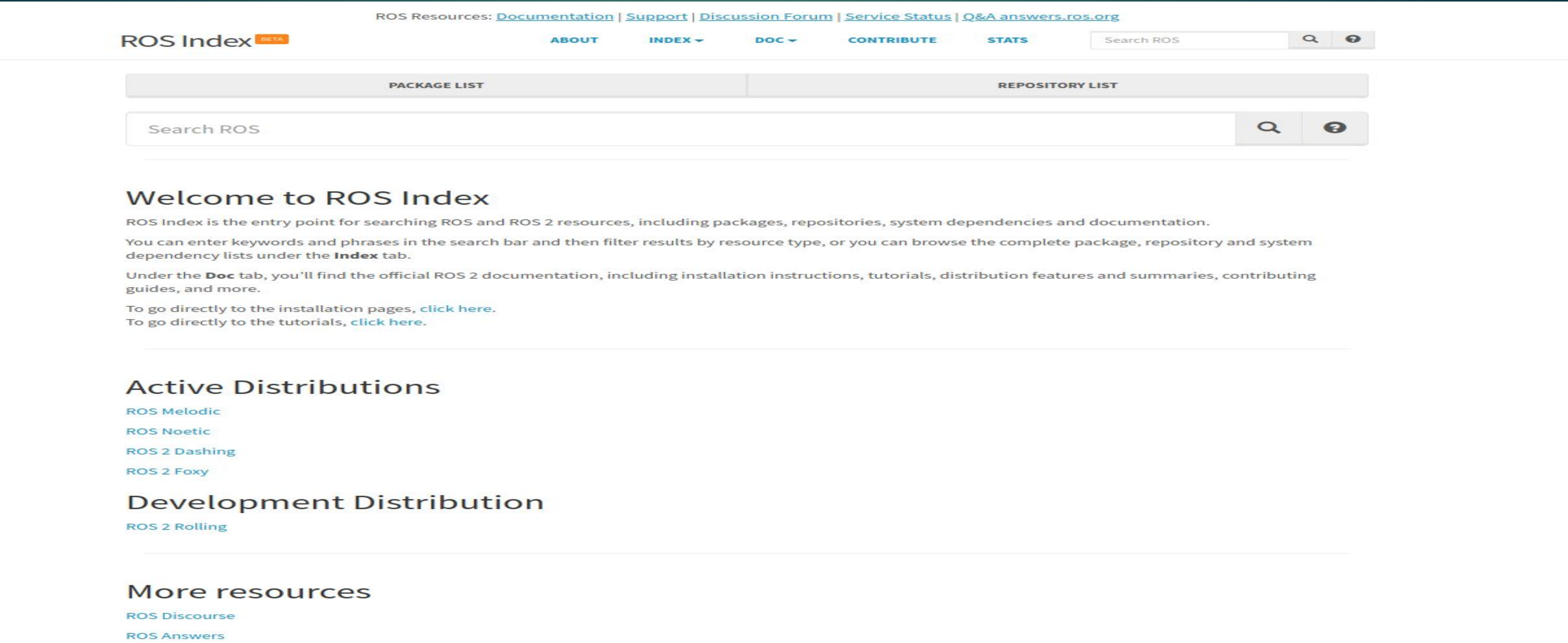
tekomoro

Intro to ROS2

(Robot Operating System)

Ros2 Index

https://index.ros.org/ (For Self Study)


ROS Resources: Documentation | Support | Discussion Forum | Service Status | Q&A answers.ros.org
ROS Index BETA
ABOUT
INDEX
DOC
CONTRIBUTE
STATS
Search ROS
PACKAGE LIST
REPOSITORY LIST
Search ROS

Welcome to ROS Index

ROS Index is the entry point for searching ROS and ROS 2 resources, including packages, repositories, system dependencies and documentation.

You can enter keywords and phrases in the search bar and then filter results by resource type, or you can browse the complete package, repository and system dependency lists under the **Index** tab.

Under the **Doc** tab, you'll find the official ROS 2 documentation, including installation instructions, tutorials, distribution features and summaries, contributing guides, and more.

To go directly to the installation pages, click here.
To go directly to the tutorials, click here.

Active Distributions

ROS Melodic
ROS Noetic
ROS 2 Dashing
ROS 2 Foxy

Development Distribution

ROS 2 Rolling

More resources

ROS Discourse
ROS Answers

tekomoro

How to use Colcon

to build Workspace use the Following Command

```
colcon build
```

you can ma symlink to your pkg use the following command

```
colcon build --packages-select <pkg_name> --symlink-install
```

Ros2 Node Tools (debug)

ros2 run -h

ros2 node (list,info)

ros2 node -h

Don't make 2 node with same name

source .bashrc

Remap : ros2 run pkg exec --ros-args --remap __node:=new_name

Colcon (pkg-select,symlink)

Topic Tools

```
ros2 topic list
ros2 topic echo    (subscriber in terminal)
ros2 topic info
ros2 topic hz
ros2 topic pub
ros2 topic pub /topic example_interfaces/msg/String "{data: "msg"}"
ros2 topic pub -r 5 /my_topic example_interfaces/msg/String "{data: "msg"}"
```

Remap Topic

```
ros2 run my_1 pub --ros-args --remap __node:=new_one --remap my_topic:=new_topic
```

How to Run Ros2 Node

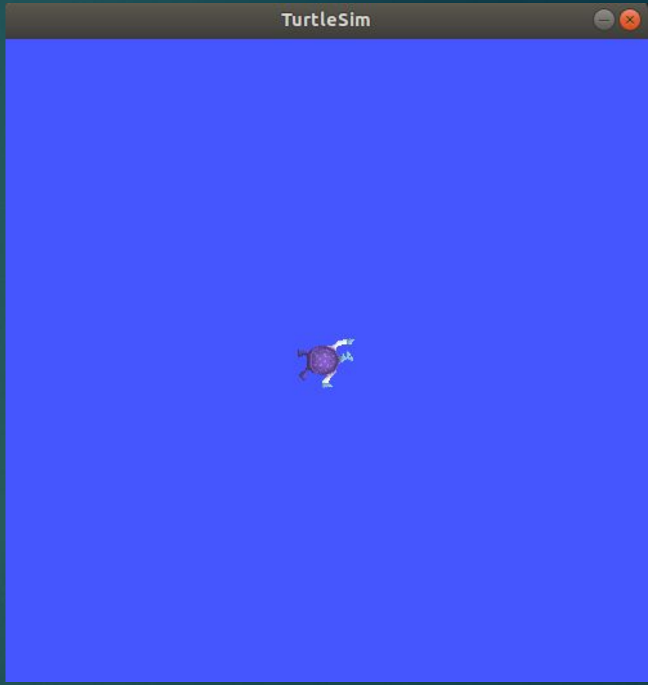
- Ros2 Run

```
ros2 run <package_name> <executable_name>
```

- To run turtlesim, open a new terminal, and enter the following command:

```
ros2 run turtlesim turtlesim_node
```

- The turtlesim window will open



- Here, the package name is turtlesim and the executable name is turtlesim_node

How to know running Node names

- ros2 node list

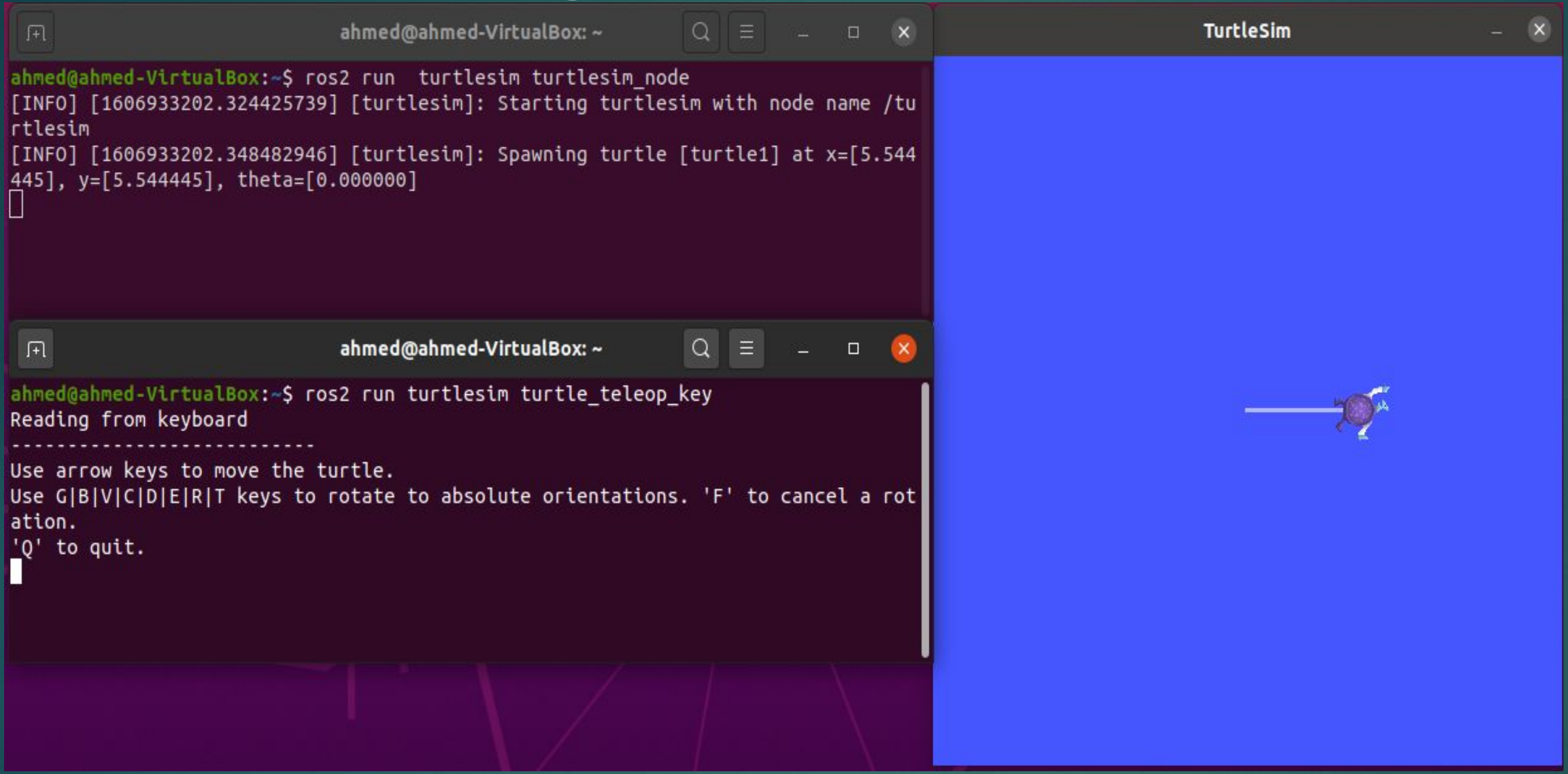
```
ros2 node list
```

- The terminal will return the node name: /turtlesim
- Open another new terminal and start the teleop node with the command

```
ros2 run turtlesim turtle_teleop_key
```

- Here, we are searching the turtlesim package again, this time for the executable named turtle_teleop_key.
- Return to the terminal where you ran ros2 node list and run it again. You will now see the names of two active nodes:/turtlesim, /teleop_turtle

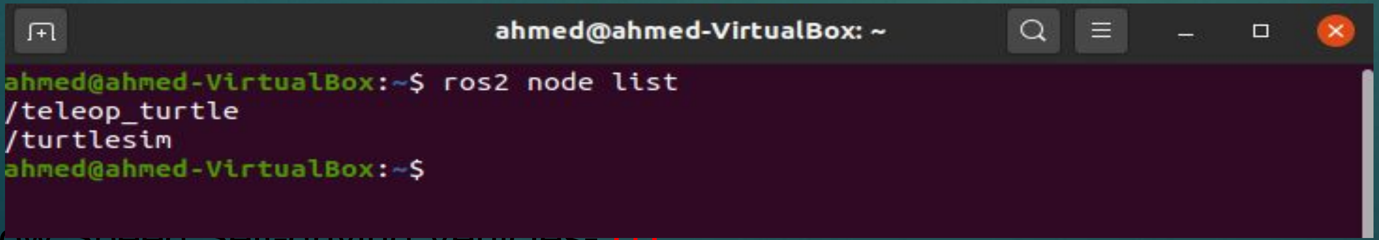
After running previous command



```
ahmed@ahmed-VirtualBox:~$ ros2 run  turtlesim turtlesim_node
[INFO] [1606933202.324425739] [turtlesim]: Starting turtlesim with node name /tu
rtlesim
[INFO] [1606933202.348482946] [turtlesim]: Spawning turtle [turtle1] at x=[5.544
445], y=[5.544445], theta=[0.000000]
```

```
ahmed@ahmed-VirtualBox:~$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rot
ation.
'Q' to quit.
```

- Type ros2 node list ,terminal will return the following:



```
ahmed@ahmed-VirtualBox:~$ ros2 node list
/teleop_turtle
/turtlesim
ahmed@ahmed-VirtualBox:~$
```

Ros2 node info

- you can access more information about the nodes using ros2 node info command :

```
ros2 node info <node_name>
```

- To examine your latest node, turtlesim, run the following command:

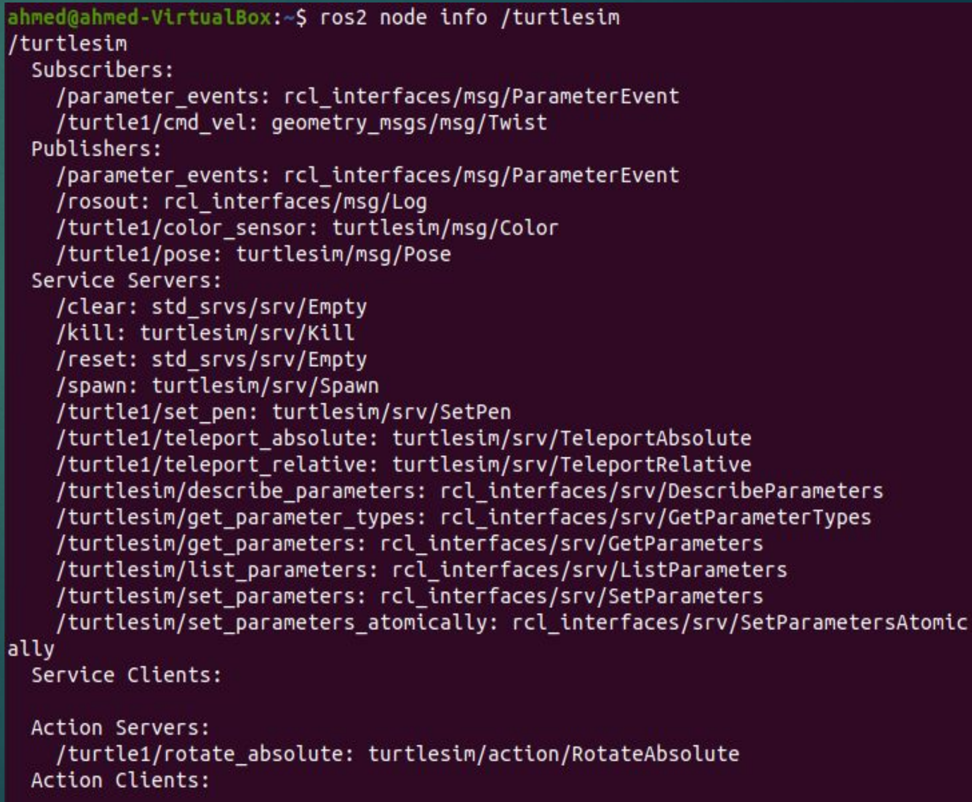
```
ros2 node info /turtlesim
```

- ros2 node info returns a list of subscribers, publishers,

services, and actions

(the ROS graph connections) that interact with that node.

The output should look like this:



```
ahmed@ahmed-VirtualBox:~$ ros2 node info /turtlesim
/turtlesim
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /turtle1/cmd_vel: geometry_msgs/msg/Twist
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /turtle1/color_sensor: turtlesim/msg/Color
    /turtle1/pose: turtlesim/msg/Pose
  Service Servers:
    /clear: std_srvs/srv/Empty
    /kill: turtlesim/srv/Kill
    /reset: std_srvs/srv/Empty
    /spawn: turtlesim/srv/Spawn
    /turtle1/set_pen: turtlesim/srv/SetPen
    /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
    /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
    /turtlesim/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /turtlesim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /turtlesim/get_parameters: rcl_interfaces/srv/GetParameters
    /turtlesim/list_parameters: rcl_interfaces/srv/ListParameters
    /turtlesim/set_parameters: rcl_interfaces/srv/SetParameters
    /turtlesim/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomic
ally
  Service Clients:

  Action Servers:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
  Action Clients:
```

Rqt_graph

- Open a new terminal and run:

```
ros2 run turtlesim turtlesim_node
```

- Open another terminal and run:

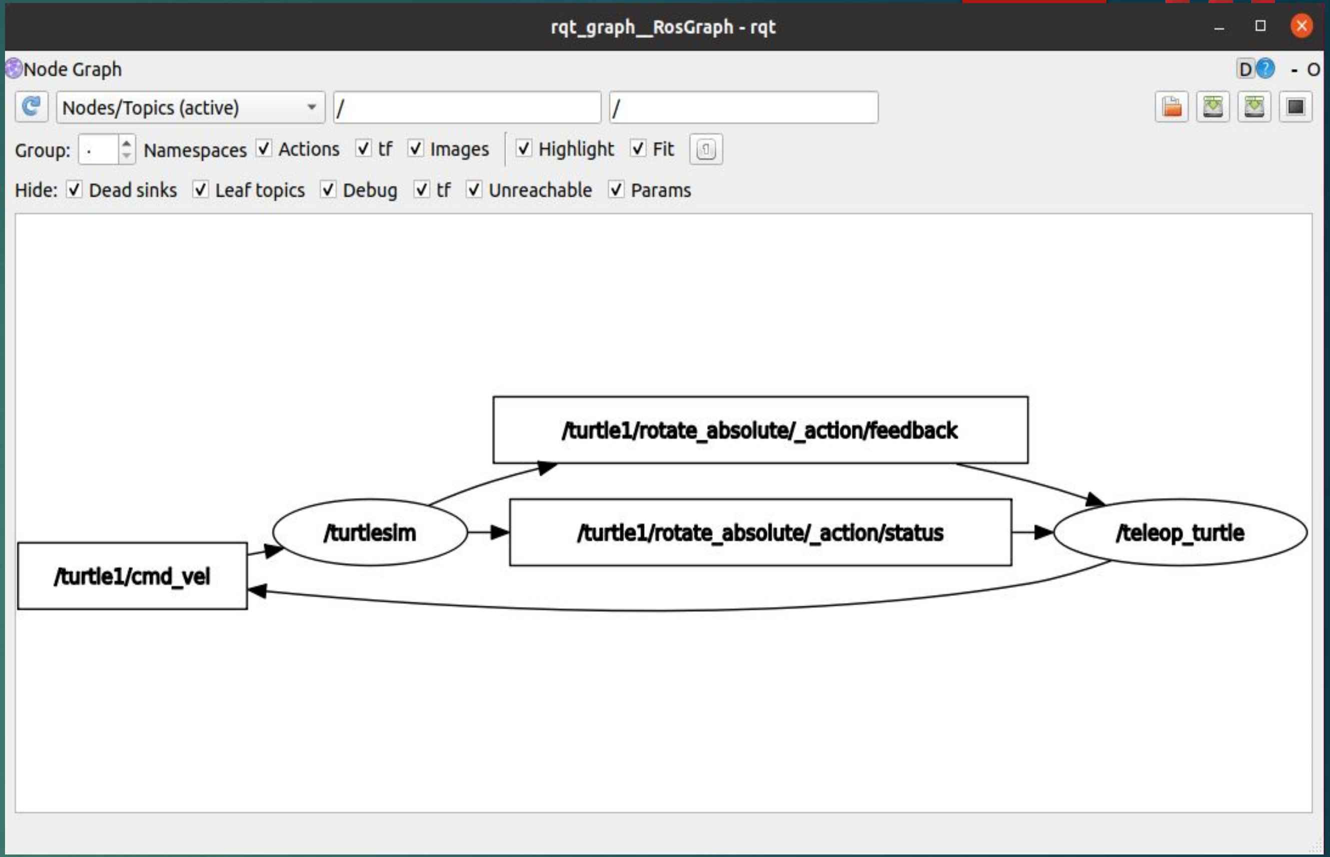
```
ros2 run turtlesim turtle_teleop_key
```

- Now run rqt_grapgh in another terminal

```
rqt_graph
```

- You should see the above nodes

and topic.

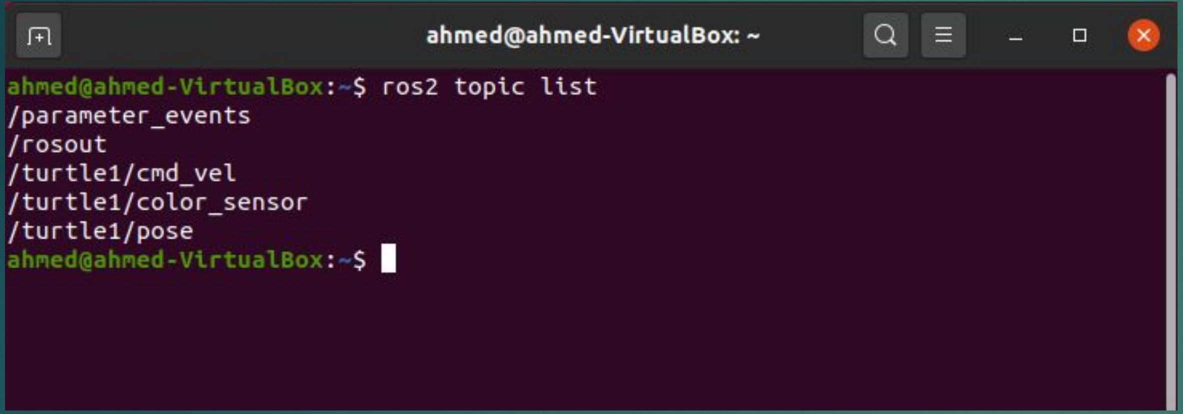

rqt_graph__RosGraph - rqt
Node Graph
Nodes/Topics (active)
Group: Namespaces Actions tf Images Highlight Fit
Hide: Dead sinks Leaf topics Debug tf Unreachable Params
/turtle1/rotate_absolute/_action/feedback
/turtlesim
/turtle1/rotate_absolute/_action/status
/teleop_turtle
/turtle1/cmd_vel

- The graph is depicting how the /turtlesim node and the /teleop_turtle node are communicating with each other over a topic. The /teleop_turtle node is publishing data (the keystrokes you enter to move the turtle around) to the /turtle1/cmd_vel topic, and the /turtlesim node is subscribed to that topic to receive the data.

How to show running topics

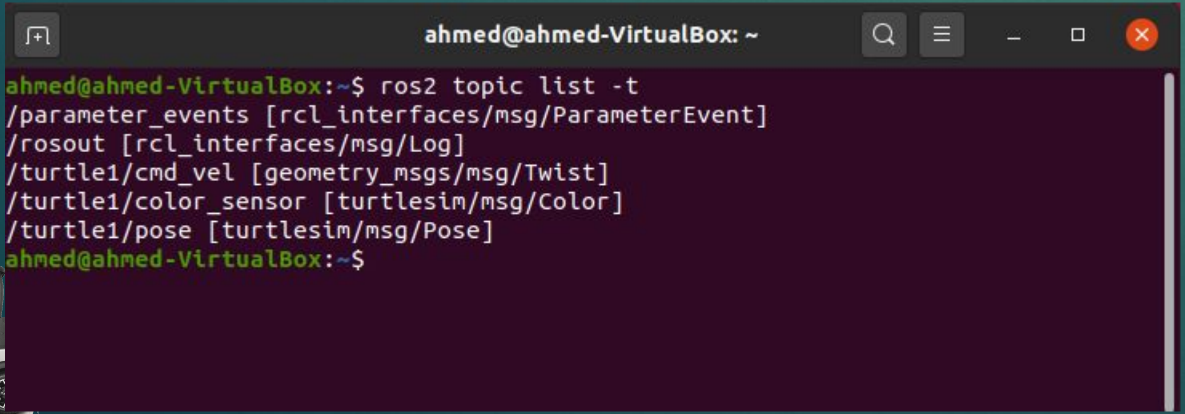
- Running the ros2 topic list command in a new terminal will return a list of all the topics currently active in the system:

```
ros2 topic list
```



```
ahmed@ahmed-VirtualBox:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
ahmed@ahmed-VirtualBox:~$
```

- ros2 topic list -t will return the same list of topics, this time with the topic type appended in brackets after each:



```
ahmed@ahmed-VirtualBox:~$ ros2 topic list -t
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
/turtle1/cmd_vel [geometry_msgs/msg/Twist]
/turtle1/color_sensor [turtlesim/msg/Color]
/turtle1/pose [turtlesim/msg/Pose]
ahmed@ahmed-VirtualBox:~$
```

tekomoro

How To see the data being published on a topic

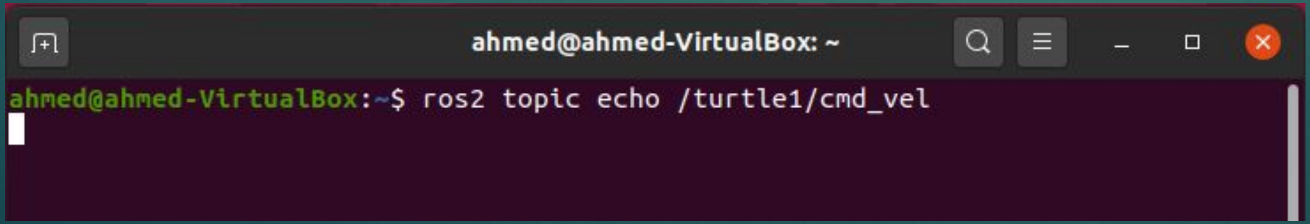
- To see the data being published on a topic, use:

```
ros2 topic echo <topic_name>
```

- Since we know that /teleop_turtle publishes data to /turtlesim over the /turtle1/cmd_vel topic, let's use echo to introspect on that topic:

```
ros2 topic echo /turtle1/cmd_vel
```

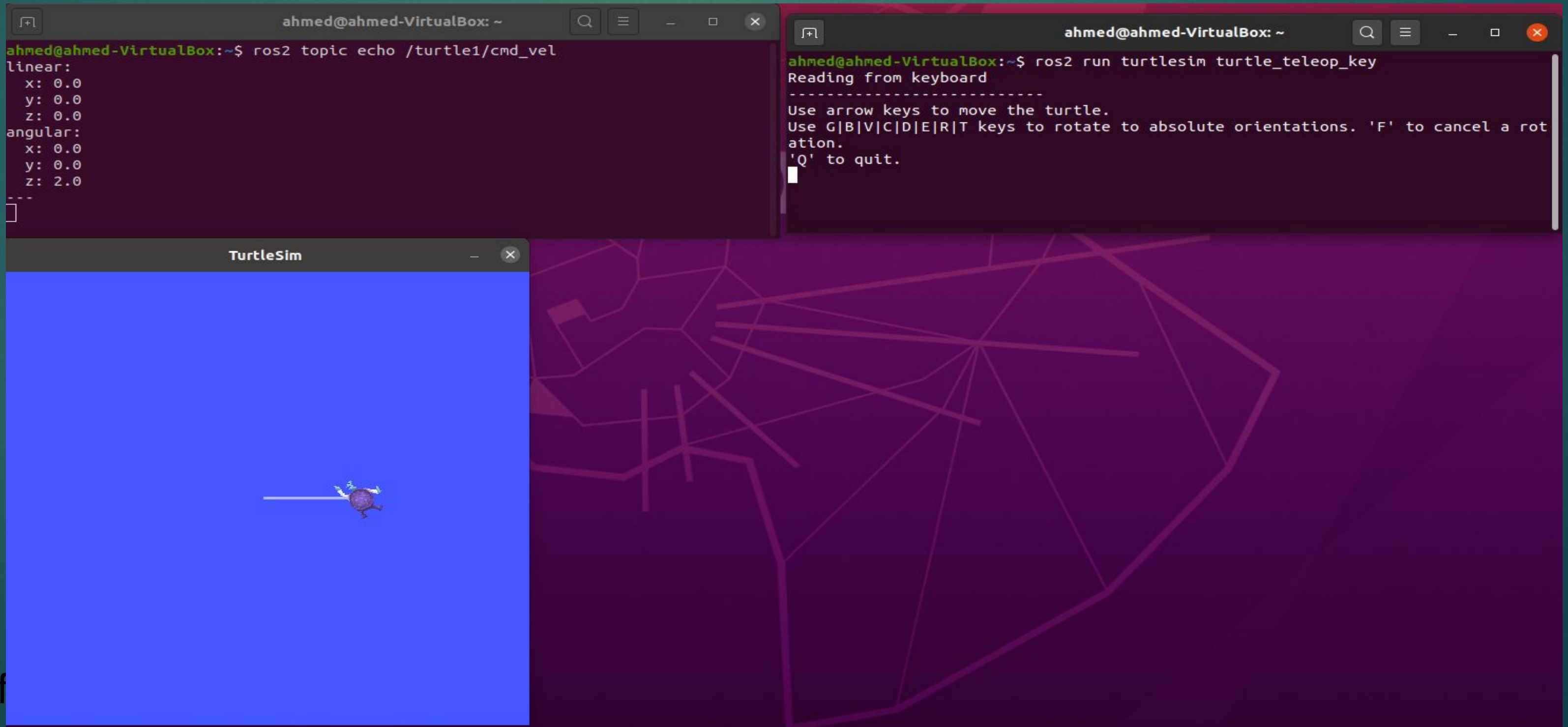
- At first, this command won't return any data. That's because it's waiting for /teleop_turtle to publish something



```
ahmed@ahmed-VirtualBox:~$ ros2 topic echo /turtle1/cmd_vel
█
```

How To see the data being published on a topic

- Return to the terminal where turtle_teleop_key is running and use the arrows to move the turtle around. Watch the terminal where your echo is running at the same time, and you'll see position data being published for every movement you make:



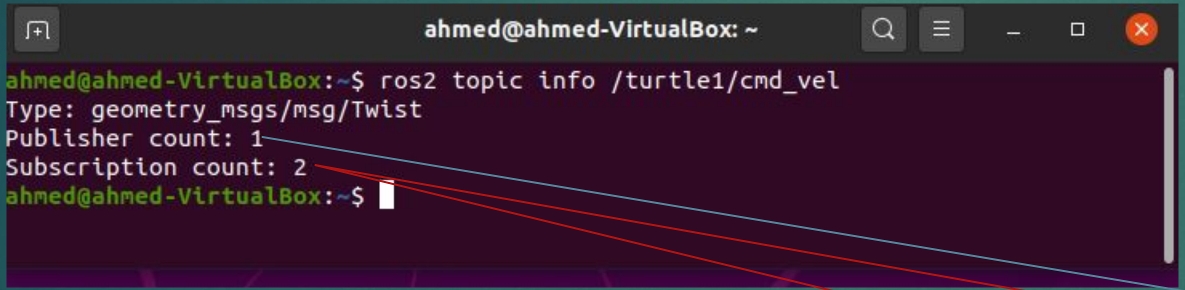
```
ahmed@ahmed-VirtualBox:~$ ros2 topic echo /turtle1/cmd_vel
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 2.0
---
```

```
ahmed@ahmed-VirtualBox:~$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rot
ation.
'Q' to quit.
```

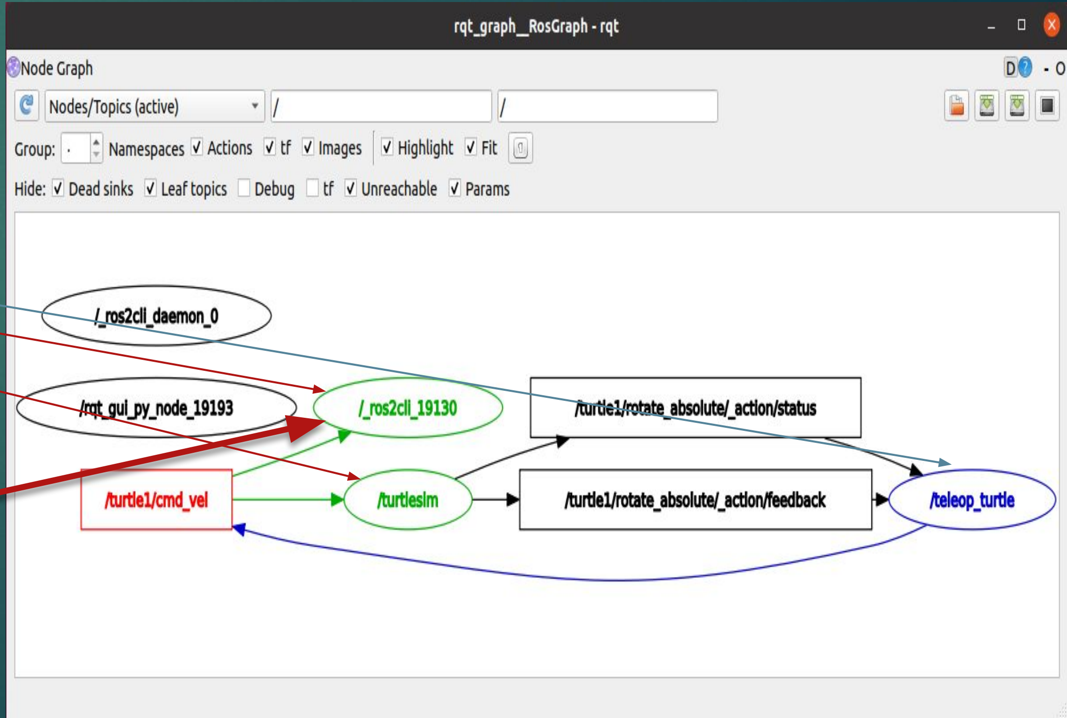
Ros2 topic info

- Topics don’t have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.
- To know further type:

```
ros2 topic info /turtle1/cmd_vel
```



```
ahmed@ahmed-VirtualBox:~$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 2
ahmed@ahmed-VirtualBox:~$
```

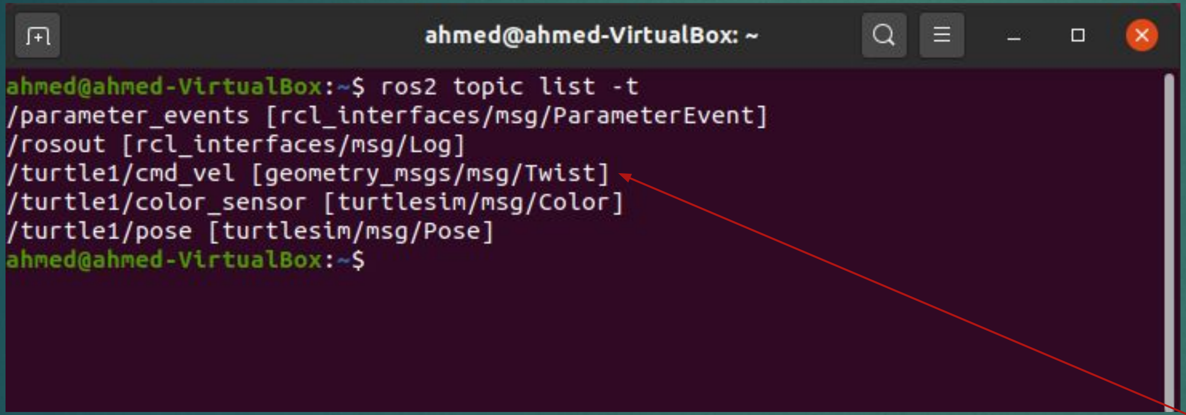


Echo node that subscribe from topic /turtle1/cmd_vel

tekomoro

ros2 interface show

- Nodes send data over topics using messages. Publishers and subscribers must send and receive the same type of message to communicate



```
ahmed@ahmed-VirtualBox:~$ ros2 topic list -t
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
/turtle1/cmd_vel [geometry_msgs/msg/Twist]
/turtle1/color_sensor [turtlesim/msg/Color]
/turtle1/pose [turtlesim/msg/Pose]
ahmed@ahmed-VirtualBox:~$
```

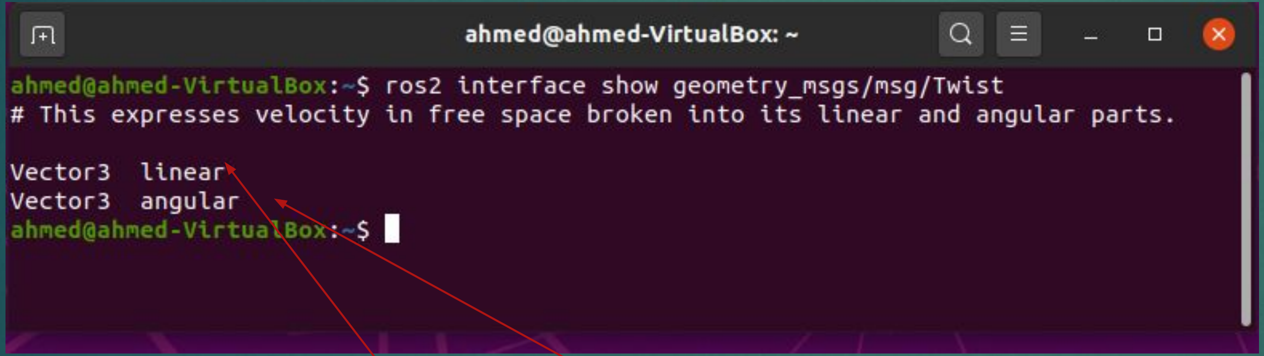
```
geometry_msgs/msg/Twist
```

- This means that in the package geometry_msgs there is a msg called Twist.

ros2 interface show

- Now we can run ros2 interface show <type>.msg on this type to learn its the details, specifically, what structure of data the message expects.

```
ros2 interface show geometry_msgs/msg/Twist
```



```
ahmed@ahmed-VirtualBox:~$ ros2 interface show geometry_msgs/msg/Twist
# This expresses velocity in free space broken into its linear and angular parts.

Vector3  linear
Vector3  angular
ahmed@ahmed-VirtualBox:~$
```

- This tells you that the /turtlesim node is expecting a message with two vectors, linear and angular

How to publish data onto a topic directly from the command line

- Now that you have the message structure, you can publish data onto a topic directly from the command line using:

```
ros2 topic pub <topic_name> <msg_type> '<args>'
```

- The '<args>' argument is the actual data you'll pass to the topic
- It's important to note that this argument needs to be input in YAML syntax
- Example

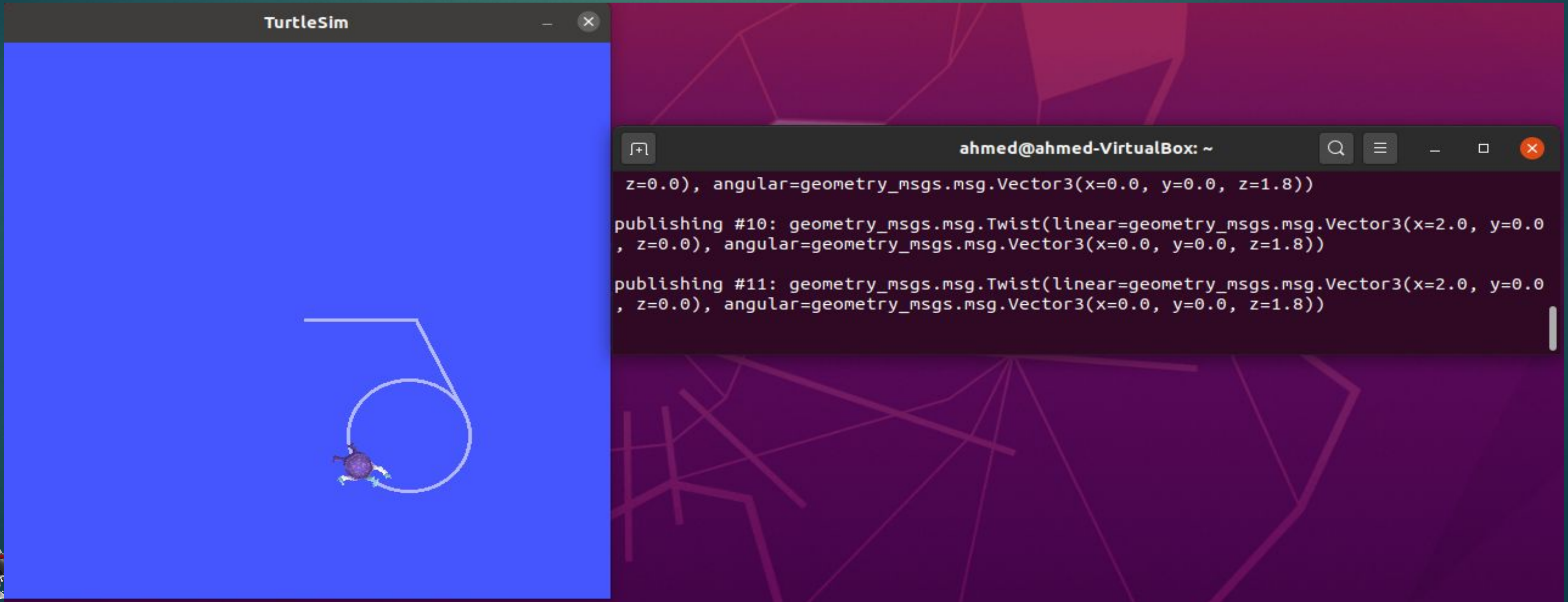
```
ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

- --once is an optional argument meaning "publish one message then exit".

How to publish data onto a topic directly from the command line

- We can specify the frequency of publishing data using option:

```
ros2 topic pub –r<frequency number> /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```



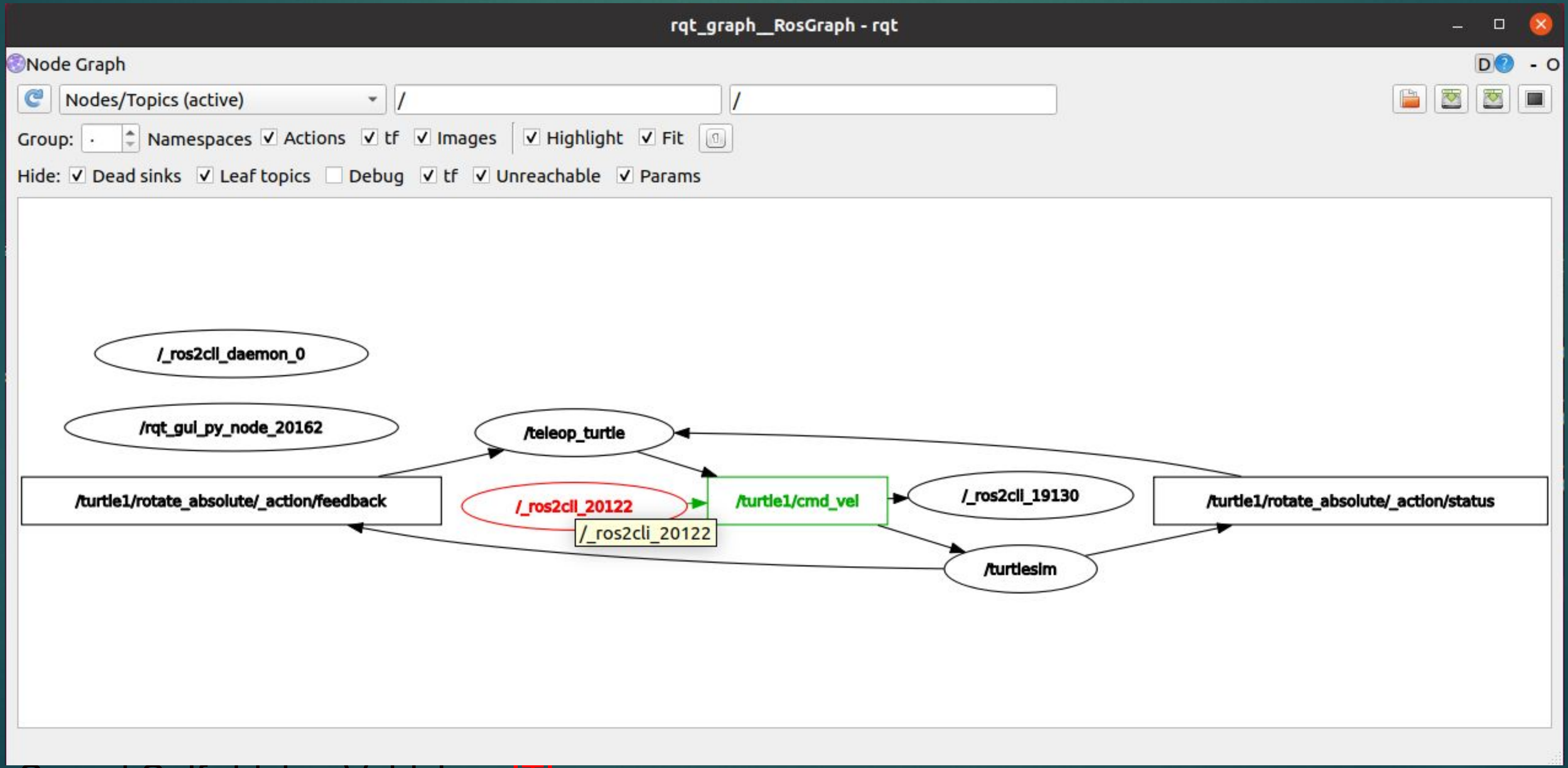
```
 z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8))

publishing #10: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0
, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8))

publishing #11: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0
, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8))
```

publish data onto a topic directly from the command line

- You can refresh rqt_graph to see what's happening graphically.

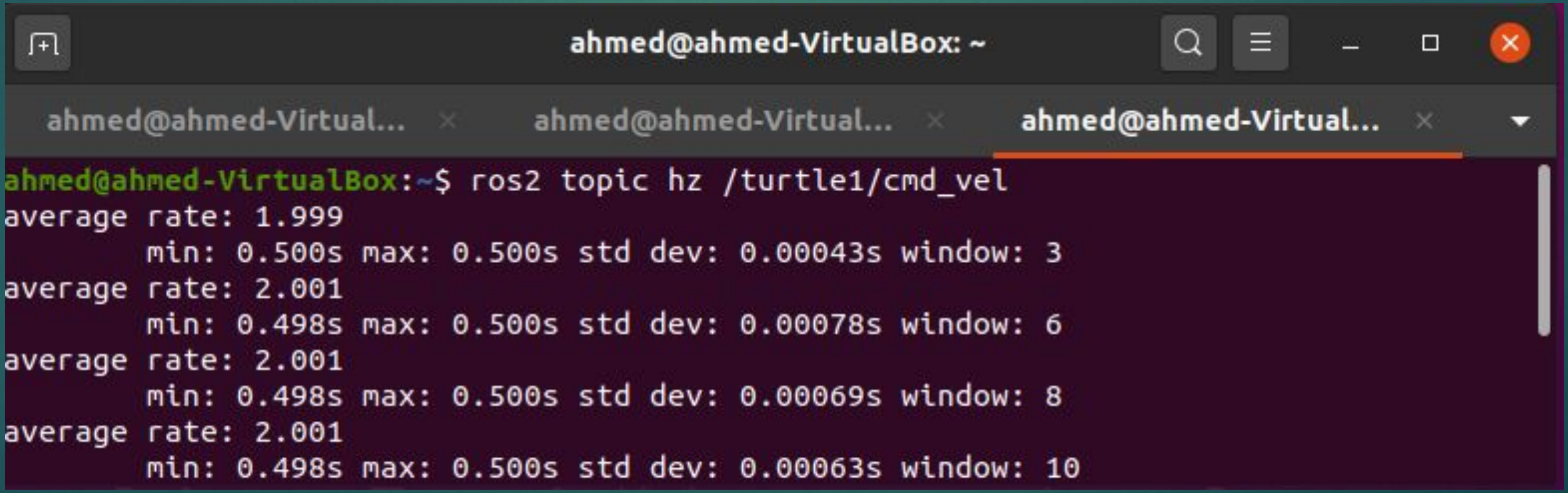


ros2 topic hz

- you can report the rate at which data is published using:

```
ros2 topic hz /turtle1/cmd_vel
```

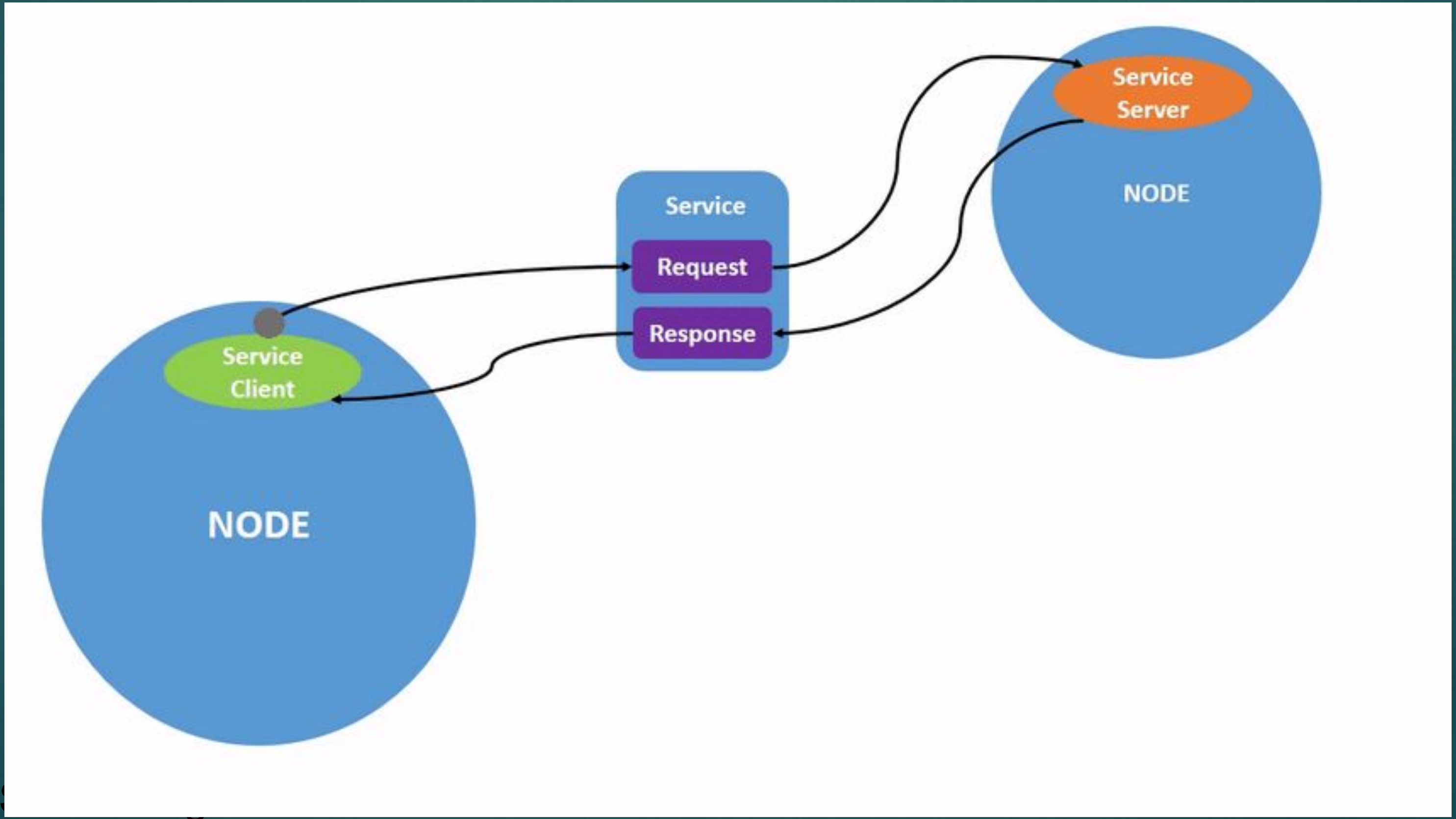
- It will return data on the rate at which the node is publishing data to the cmd_vel topic.



```
ahmed@ahmed-VirtualBox:~$ ros2 topic hz /turtle1/cmd_vel
average rate: 1.999
        min: 0.500s max: 0.500s std dev: 0.00043s window: 3
average rate: 2.001
        min: 0.498s max: 0.500s std dev: 0.00078s window: 6
average rate: 2.001
        min: 0.498s max: 0.500s std dev: 0.00069s window: 8
average rate: 2.001
        min: 0.498s max: 0.500s std dev: 0.00063s window: 10
```

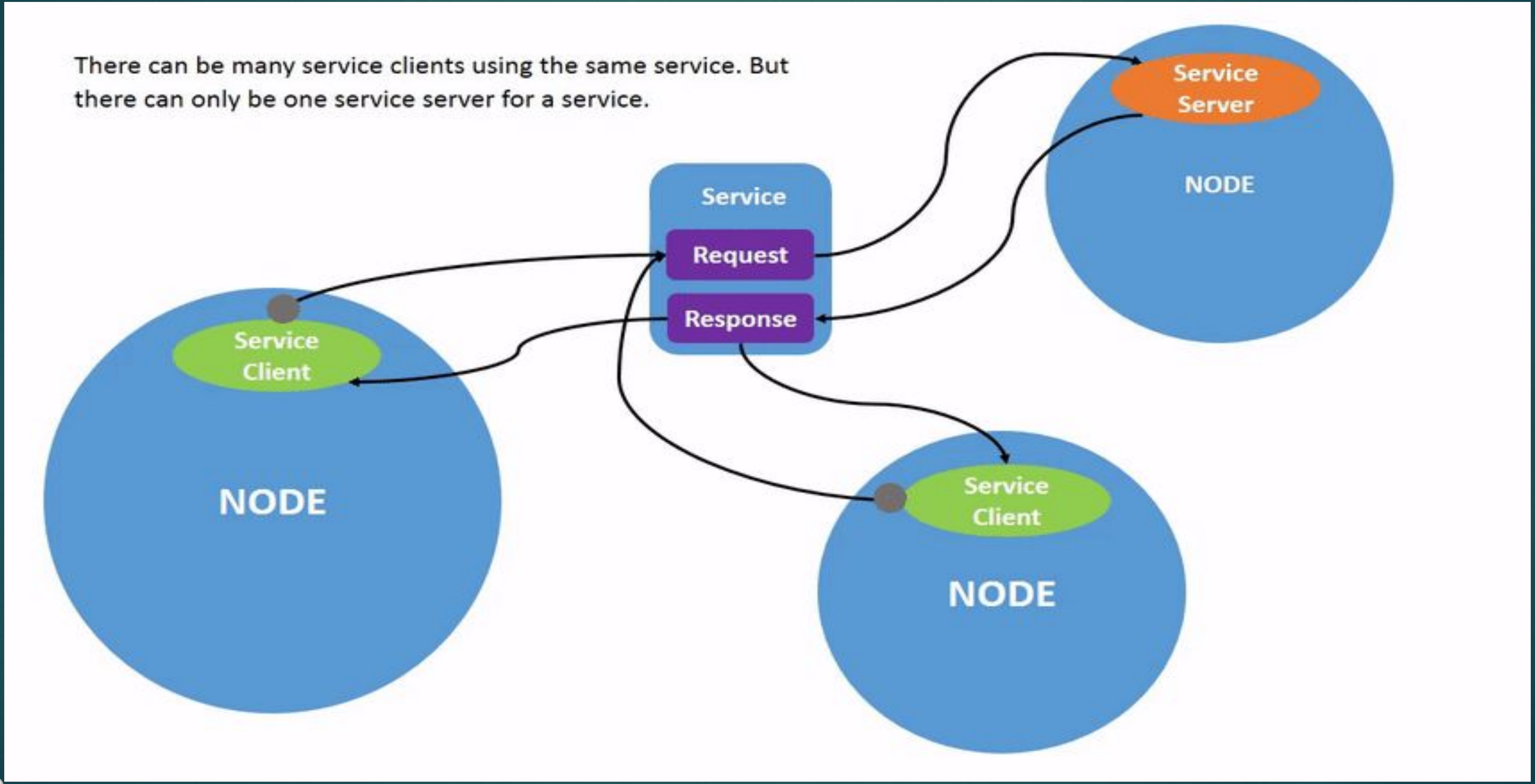
Understanding ROS 2 services

- Services are another method of communication for nodes on the ROS graph. Services are based on a call-and-response model



tekomoro

ROS2 Services


There can be many service clients using the same service. But there can only be one service server for a service.

Make Service Server

- make service client simple and fundamental concept
- make service client Using OOP

How to list services

- Open a new terminal and run

```
ros2 run turtlesim turtlesim_node
```

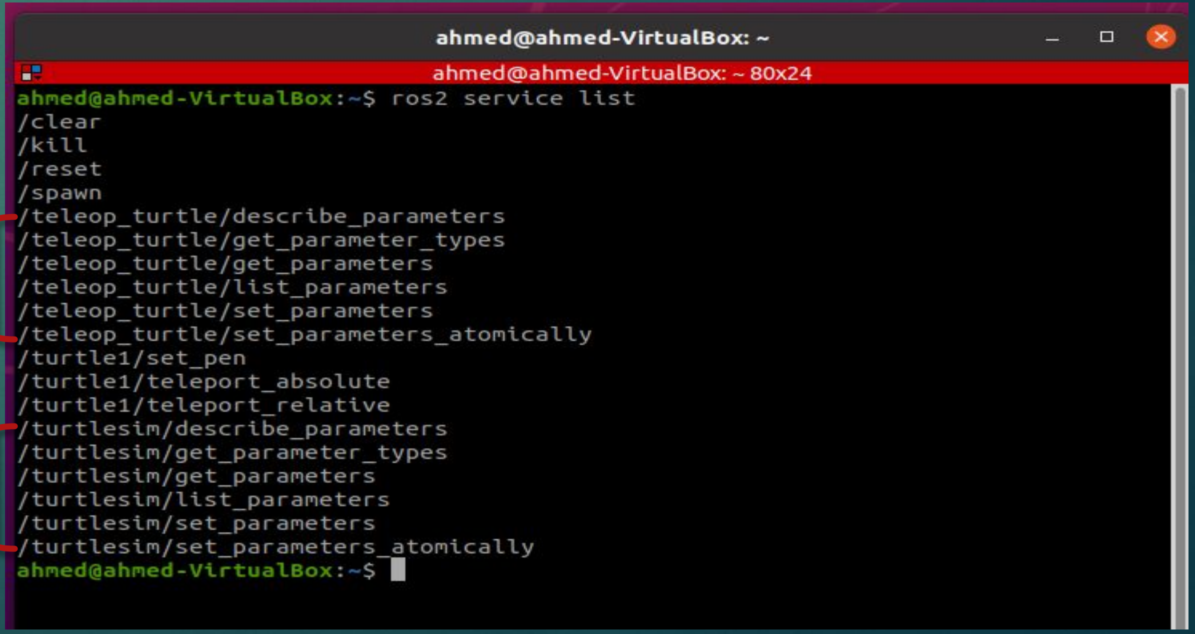
- Open another terminal and run:

```
ros2 run turtlesim turtle_teleop_key
```

- Running the ros2 service list command in a new terminal will return a list of all the services currently active in the system:

```
ros2 service list
```

- You will see that both nodes have the same six services with parameters in their names. Nearly every node in ROS 2 has these infrastructure services that parameters are built off of.



```
ahmed@ahmed-VirtualBox:~$ ros2 service list
/clear
/kill
/reset
/spawn
/teleop_turtle/describe_parameters
/teleop_turtle/get_parameter_types
/teleop_turtle/get_parameters
/teleop_turtle/list_parameters
/teleop_turtle/set_parameters
/teleop_turtle/set_parameters_atomically
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
ahmed@ahmed-VirtualBox:~$
```

tekomoro

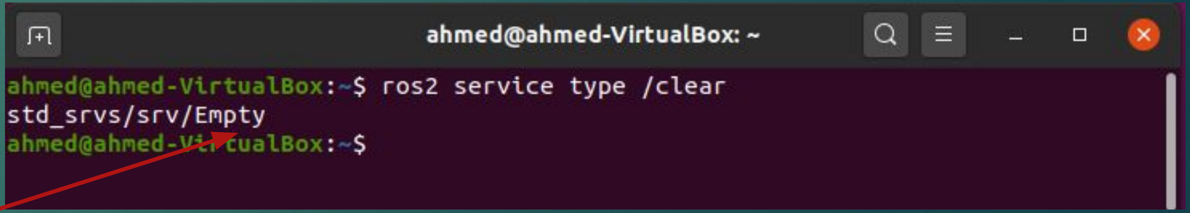
ros2 service type

- Services have types that describe how the request and response data of a service is structured. Service types are defined similarly to topic types, except service types have two parts: one message for the request and another for the response.
- To find out the type of a service, use the command:

```
ros2 service type <service_name>
```

- Let's take a look at turtlesim's /clear service. In a new terminal, enter the command:

```
ros2 service type /clear
```



ahmed@ahmed-VirtualBox: ~

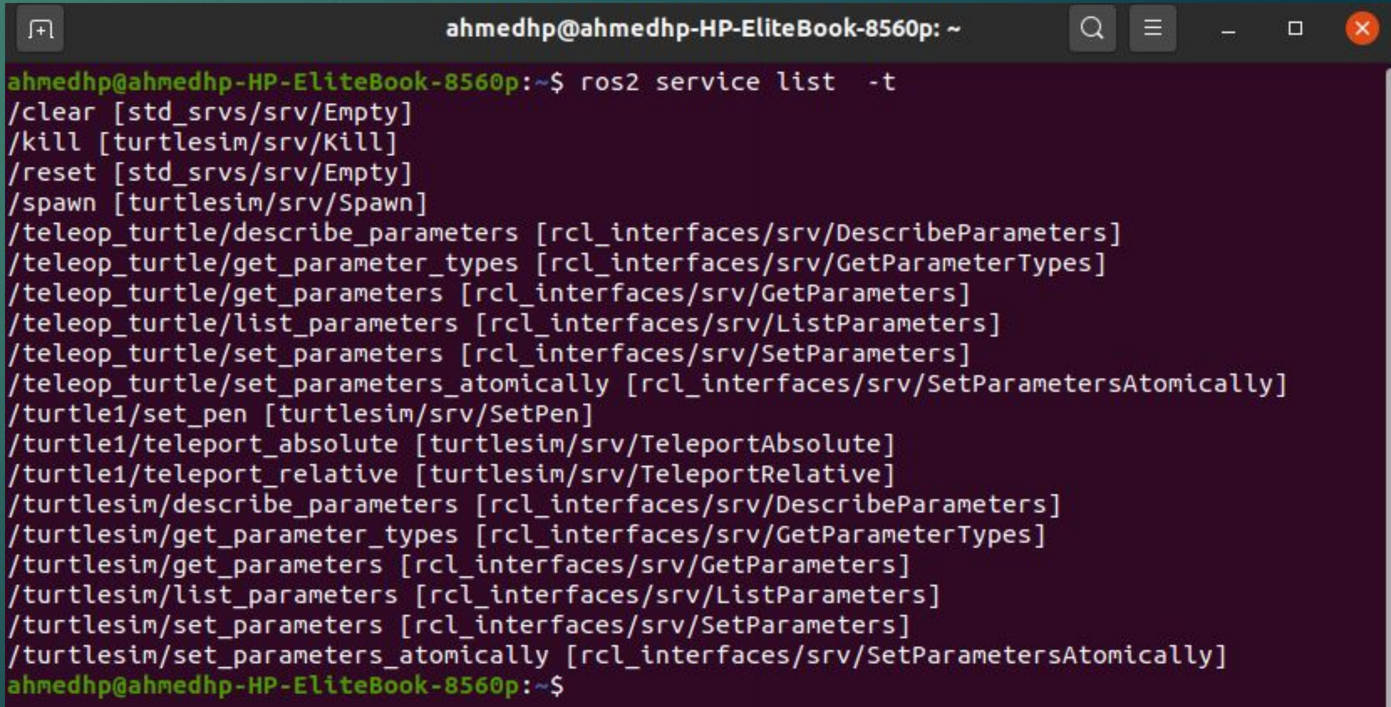
```
ahmed@ahmed-VirtualBox:~$ ros2 service type /clear
std_srvs/srv/Empty
ahmed@ahmed-VirtualBox:~$
```

- The Empty type means the service call sends no data when making a request and receives no data when receiving a response.

How to list services with type

To see the types of all the active services at the same time, you can append the -t to list

```
ros2 service list -t
```



```
ahmedhp@ahmedhp-HP-EliteBook-8560p:~$ ros2 service list  -t
/clear [std_srvs/srv/Empty]
/kill [turtlesim/srv/Kill]
/reset [std_srvs/srv/Empty]
/spawn [turtlesim/srv/Spawn]
/teleop_turtle/describe_parameters [rcl_interfaces/srv/DescribeParameters]
/teleop_turtle/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]
/teleop_turtle/get_parameters [rcl_interfaces/srv/GetParameters]
/teleop_turtle/list_parameters [rcl_interfaces/srv/ListParameters]
/teleop_turtle/set_parameters [rcl_interfaces/srv/SetParameters]
/teleop_turtle/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]
/turtle1/set_pen [turtlesim/srv/SetPen]
/turtle1/teleport_absolute [turtlesim/srv/TeleportAbsolute]
/turtle1/teleport_relative [turtlesim/srv/TeleportRelative]
/turtlesim/describe_parameters [rcl_interfaces/srv/DescribeParameters]
/turtlesim/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]
/turtlesim/get_parameters [rcl_interfaces/srv/GetParameters]
/turtlesim/list_parameters [rcl_interfaces/srv/ListParameters]
/turtlesim/set_parameters [rcl_interfaces/srv/SetParameters]
/turtlesim/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]
ahmedhp@ahmedhp-HP-EliteBook-8560p:~$
```

ros2 service find

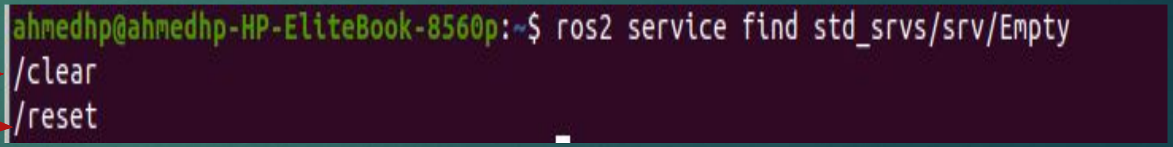
If you want to find all the services of a specific type, you can use the command:

```
ros2 service find <type name>
```

For example, you can find all the Empty typed services like this:

```
ros2 service find std_srvs/srv/Empty
```

Service names



```
ahmedhp@ahmedhp-HP-EliteBook-8560p:~$ ros2 service find std_srvs/srv/Empty
/clear
/reset
```

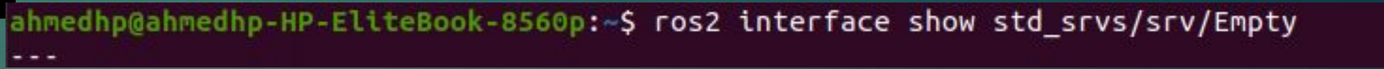
ros2 interface show

to know the structure of the input arguments for service you can use ros2 interface show command

```
ros2 interface show <type_name>.srv
```

To run this command on the /clear service's type, Empty:

```
ros2 interface show std_srvs/srv/Empty.srv
```



```
ahmedhp@ahmedhp-HP-EliteBook-8560p:~$ ros2 interface show std_srvs/srv/Empty
---
```

The --- separates the request structure (above) from the response structure (below).

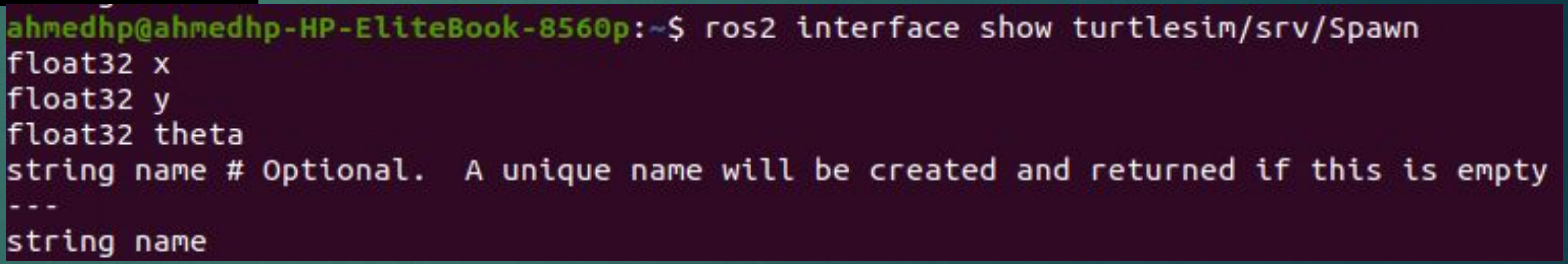
the Empty type doesn't send or receive any data. So, naturally, its structure is blank.

ros2 interface show

Let's introspect a service with a type that sends and receives data, like /spawn

To see the arguments in a /spawn call-and-request, run the command:

```
ros2 interface show turtlesim/srv/Spawn.srv
```



```
ahmedhp@ahmedhp-HP-EliteBook-8560p:~$ ros2 interface show turtlesim/srv/Spawn
float32 x
float32 y
float32 theta
string name # Optional.  A unique name will be created and returned if this is empty
---
string name
```

The information above the --- line tells us the arguments needed to call /spawn. x, y and theta determine the location of the spawned turtle, and name is clearly optional.

ros2 service call

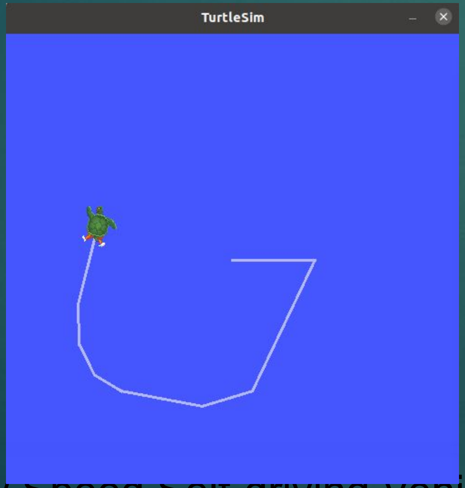
you can call a service using:

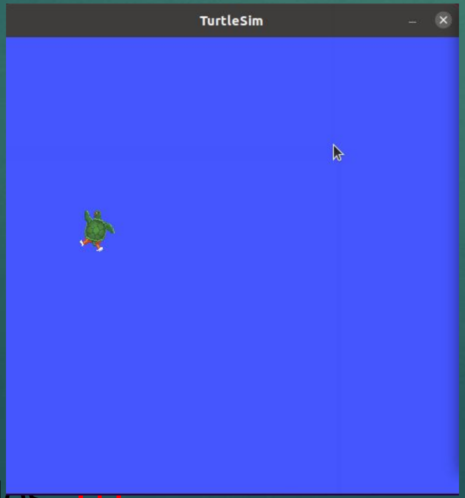
```
ros2 service call <service_name> <service_type> <arguments>
```

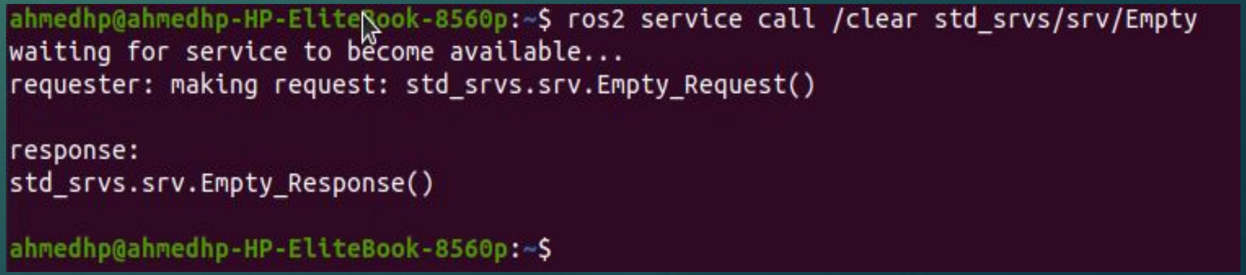
The <arguments> part is optional. For example, you know that Empty typed services don't have any arguments:

```
ros2 service call /clear std_srvs/srv/Empty
```

This command will clear the turtlesim window of any lines your turtle has drawn.







```
ahmedhp@ahmedhp-HP-EliteBook-8560p:~$ ros2 service call /clear std_srvs/srv/Empty
waiting for service to become available...
requester: making request: std_srvs.srv.Empty_Request()

response:
std_srvs.srv.Empty_Response()

ahmedhp@ahmedhp-HP-EliteBook-8560p:~$
```

Spawn New Turtle

let’s spawn a new turtle by calling /spawn and inputting arguments. Input <arguments> in a service call from the command-line need to be in YAML syntax.

```
ros2 service call /clear std_srvs/srv/Empty
```