribers "subscriber_1", "subscriber_2" and "subscriber_3", and specify the ROS topic information as std_msgs/msg/Int32 type.

```
    RCCHECK(rclc_subscription_init_default(
        &subscriber_1,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "subscriber_1"));
    RCCHECK(rclc_subscription_init_default(
        &subscriber_2,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "subscriber_2"));
    RCCHECK(rclc_subscription_init_default(
        &subscriber_3,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "subscriber_3"));
```

// Set the publishing frequency of the three publishers to 10HZ, or you can set the publishing frequency separately.

```
    const unsigned int timer1_timeout = 100;
    const unsigned int timer2_timeout = 100;
    const unsigned int timer3_timeout = 100;
    RCCHECK(rclc_timer_init_default(
        &timer_1,
        &support,
        RCL_MS_TO_NS(timer1_timeout),
        timer1_callback));

    RCCHECK(rclc_timer_init_default(
        &timer_2,
        &support,
        RCL_MS_TO_NS(timer2_timeout),
        timer2_callback));

    RCCHECK(rclc_timer_init_default(
        &timer_3,
        &support,
        RCL_MS_TO_NS(timer3_timeout),
        timer3_callback));
```

Add publisher timers and subscribers to the executor, where the handle_num parameter of the executor represents the number added to the executor.

```
    rclc_executor_t executor;
    int handle_num = 6;
    RCCHECK(rclc_executor_init(&executor, &support.context, handle_num,
&allocator));

    RCCHECK(rclc_executor_add_timer(&executor, &timer_1));
    RCCHECK(rclc_executor_add_timer(&executor, &timer_2));
    RCCHECK(rclc_executor_add_timer(&executor, &timer_3));

    RCCHECK(rclc_executor_add_subscription(&executor, &subscriber_1, &msg_sub1,
&subscription_1_callback, ON_NEW_DATA));
    RCCHECK(rclc_executor_add_subscription(&executor, &subscriber_2, &msg_sub2,
&subscription_2_callback, ON_NEW_DATA));
    RCCHECK(rclc_executor_add_subscription(&executor, &subscriber_3, &msg_sub3,
&subscription_3_callback, ON_NEW_DATA));
```

The timer callback function contents of the three publishers publish topic data.

```
void timer1_callback(rcl_timer_t *timer, int64_t last_call_time)
{
    RCLC_UNUSED(last_call_time);
    if (timer != NULL)
    {
        RCSOFTCHECK(rcl_publish(&publisher_1, &msg_pub1, NULL));
        msg_pub1.data++;
    }
}
void timer2_callback(rcl_timer_t *timer, int64_t last_call_time)
{
    RCLC_UNUSED(last_call_time);
    if (timer != NULL)
    {
        RCSOFTCHECK(rcl_publish(&publisher_2, &msg_pub2, NULL));
        msg_pub2.data++;
    }
}
void timer3_callback(rcl_timer_t *timer, int64_t last_call_time)
{
    RCLC_UNUSED(last_call_time);
    if (timer != NULL)
    {
        RCSOFTCHECK(rcl_publish(&publisher_3, &msg_pub3, NULL));
        msg_pub3.data++;
    }
}
```

When microros subscribers receive topic data, the subscriber callback function is triggered and the received data is printed out.

```
void subscription_1_callback(const void * msgin)
```

```
{
    const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;
    printf("Sub1 Received: %d\n",  (int)  msg->data);
}
void subscription_2_callback(const void * msgin)
{
    const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;
    printf("Sub2 Received: %d\n",  (int)  msg->data);
}
void subscription_3_callback(const void * msgin)
{
    const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;
    printf("Sub3 Received: %d\n",  (int)  msg->data);
}
```

Call rclc_executor_spin_some in the loop to make microros work normally.

```
    while (1)
    {
        rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
        usleep(1000);
    }
```

4. Compile, download and flash firmware

Use a Type-C data cable to connect the virtual machine/computer and the microROS control board. If the system pops up, choose to connect to the virtual machine.

Activate the ESP-IDF development environment. Note that every time you open a new terminal, you need to activate the ESP-IDF development environment before compiling the firmware.

```
source ~/esp/esp-idf/export.sh
```

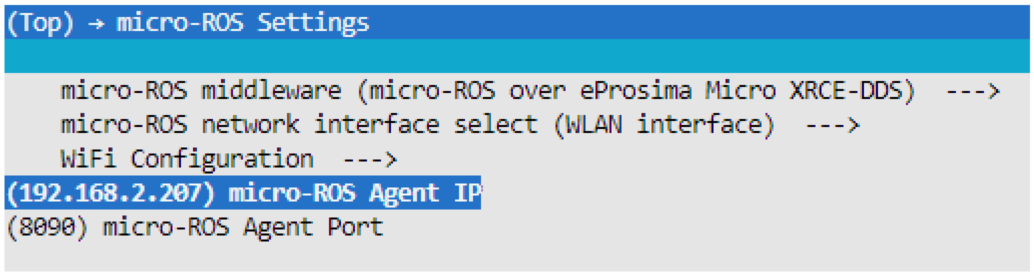
Enter the project directory

```
cd ~/esp/Samples/microros_samples/publisher_subscriber
```

Open the ESP-IDF configuration tool.

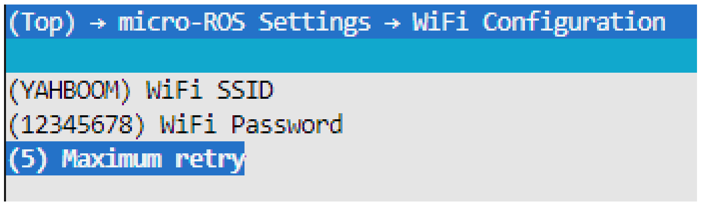
```
idf.py menuconfig
```

Open micro-ROS Settings, fill in the IP address of the agent host in micro-ROS Agent IP, and fill in the port number of the agent host in micro-ROS Agent Port.



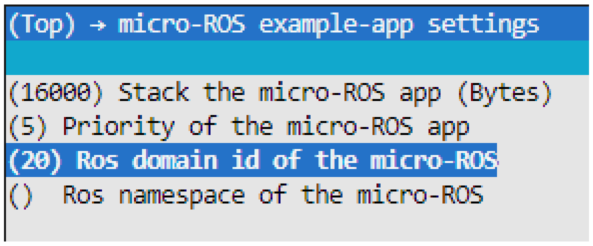
```
(Top) → micro-ROS Settings

    micro-ROS middleware (micro-ROS over eProsima Micro XRCE-DDS)  --->
    micro-ROS network interface select (WLAN interface)  --->
    WiFi Configuration  --->
(192.168.2.207) micro-ROS Agent IP
(8090) micro-ROS Agent Port
```

Open micro-ROS Settings->WiFi Configuration in sequence, and fill in your own WiFi name and password in the WiFi SSID and WiFi Password fields.



```
(Top) → micro-ROS Settings → WiFi Configuration

(YAHBOOM) WiFi SSID
(12345678) WiFi Password
(5) Maximum retry
```

Open the micro-ROS example-app settings. The Ros domain id of the micro-ROS defaults to 20. If multiple users are using it at the same time in the LAN, the parameters can be modified to avoid conflicts. Ros namespace of the micro-ROS is empty by default and does not need to be modified under normal circumstances. If non-empty characters (within 10 characters) are modified, the namespace parameter will be added before the node and topic.



```
(Top) → micro-ROS example-app settings

(16000) Stack the micro-ROS app (Bytes)
(5) Priority of the micro-ROS app
(20) Ros domain id of the micro-ROS
()  Ros namespace of the micro-ROS
```

After modification, press S to save, and then press Q to exit the configuration tool.

Compile, burn, and open the serial port simulator.

```
idf.py build flash monitor
```

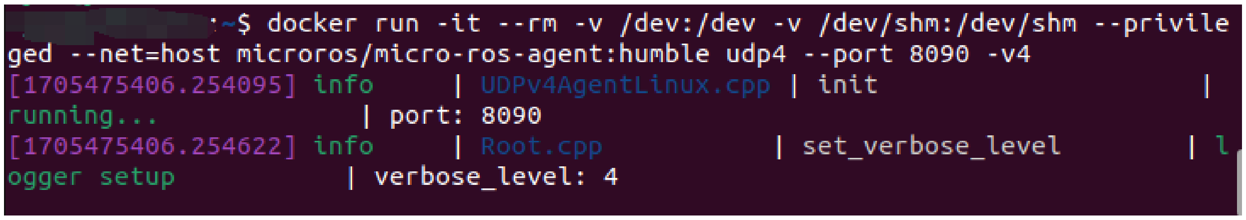
If you need to exit the serial port simulator, press **Ctrl+]**.

5. Experimental results

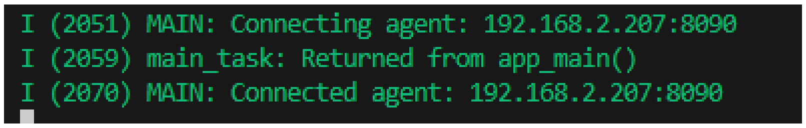
After powering on, ESP32 tries to connect to the WiFi hotspot, and then tries to connect to the proxy IP and port.

If the agent is not turned on in the virtual machine/computer terminal, please enter the following command to turn on the agent. If the agent is already started, there is no need to start the agent again.

```
docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --net=host
microros/micro-ros-agent:humble udp4 --port 8090 -v4
```

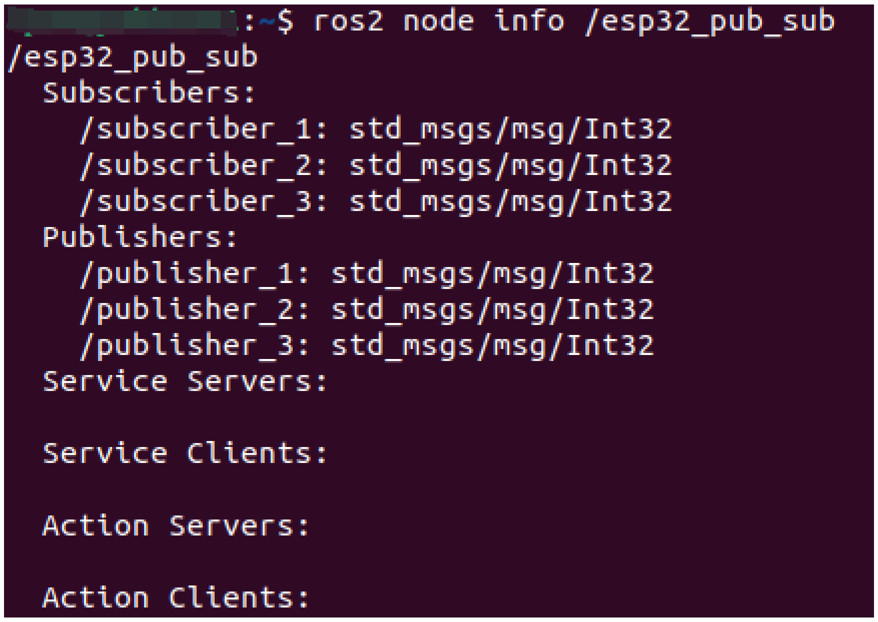

```
[redacted]:~$ docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privile
ged --net=host microros/micro-ros-agent:humble udp4 --port 8090 -v4
[1705475406.254095] info     | UDPv4AgentLinux.cpp | init                  |
running...             | port: 8090
[1705475406.254622] info     | Root.cpp            | set_verbose_level     | l
ogger setup            | verbose_level: 4
```

After the connection is successful, a node and a subscriber are created.


```
I (2051) MAIN: Connecting agent: 192.168.2.207:8090
I (2059) main_task: Returned from app_main()
I (2070) MAIN: Connected agent: 192.168.2.207:8090
```

At this time, you can open another terminal in the virtual machine/computer and view the /esp32_pub_sub node.

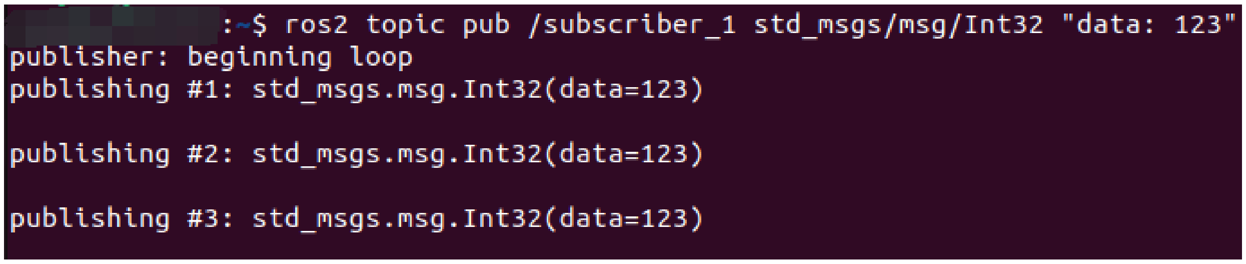
```
ros2 node list
ros2 node info /esp32_pub_sub
```


```
[redacted]:~$ ros2 node info /esp32_pub_sub
/esp32_pub_sub
  Subscribers:
    /subscriber_1: std_msgs/msg/Int32
    /subscriber_2: std_msgs/msg/Int32
    /subscriber_3: std_msgs/msg/Int32
  Publishers:
    /publisher_1: std_msgs/msg/Int32
    /publisher_2: std_msgs/msg/Int32
    /publisher_3: std_msgs/msg/Int32
  Service Servers:

  Service Clients:

  Action Servers:

  Action Clients:
```

Publish information with int data 123 to the /subscriber_1 topic.

```
ros2 topic pub /subscriber_1 std_msgs/msg/Int32 "data: 123"
```

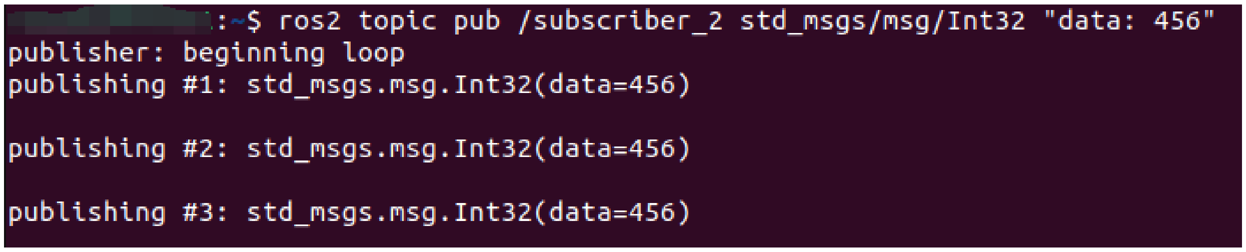
Press Ctrl+C to end the command


```
[redacted]:~$ ros2 topic pub /subscriber_1 std_msgs/msg/Int32 "data: 123"
publisher: beginning loop
publishing #1: std_msgs.msg.Int32(data=123)

publishing #2: std_msgs.msg.Int32(data=123)

publishing #3: std_msgs.msg.Int32(data=123)
```

Publish information with int data 456 to the /subscriber_2 topic.

```
ros2 topic pub /subscriber_2 std_msgs/msg/Int32 "data: 456"
```

Press Ctrl+C to end the command

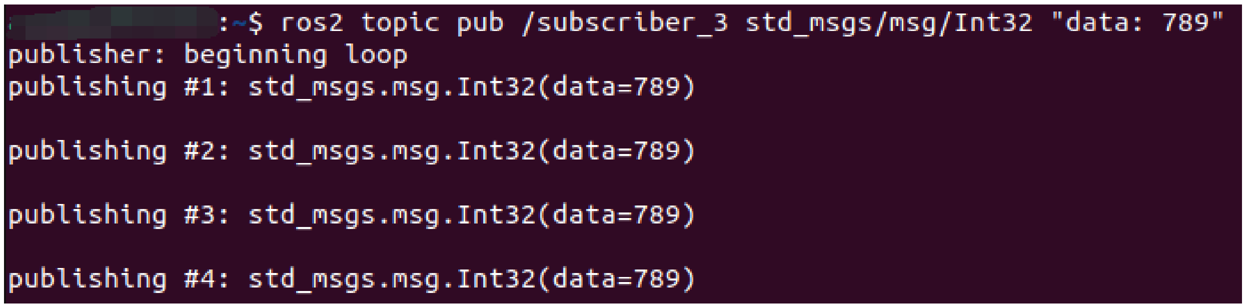


```
[redacted]:~$ ros2 topic pub /subscriber_2 std_msgs/msg/Int32 "data: 456"
publisher: beginning loop
publishing #1: std_msgs.msg.Int32(data=456)

publishing #2: std_msgs.msg.Int32(data=456)

publishing #3: std_msgs.msg.Int32(data=456)
```

Publish information with int data 789 to the /subscriber_3 topic.

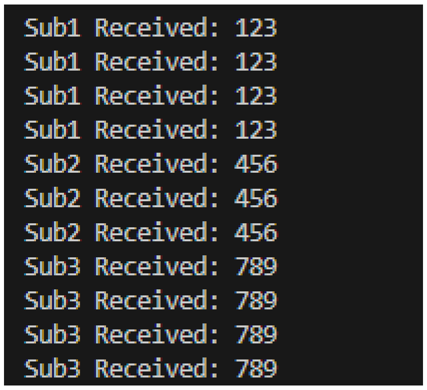
```
ros2 topic pub /subscriber_3 std_msgs/msg/Int32 "data: 789"
```

Press Ctrl+C to end the command



```
[redacted]:~$ ros2 topic pub /subscriber_3 std_msgs/msg/Int32 "data: 789"
publisher: beginning loop
publishing #1: std_msgs.msg.Int32(data=789)

publishing #2: std_msgs.msg.Int32(data=789)

publishing #3: std_msgs.msg.Int32(data=789)

publishing #4: std_msgs.msg.Int32(data=789)
```

You can see the corresponding information printed on the serial port simulator, indicating that the subscription is successful.

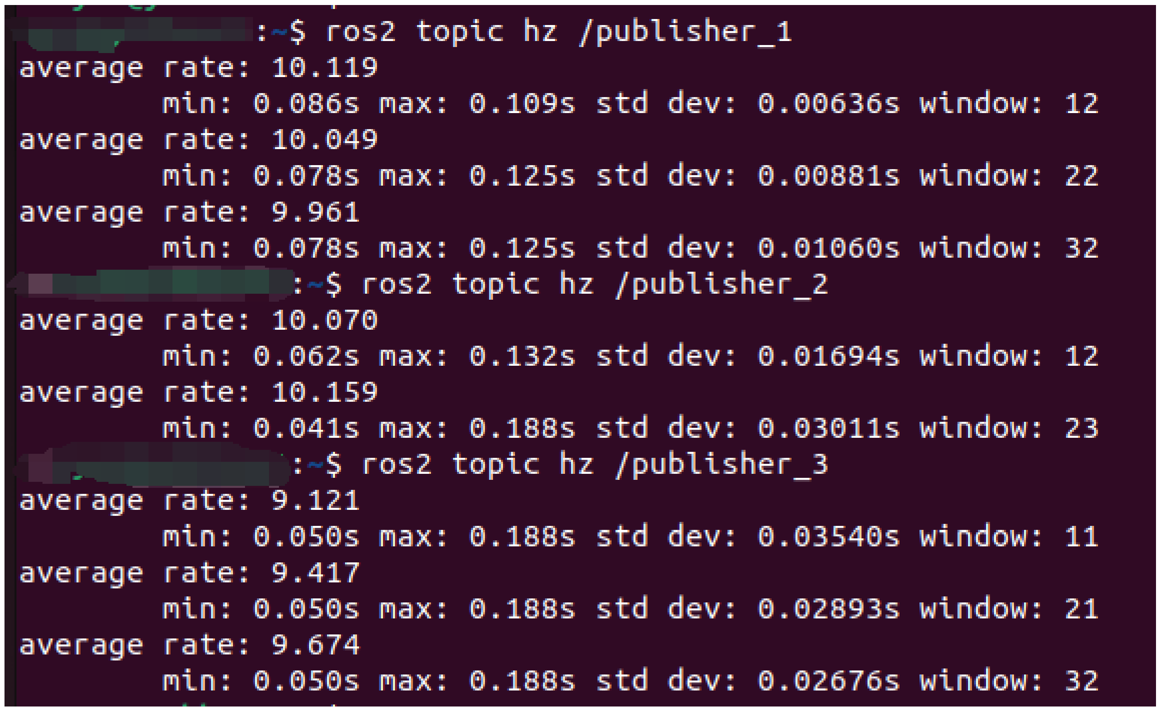


```
Sub1 Received: 123
Sub1 Received: 123
Sub1 Received: 123
Sub1 Received: 123
Sub2 Received: 456
Sub2 Received: 456
Sub2 Received: 456
Sub3 Received: 789
Sub3 Received: 789
Sub3 Received: 789
Sub3 Received: 789
```

View the frequency of /publisher_1, /publisher_2, and /publisher_3 topics

```
ros2 topic hz /publisher_1
ros2 topic hz /publisher_2
ros2 topic hz /publisher_3
```

Press Ctrl+C to end the command.

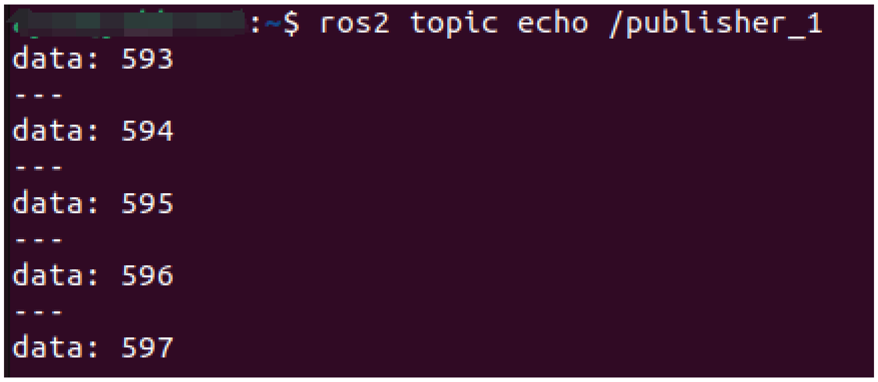


```
[redacted]:~$ ros2 topic hz /publisher_1
average rate: 10.119
	min: 0.086s max: 0.109s std dev: 0.00636s window: 12
average rate: 10.049
	min: 0.078s max: 0.125s std dev: 0.00881s window: 22
average rate: 9.961
	min: 0.078s max: 0.125s std dev: 0.01060s window: 32
[redacted]:~$ ros2 topic hz /publisher_2
average rate: 10.070
	min: 0.062s max: 0.132s std dev: 0.01694s window: 12
average rate: 10.159
	min: 0.041s max: 0.188s std dev: 0.03011s window: 23
[redacted]:~$ ros2 topic hz /publisher_3
average rate: 9.121
	min: 0.050s max: 0.188s std dev: 0.03540s window: 11
average rate: 9.417
	min: 0.050s max: 0.188s std dev: 0.02893s window: 21
average rate: 9.674
	min: 0.050s max: 0.188s std dev: 0.02676s window: 32
```

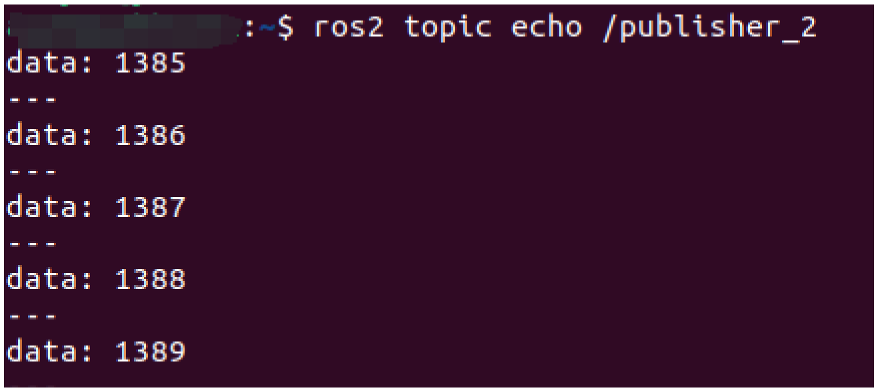
Subscribe to data for /publisher_1, /publisher_2 and /publisher_3 topics

```
ros2 topic echo /publisher_1
ros2 topic echo /publisher_2
ros2 topic echo /publisher_3
```

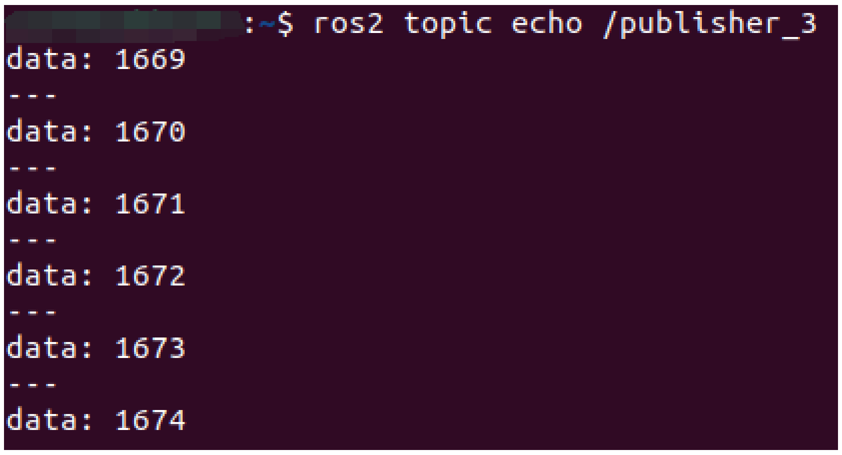
Press Ctrl+C to end the command.



```
[redacted]:~$ ros2 topic echo /publisher_1
data: 593
---
data: 594
---
data: 595
---
data: 596
---
data: 597
```



```
[redacted]:~$ ros2 topic echo /publisher_2
data: 1385
---
data: 1386
---
data: 1387
---
data: 1388
---
data: 1389
```



```
[redacted]:~$ ros2 topic echo /publisher_3
data: 1669
---
data: 1670
---
data: 1671
---
data: 1672
---
data: 1673
---
data: 1674
```