

# 16.ROS2 distributed communication

---

## 16.ROS2 distributed communication

1. Concept
2. Implementation
  - 2.1. Default realize
  - 2.2. Distributed network grouping
3. Note
4. Calculation rules for DDS domain ID values

## 1. Concept

---

Multi-machine communication, or distributed communication, refers to a communication strategy that can realize data interaction between different hosts through the network.

ROS2 itself is a distributed communication framework that can easily realize communication between different devices. The middleware that ROS2 is based on is DDS. When in the same network, distributed communication can be achieved through the domain ID mechanism (ROS\_DOMAIN\_ID) of DDS. The general process is: before starting the node, you can set the value of the domain ID. If different nodes have the same domain IDs, they can be discovered and communicated freely. On the contrary, if the domain ID values are different, this cannot be achieved. By default, the domain ID used when all nodes are started is 0. In other words, as long as they are on the same network, you do not need to do any configuration, and different nodes on different ROS2 devices can achieve distributed communication.

The application scenarios of distributed communication are relatively wide, such as unmanned vehicle formation, UAV formation, remote control, etc. The interaction of these data all relies on distributed communication.

## 2. Implementation

---

### 2.1. Default realize

As long as the master and slave machines [there can be multiple] are in the same network, distributed communication has been achieved. For example, the master and slave are connected to the same WiFi or the same router.

In Windows, set the network of the virtual machine to [Bridge Mode] and it will be on the same network as the host.

Testing:

1. Host side [car] executes:

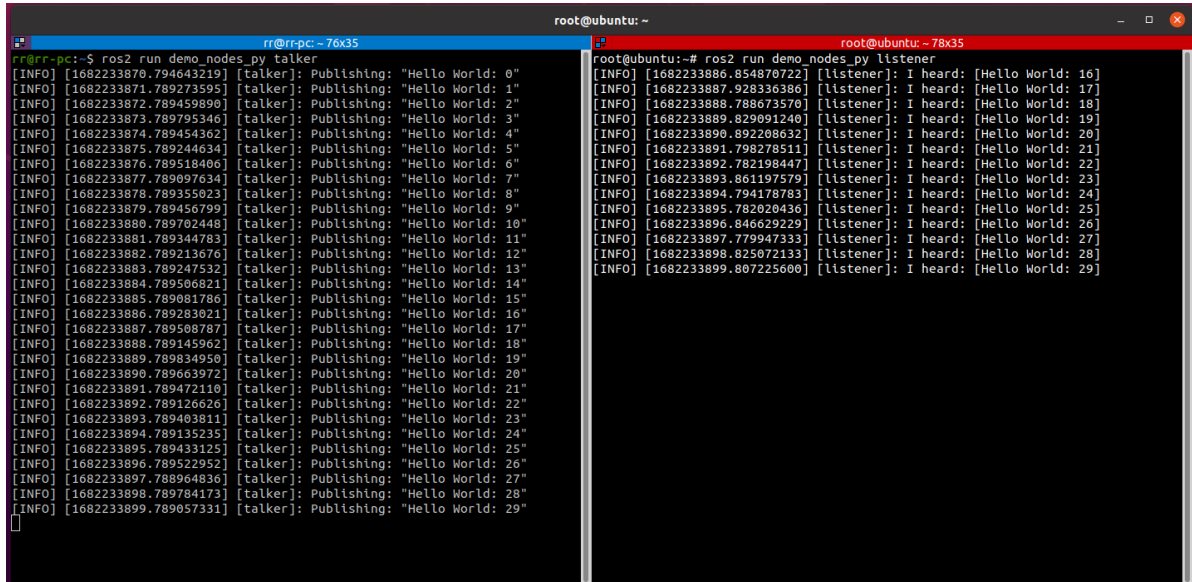
What is demonstrated here is that the car is in docker. The network mode used by docker is the host mode. The host mode simply shares the same network with the car, so it is no different from executing on the car.

```
ros2 run demo_nodes_py talker
```

2. At the same time, execute from the machine side [virtual machine]:

```
ros2 run demo_nodes_py listener
```

If the display is as follows: The topics published by the host side can be subscribed to by the slave side in time, indicating that multi-machine communication has been achieved.



```
rrrpc-76x35:~$ ros2 run demo_nodes_py talker
[INFO] [1682233870.794643219] [talker]: Publishing: "Hello World: 0"
[INFO] [1682233871.789273595] [talker]: Publishing: "Hello World: 1"
[INFO] [1682233872.789459890] [talker]: Publishing: "Hello World: 2"
[INFO] [1682233873.789795346] [talker]: Publishing: "Hello World: 3"
[INFO] [1682233874.789454362] [talker]: Publishing: "Hello World: 4"
[INFO] [1682233875.789244634] [talker]: Publishing: "Hello World: 5"
[INFO] [1682233876.789518406] [talker]: Publishing: "Hello World: 6"
[INFO] [1682233877.789097634] [talker]: Publishing: "Hello World: 7"
[INFO] [1682233878.789355023] [talker]: Publishing: "Hello World: 8"
[INFO] [1682233879.789456799] [talker]: Publishing: "Hello World: 9"
[INFO] [1682233880.789702448] [talker]: Publishing: "Hello World: 10"
[INFO] [1682233881.789344783] [talker]: Publishing: "Hello World: 11"
[INFO] [1682233882.789213676] [talker]: Publishing: "Hello World: 12"
[INFO] [1682233883.789247532] [talker]: Publishing: "Hello World: 13"
[INFO] [1682233884.789506521] [talker]: Publishing: "Hello World: 14"
[INFO] [1682233885.789081786] [talker]: Publishing: "Hello World: 15"
[INFO] [1682233886.789283021] [talker]: Publishing: "Hello World: 16"
[INFO] [1682233887.789508787] [talker]: Publishing: "Hello World: 17"
[INFO] [1682233888.789145962] [talker]: Publishing: "Hello World: 18"
[INFO] [1682233889.789834950] [talker]: Publishing: "Hello World: 19"
[INFO] [1682233890.789663972] [talker]: Publishing: "Hello World: 20"
[INFO] [1682233891.789472110] [talker]: Publishing: "Hello World: 21"
[INFO] [1682233892.789126626] [talker]: Publishing: "Hello World: 22"
[INFO] [1682233893.789403811] [talker]: Publishing: "Hello World: 23"
[INFO] [1682233894.789135235] [talker]: Publishing: "Hello World: 24"
[INFO] [1682233895.789433125] [talker]: Publishing: "Hello World: 25"
[INFO] [1682233896.789522952] [talker]: Publishing: "Hello World: 26"
[INFO] [1682233897.788964836] [talker]: Publishing: "Hello World: 27"
[INFO] [1682233898.789784173] [talker]: Publishing: "Hello World: 28"
[INFO] [1682233899.789057331] [talker]: Publishing: "Hello World: 29"

root@ubuntu:~$ ros2 run demo_nodes_py listener
[INFO] [1682233886.854870722] [listener]: I heard: [Hello World: 16]
[INFO] [1682233887.928336386] [listener]: I heard: [Hello World: 17]
[INFO] [1682233888.788673570] [listener]: I heard: [Hello World: 18]
[INFO] [1682233889.829091240] [listener]: I heard: [Hello World: 19]
[INFO] [1682233890.892208632] [listener]: I heard: [Hello World: 20]
[INFO] [1682233891.798278511] [listener]: I heard: [Hello World: 21]
[INFO] [1682233892.782198447] [listener]: I heard: [Hello World: 22]
[INFO] [1682233893.861197579] [listener]: I heard: [Hello World: 23]
[INFO] [1682233894.794178783] [listener]: I heard: [Hello World: 24]
[INFO] [1682233895.782020436] [listener]: I heard: [Hello World: 25]
[INFO] [1682233896.846629229] [listener]: I heard: [Hello World: 26]
[INFO] [1682233897.779947333] [listener]: I heard: [Hello World: 27]
[INFO] [1682233898.825072133] [listener]: I heard: [Hello World: 28]
[INFO] [1682233899.807225600] [listener]: I heard: [Hello World: 29]
```

## 2.2. Distributed network grouping

Assume that there are other robots in use in the network you are currently on. In order to avoid interference from other robots, you can also set a group for your robot.

ROS2 provides a DOMAIN mechanism, which is similar to grouping. Only computers in the same DOMAIN can communicate. We can add such a configuration to the .bashrc of the host [car] and slave [virtual machine]. Assign both to a group:

```
$ export ROS_DOMAIN_ID=<your_domain_id>
```

If the IDs assigned to the host side [car] and the slave side [virtual machine] are different, the two cannot communicate and achieve the purpose of grouping.

Testing:

1. Host side [car] executes:

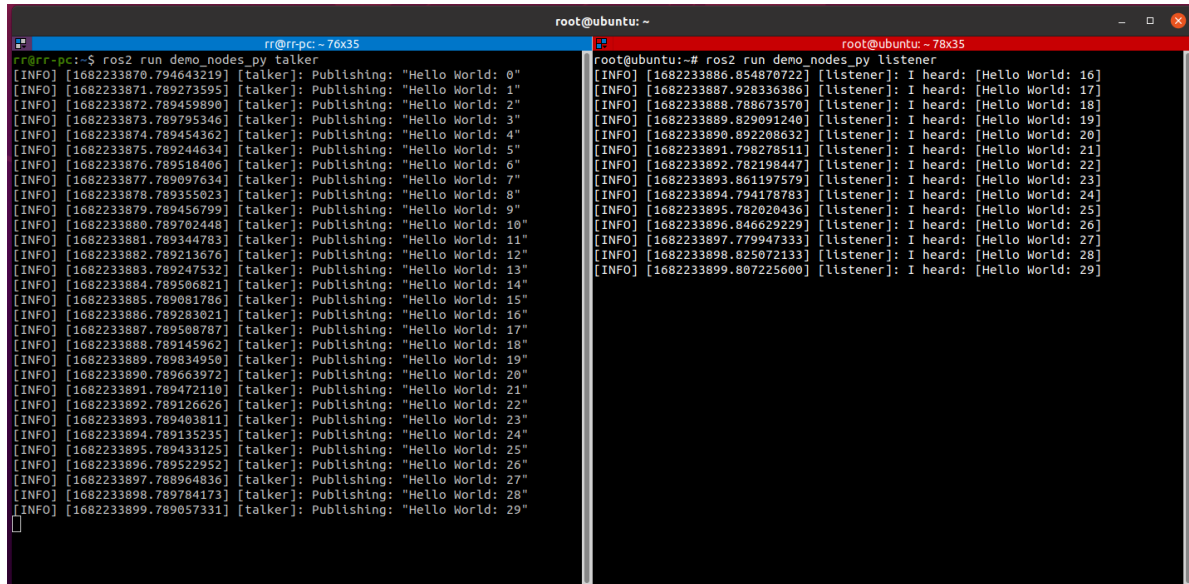
What is demonstrated here is that the car is in docker. The network mode used by docker is the host mode. The host mode simply shares the same network with the car, so it is no different from executing on the car.

```
echo "export ROS_DOMAIN_ID=6" >> ~/.bashrc # The 6 here is ROS_DOMAIN_ID, you
don't have to use 6, just comply with the rules of ROS_DOMAIN_ID
source ~/.bashrc
ros2 run demo_nodes_py talker
```

2. At the same time, execute from the machine side [virtual machine]:

```
echo "export ROS_DOMAIN_ID=6" >> ~/.bashrc # This is consistent with the value
on the host side.
source ~/.bashrc
ros2 run demo_nodes_py listener
```

If the display is as follows: The topic published by the host side can be subscribed to by the slave side in time, which means that grouped multi-machine communication has been implemented.



The image shows two terminal windows side-by-side. The left window is titled 'rr@rr-pc: ~ 76x35' and shows the output of 'ros2 run demo\_nodes\_py talker'. It displays a series of 'Hello World' messages from a 'talker' node, with timestamps and topic names. The right window is titled 'root@ubuntu: ~ 78x35' and shows the output of 'ros2 run demo\_nodes\_py listener'. It displays a series of 'Hello World' messages received by a 'listener' node, with timestamps and topic names. The messages in both windows are synchronized, indicating successful communication between the two machines.

### 3. Note

Setting the value of ROS\_DOMAIN\_ID is not arbitrary, and there are certain constraints:

1. It is recommended that the value of ROS\_DOMAIN\_ID is between [0,101], including 0 and 101;
2. The total number of nodes in each domain ID is limited and needs to be less than or equal to 120;
3. If the domain ID is 101, then the total number of nodes in the domain needs to be less than or equal to 54.

### 4. Calculation rules for DDS domain ID values

The relevant calculation rules for domain ID values are as follows:

1. DDS is based on TCP/IP or UDP/IP network communication protocol. Network communication requires specifying a port number. The port number is represented by a 2-byte unsigned integer, and its value range is between [0,65535] ;
2. The allocation of port numbers also has its rules and cannot be used arbitrarily. According to the DDS protocol, 7400 is used as the starting port, that is, the available ports are [7400, 65535]. It is also known that according to the DDS protocol, by default, Each domain ID occupies 250 ports, so the number of domain IDs is:  $(65535-7400)/250 = 232$  (number), and the corresponding value range is [0,231];

3. The operating system will also set some reserved ports. When using ports in DDS, you need to avoid these reserved ports to avoid conflicts during use. The reserved ports of different operating systems are different.

The end result is that the available domain IDs are [0,101] and [215-231] under Linux and [0,166] in Windows and Mac,

In summary, in order to be compatible with multiple platforms, it is recommended that the domain ID be in the range of [0,101].

4. Each domain ID occupies 250 ports by default, and each ROS2 node needs to occupy two ports.

In addition, according to the DDS protocol, within the port segment of each domain ID, the first and second ports are the Discovery Multicast port and User Multicast port, and the 11th and 12th ports starting from the Discovery Unicast port and User Unicast port of the first node in the domain ,

The ports occupied by subsequent nodes are postponed in sequence, so the maximum number of nodes in a domain ID is:  $(250-10)/2 = 120$  (number);

5. Special case: When the domain ID value is 101, the second half of the ports belong to the reserved ports of the operating system, and the maximum number of nodes is 54.

Just understand the above calculation rules.