

ROS 2

[03 – ROS2 Basic]

한국폴리텍대학교 성남캠퍼스

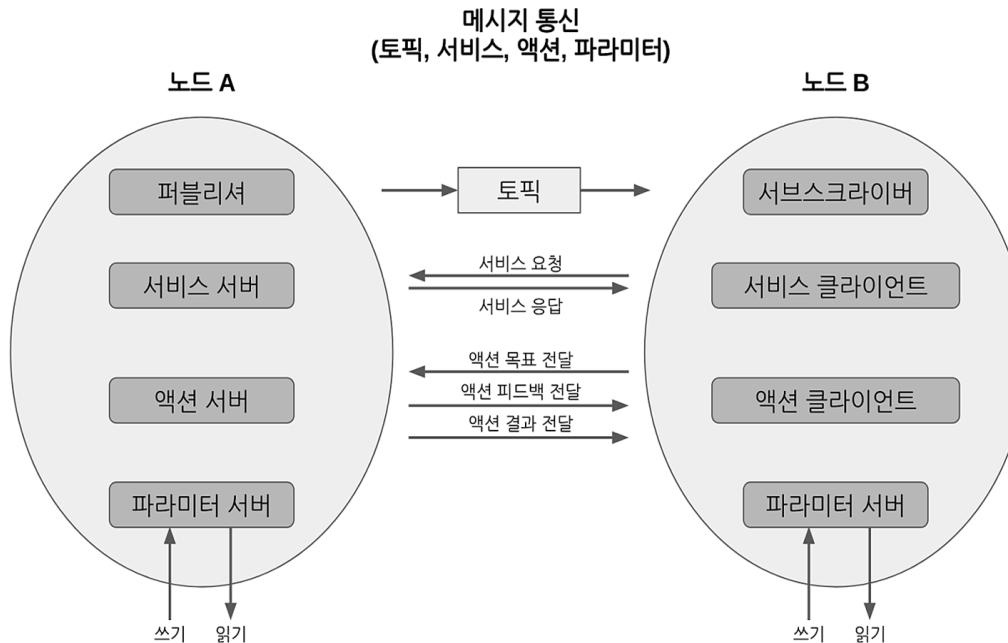
1 ROS2 Overview

❖ Message Communication

- [노드] ROS에서 최소 단위의 실행 가능한 프로세스
 - [패키지] 하나 이상의 노드 또는 노드 실행을 위한 정보 등을 묶어 놓은 것
 - [메타 패키지] 패키지의 묶음
-
- ROS에서는 최소한의 실행 단위로 프로그램을 나누어 프로그래밍
 - 노드는 각각 별개의 프로그램으로 보면 됨
 - 수많은 노드들이 연동되는 ROS 시스템의 특성 상 노드와 노드 사이에 입력과 출력 데이터를 서로 주고받도록 설계
-
- 노드의 이해와 더불어 노드 간 데이터 교환 방법, 데이터 유형에 대한 이해가 필수

❖ ROS2 Message Communication

- 노드 간 주고 받는 데이터를 메시지(message)라 칭하며, 주고 받는 방식을 메시지 통신이라 함
- 메시지 데이터는 integer, floating point, boolean, string과 같은 변수로 구성
- 메시지를 주고받는 통신 방법에 따라 토픽(topic), 서비스(service), 액션(action), 파라미터(parameter)로 구분

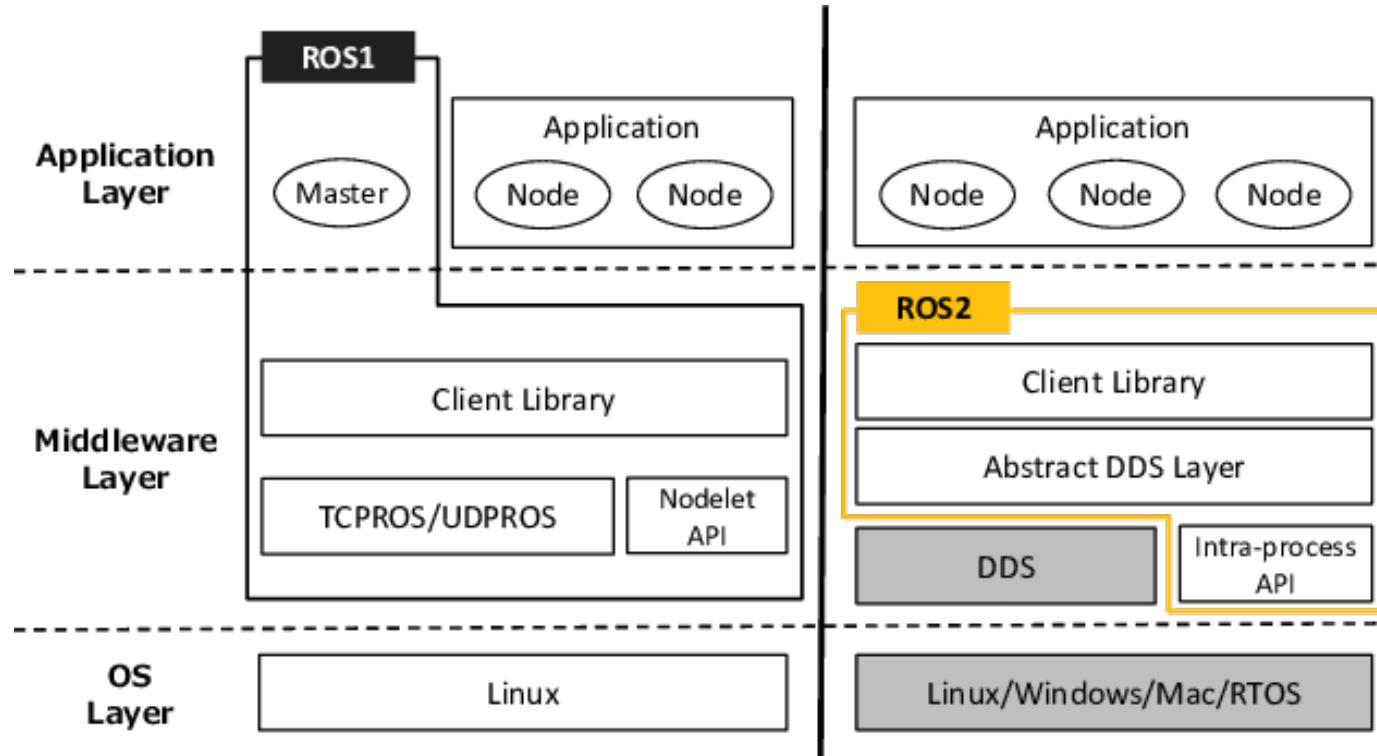


❖ ROS2 Message Communication

- ROS에서 사용되는 각 메시지 통신 방법의 목적과 사용 방법은 다르기는 하지만 토픽의 발간(publish)과 구독(subscribe)의 개념을 응용
- 데이터를 보내고 받는 발간, 구독 개념은 ROS 1은 물론 ROS 2에서도 매우 중요한 개념으로 변함이 없음
- ROS 1, 2에 사용되는 통신 라이브러리에 적용된 기술에는 차이가 있음
- ROS 1에서는 자체 개발한 TCPROS 통신 라이브러리 사용
- ROS 2에서는 OMG(Object Management Group)에 의해 표준화된 DDS(Data Distribution Service) 통신 라이브러리 사용

❖ ROS2 Message Communication

- 로봇 기술의 시장 진입을 위해 표준 방식인 DDS의 Real Time Publish와 Subscribe 프로토콜인 DDSI-RTPS를 사용



❖ ROS2 Message Communication

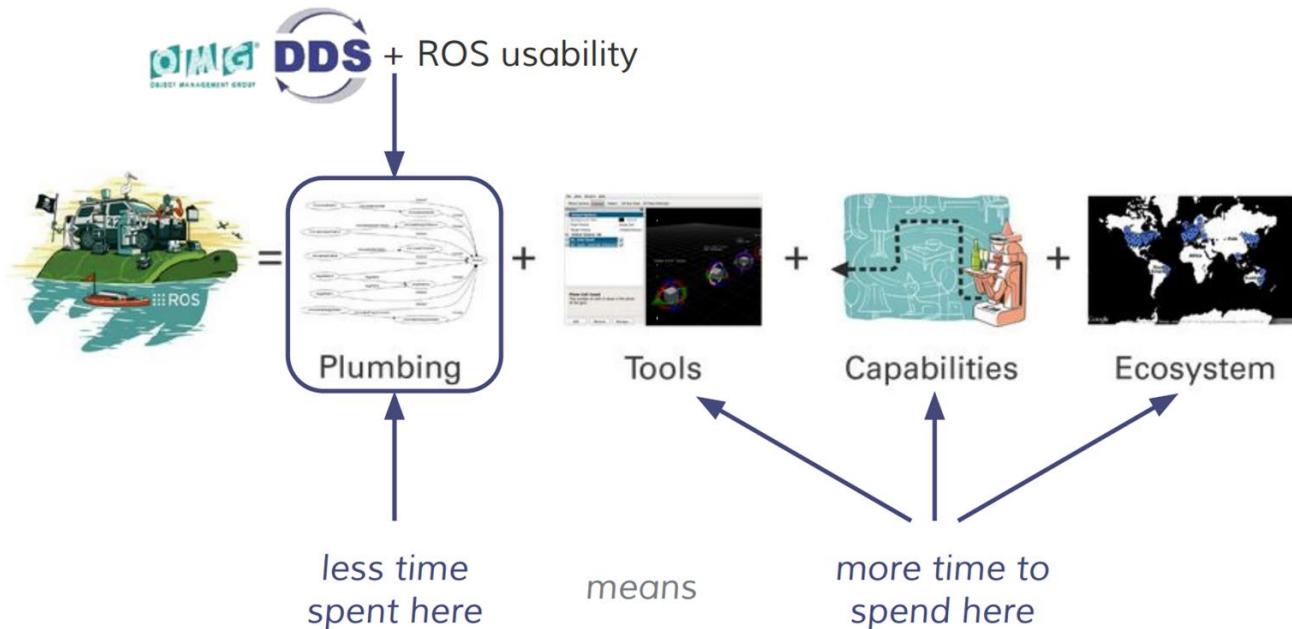
- DDS 도입으로 기존 메시지 형태 이외 OMG의 IDL(Interface Description Language)를 사용
 - ✓ 메시지 정의 및 직렬화를 더 쉽게, 더 포괄적으로 다룰 수 있게 되었음
- DDS의 중요 컨셉인 DCPS(data-centric publish-subscribe), DLRL(data local reconstruction layer)의 내용이 적용된 DDSI-RTPS을 채용
 - ✓ 실시간 데이터 전송 보장, 임베디드 시스템도 사용할 수 있음
- DDS의 사용으로 노드 간의 동적 검색 기능을 지원
 - ✓ 프로그램 간에 통신 가능 ROS1에서 각 노드들의 정보를 관리하였던 ROS Master가 없어도 여러 DDS 간의 통신이 가능함

❖ ROS2 Message Communication

- 노드 간의 데이터 통신을 세부적으로 조정하는 QoS(Quality of Service)를 매개 변수 형태로 설정할 수 있음
 - ✓ TCP처럼 데이터 손실 방지로 신뢰성 향상
 - ✓ UDP처럼 통신 속도를 우선시하여 사용할 수 있음
- 산업용으로 사용되는 미들웨어로 DDS-Security 도입으로 보안 강화
- 다양한 기능을 갖춘 DDS를 이용하여 ROS1의 퍼블리시, 서브스크라이브형 메시지 전달은 물론, 실시간 데이터 전송, 불안정한 네트워크에 대한 대응, 보안 등이 강화

❖ ROS2 Message Communication

- DDS의 채용은 ROS1에서 ROS2로의 전환에서 가장 큰 차이점이라 할 수 있음
- 개발자 및 사용자로 하여금 통신 미들웨어에 대한 개발 및 이용 부담을 줄여 진짜로 집중해야 할 부분에 더 많은 시간을 쓸을 수 있는 환경 구축



2 Data Distribution Service

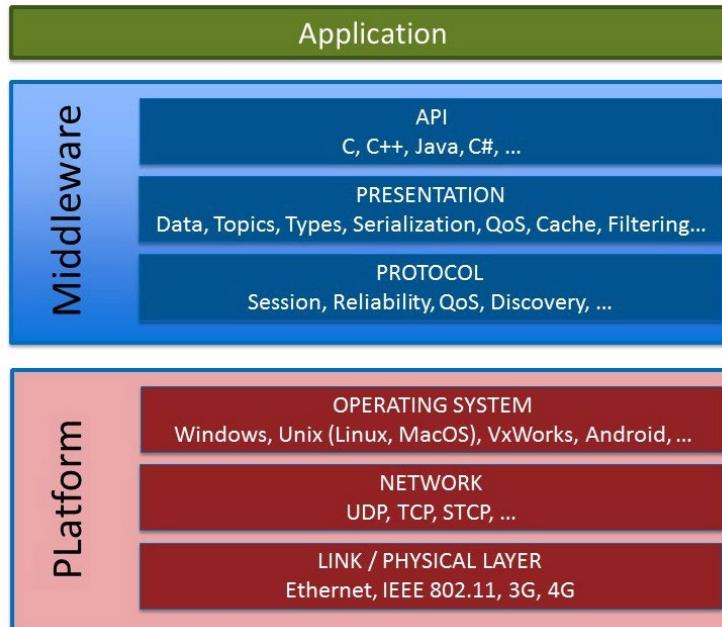
❖ What is DDS?

- DDS는 데이터 분산 시스템이라는 용어로 OMG에서 표준을 정하고자 만든 트레이드 마크(TM)
- 데이터 통신을 위한 미들웨어
- DDS가 ROS2의 미들웨어로 사용되고 있는 표준임을 감안하면 너무 자세히 알 필요는 없으며, ROS 프로그래밍에 필요한 개념만 학습하면 됨
- DDS(Data Distribution Service, 데이터 분산 서비스)

Data Distribution Service

❖ What is DDS?

- DDS는 데이터를 중심으로 연결성을 갖는 미들웨어의 프로토콜(DDSI-RTPS)과 같은 DDS 사양을 만족하는 미들웨어 API
- OSI 7-Layer에서 호스트 계층(Host layers)에 해당되는 4~7 계층에 해당됨
- ROS2에서는 운영 체제와 사용자 애플리케이션 사이에 있는 소프트웨어 계층으로 시스템의 다양한 구성 요소 쉽게 통신하고 데이터를 공유



■ DDS의 10가지 특징

1. Industry Standards
2. OS Independent
3. Language Independent
4. Transport on UDP/IP
5. Data Centricity
6. Dynamic Discovery
7. Scalable Architecture
8. Interoperability
9. Quality of Service (QoS)
10. Security

■ DDS의 10가지 특징

❖ 1. Industry Standards

- DDS
 - ✓ 분산 객체에 대한 기술 표준을 제정하기 위해 1989년에 설립된 비영리 단체인 OMG(Object Management Group, 객체 관리 그룹)가 관리하고 있는 만큼 산업 표준
 - ✓ OMG가 진행하여 ISO 승인된 표준으로는 UML, SysML, CORBA 등이 있음
- 2001년 DDS 표준화 작업이 진행되어 OpenFMB, Adaptive AUTOSAR, MD PnP, GVA, NGVA, ROS2와 같은 시스템에서 DDS 사용
- ROS1의 TCPROS는 독자적인 미들웨어였으나 ROS2에서는 DDS 사용으로 더 넓은 범위로 사용 가능하게 되었으며 산업 표준을 지키고 있는 만큼 로봇 운영체제 ROS가 IoT, 자동차, 국방, 항공, 우주 분야로 확장 가능함

■ DDS의 10가지 특징

❖ 2. OS Independent

- DDS는 Linux, Windows, macOS, Android, VxWorks 등 다양한 운영체제를 지원하고 있음
- 멀티 운영체제 지원이 가능하여 ROS2의 추진 방향에 적합

■ DDS의 10가지 특징

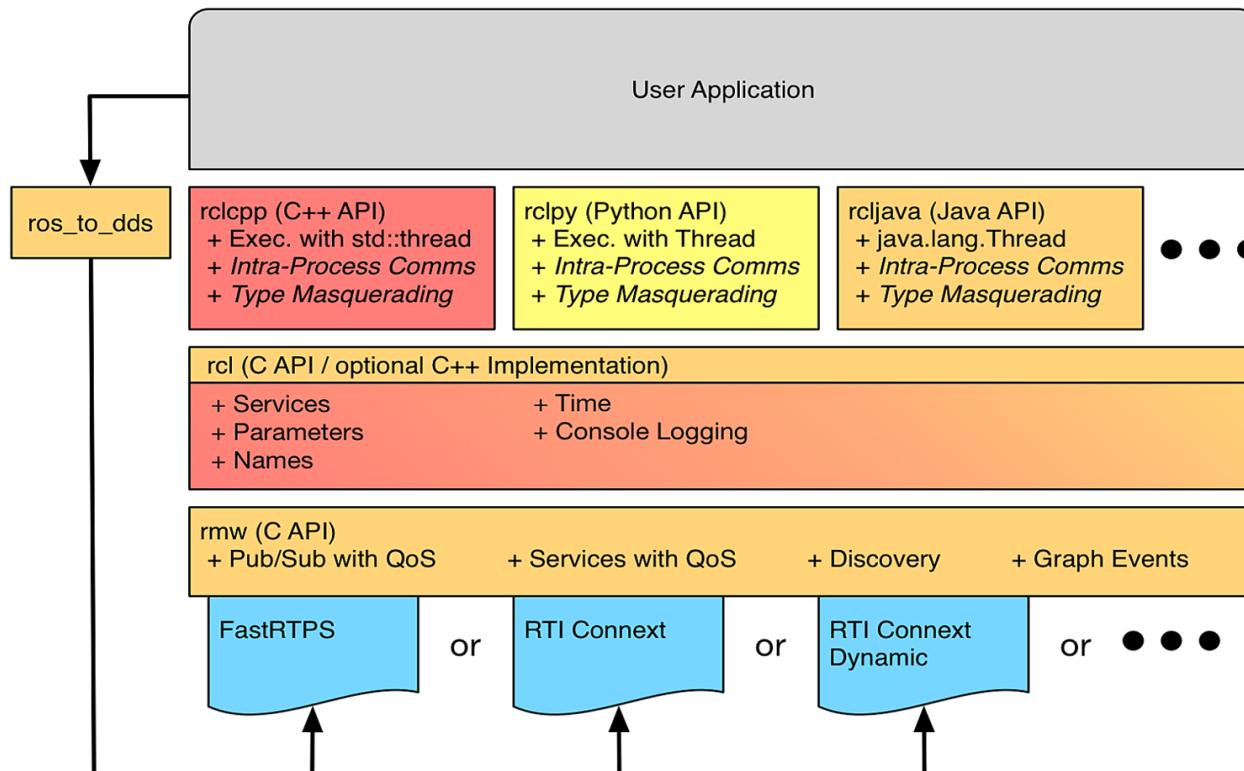
❖ 3. Language Independent

- DDS는 다양한 기업에서 통신 미들웨어 형태로 제공
 - ✓ ROS2를 지원하는 업체는 ADLink, Eclipse Foundation, Eprosima, Gurum Network, RTI로 총 5 곳
 - ✓ DDS 제품명으로는 ADLINK의 OpenSplice, Eclipse Foundation의 Cyclone DDS, Eprosima의 Fast DDS, Gurum Network의 Gurum DDS, RTI의 Connex DDS가 있음
- ROS2에서는 이러한 미들웨어를 유저가 원하는 사용 목적에 맞게 선택하여 사용할 수 있도록 ROS Middleware(RMW)형태로 지원
- RMW는 여러 DDS 구현의 지원을 위해 API의 추상화 인터페이스로 지원

■ DDS의 10가지 특징

❖ 3. Language Independent

- ROS2에서도 이런 장점을 살려 DDS를 RMW(ROS middleware)로 디자인



■ DDS의 10가지 특징

❖ 3. Language Independent

- 각 벤더 별로 RMW가 제작. 사용자 코드를 위해 rclcpp, rclc, rclpy, rcljava, rclobjc, rclada, rclgo, rclnodejs 같이 다양한 언어를 지원하는 ROS 클라이언트 라이브러리 (ROS Client Library)를 제작이 가능

■ DDS의 10가지 특징

❖ 4. Transport on UDP/IP

- DDS 벤더 별로 DDS Interoperability Wire Protocol (DDSI-RTPS)의 구현 방식에 따라 상이할 수 있음
- UDP 기반의 신뢰성 있는 멀티캐스트(reliable multicast)를 구현하여 시스템이 최신 네트워킹 인프라의 이점을 효율적으로 활용할 수 있음
- UDP 기반이라는 것이 ROS1에서의 TCPROS가 TCP 기반이었던 것에 비해 매우 큰 변임
- UDP의 멀티캐스트(multicast)는 브로드 캐스트(broadcast)처럼 여러 목적지로 동시에 데이터를 보낼 수 있지만, 불특정 목적지가 아닌 특정된 도메인 그룹에 대해서만 데이터를 전송

■ DDS의 10가지 특징

❖ 4. Transport on UDP/IP

- ROS2에서는 `ROS_DOMAIN_ID`라는 환경 변수로 도메인을 설정
- 멀티캐스트의 방식 도입으로 ROS2에서는 전역 공간이라 불리는 DDS Global Space이라는 공간에 있는 토픽들에 대해 구독 및 발행을 할 수 있음
- UDP와 TCP의 장단점은 QoS(Quality of Service)를 통해 보완 및 해결
- 참고로 일부 RMW 기능에는 TCP 기반으로 구현되는 경우도 있음

■ DDS의 10가지 특징

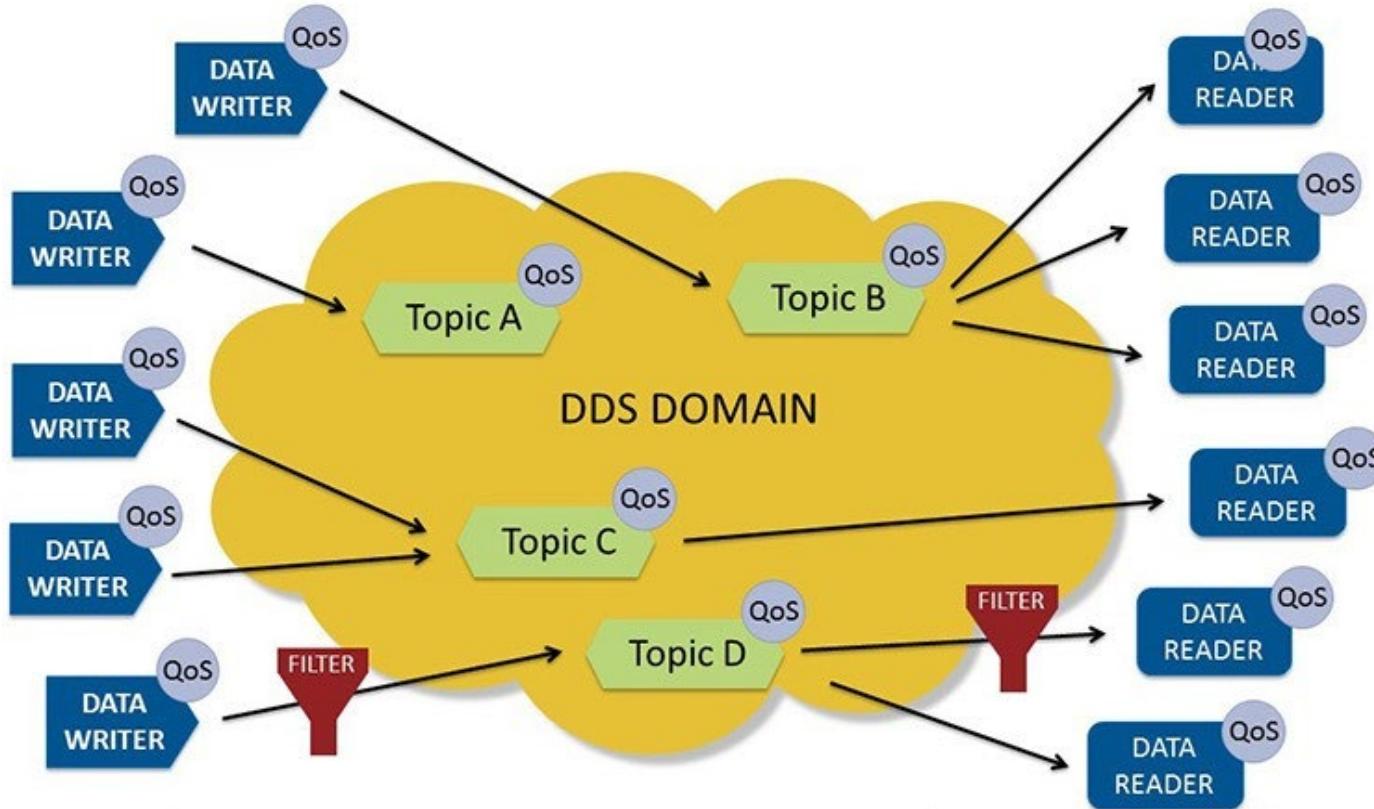
❖ 5. Data Centricity

- DDS의 특징은 `Data Centric`으로, 데이터 중심적으로 DDS 사양에도 DCPS(data-centric publish-subscribe)이라는 개념이 기술되어 있음
- 적절한 수신자에게 적절한 정보를 효율적으로 전달하는 것을 목표로 하는 발간 및 구독 방식임
- DDS의 미들웨어는 어떤 데이터인지, 데이터가 어떤 형식인지, 데이터를 어떻게 보낼 것인지, 데이터를 어떻게 안전하게 보낼 것인지에 대한 기능이 DDS 미들웨어 포함되어 있음

Data Distribution Service

■ DDS의 10가지 특징

❖ 5. Data Centricity



■ DDS의 10가지 특징

❖ 6. Dynamic Discovery

- DDS의 특징은 `Data Centric`으로, 데이터 중심적으로 DDS 사양에도 DCPS(data-centric publish-subscribe)이라는 개념이 기술되어 있음
- 적절한 수신자에게 적절한 정보를 효율적으로 전달하는 것을 목표로 하는 발간 및 구독 방식임
- DDS의 미들웨어는 어떤 데이터인지, 데이터가 어떤 형식인지, 데이터를 어떻게 보낼 것인지, 데이터를 어떻게 안전하게 보낼 것인지에 대한 기능이 DDS 미들웨어 포함되어 있음

■ DDS의 10가지 특징

❖ 7. Scalable Architecture

- DDS 아키텍처는 IoT 디바이스와 같은 소형 디바이스부터 인프라, 국방, 항공, 우주 산업과 같은 초대형 시스템으로 까지 확장할 수 있도록 설계되었음
- DDS의 Participant 형태의 노드는 확장 가능한 형태로 제공됨
 - ✓ 단일 표준 통신 계층에서 분산 시스템 개발을 더욱 단순화 시켜 개발이 편리함
 - ✓ ROS와 같이 최소 실행 가능한 노드 단위로 나누어 수백, 수천 개의 노드를 관리해야 하는 시스템에 적합함
 - ✓ ROS 시스템을 복수의 로봇, 주변 인프라와 다양한 IT 기술, 데이터베이스, 클라우드로 연결 및 확장 가능

■ DDS의 10가지 특징

❖ 8. Interoperability

- DDS는 상호 운용성을 지원
 - ✓ DDS의 표준 사양을 지키고 있는 벤더들의 제품을 혼용하여도 적용할 수 있음
- 대표적인 DDS 벤더들 중 ROS2를 지원하는 업체는 ADLink, Eclipse Foundation, Eprosima, Gurum Network, RTI로 총 5 곳이며
 - ✓ ADLINK의 OpenSplice
 - ✓ Eclipse Foundation의 Cyclone DDS (Open Source)
 - ✓ Eprosima의 Fast DDS(Open Source)
 - ✓ Gurum Network의 Gurum DDS,
 - ✓ RTI의 Connext DDS

Data Distribution Service

■ DDS의 10가지 특징

❖ 8. Interoperability

OMG Member Vendors



■ DDS의 10가지 특징

❖ 9. Quality of Service (QoS)

- DDS 도입으로 유저가 목적에 맞게 데이터의 송수신 설정을 할 수 있으며, 이를 QoS(Quality of Service)라 하며, 발행 및 구독 등을 선언하고 사용할 때 매개 변수처럼 QoS를 사용할 수 있음
- DDS 사양 상 설정 가능한 QoS 항목은 22가지이나, ROS2에서는 데이터 손실을 방지하는(reliable), 통신 속도를 우선시 하는(best effort) 기능이 대표적으로 사용되고 있음
 - ✓ 그 외에 History, Durability, Deadline, Lifespan, Liveliness 등이 있음
- 다양한 QoS 설정을 통해 DDS는 적시성, 트래픽 우선순위, 안정성 및 리소스 사용과 같은 요소를 프로그래밍을 통해 제어할 수 있게 됨

■ DDS의 10가지 특징

❖ 9. Quality of Service (QoS)

- QoS는 데이터 통신 옵션으로 ROS2에서는 TCP, UDP 방식을 선택적으로 사용할 수 있음
 - ✓ 데이터 전송의 실시간성, 지속성, 중복성과 관련된 옵션 설정 가능
 - ✓ History :데이터를 몇 개나 보관할지 결정
 - ✓ Reliability : 데이터 전송에서 속도를 우선하는지 신뢰성을 우선하는지를 결정
 - ✓ Durability : 데이터를 수신하는 Subscribe가 생성되기 전의 데이터를 사용 여부 결정
 - ✓ Deadline : 정해진 주기 안에 데이터가 발신, 수신되지 않을 경우 이벤트 함수 실행
 - ✓ Lifespan : 정해진 주기 안에서 수신되는 데이터만 유효 판정, 그렇지 않은 데이터는 삭제
 - ✓ Liveliness : 정해진 주기 안에서 노드 혹은 토픽의 생사를 확인

■ DDS의 10가지 특징

❖ 10. Security

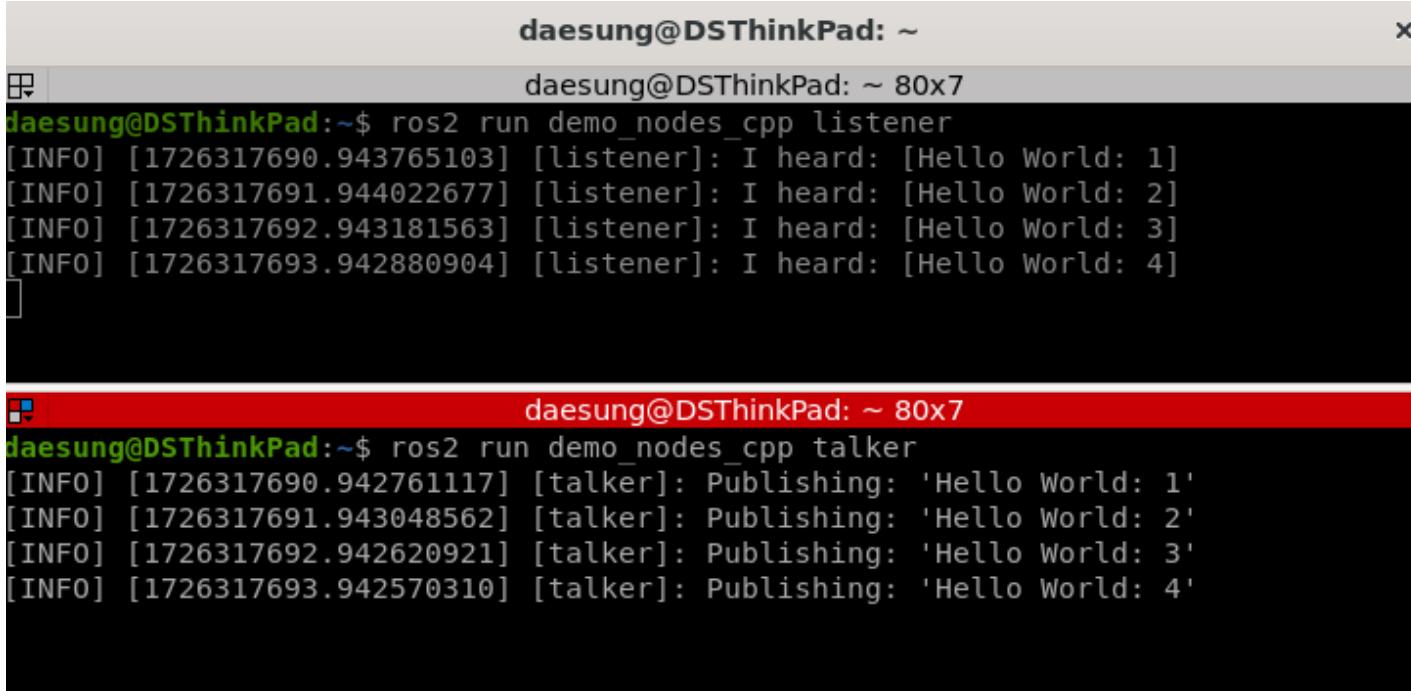
- DDS의 사양에는 DDS-Security이라는 DDS 보안 사양을 ROS에 적용하여 보안에 대한 이슈를 통신단부터 해결
- ROS 커뮤니티에서는 SROS2(Secure Robot Operating System 2)라는 툴을 개발
- 보안 관련 RCL 서포트 및 보안 프로그래밍에 익숙지 않은 로보틱스 개발자를 위해 보안을 위한 툴킷을 만들어 배포하고 있음

Data Distribution Service

DDS Demo

❖ Demo Publisher / Subscriber with C++

- CPP demo talker → CPP demo listener



The screenshot shows two terminal windows on a Linux system. The top window is titled "daesung@DSThinkPad: ~" and has a light gray background. It displays the command "ros2 run demo_nodes_cpp listener" followed by four INFO messages from the "listener" node: "[INFO] [1726317690.943765103] [listener]: I heard: [Hello World: 1]", "[INFO] [1726317691.944022677] [listener]: I heard: [Hello World: 2]", "[INFO] [1726317692.943181563] [listener]: I heard: [Hello World: 3]", and "[INFO] [1726317693.942880904] [listener]: I heard: [Hello World: 4]". The bottom window is also titled "daesung@DSThinkPad: ~" and has a red header bar. It displays the command "ros2 run demo_nodes_cpp talker" followed by four INFO messages from the "talker" node: "[INFO] [1726317690.942761117] [talker]: Publishing: 'Hello World: 1'", "[INFO] [1726317691.943048562] [talker]: Publishing: 'Hello World: 2'", "[INFO] [1726317692.942620921] [talker]: Publishing: 'Hello World: 3'", and "[INFO] [1726317693.942570310] [talker]: Publishing: 'Hello World: 4'". Both windows show the user's name "daesung" and the host "DSThinkPad" at the prompt.

```
daesung@DSThinkPad:~$ ros2 run demo_nodes_cpp listener
[INFO] [1726317690.943765103] [listener]: I heard: [Hello World: 1]
[INFO] [1726317691.944022677] [listener]: I heard: [Hello World: 2]
[INFO] [1726317692.943181563] [listener]: I heard: [Hello World: 3]
[INFO] [1726317693.942880904] [listener]: I heard: [Hello World: 4]

daesung@DSThinkPad:~$ ros2 run demo_nodes_cpp talker
[INFO] [1726317690.942761117] [talker]: Publishing: 'Hello World: 1'
[INFO] [1726317691.943048562] [talker]: Publishing: 'Hello World: 2'
[INFO] [1726317692.942620921] [talker]: Publishing: 'Hello World: 3'
[INFO] [1726317693.942570310] [talker]: Publishing: 'Hello World: 4'
```

■ DDS 사용 예시

❖ RMW(ROS Middleware) 변경

- RMW_IMPLEMENTATION 변수로 RMW 변경 가능
- rmw_cyclonedds_cpp
- rmw_fastrtps_cpp
- rmw_connex_cpp
- rmw_gurumdds_cpp

Data Distribution Service

■ DDS 사용 예시

❖ RMW Interoperability

The image shows two terminal windows side-by-side, both titled "daesung@DSThinkPad: ~".

Top Terminal:

```
daesung@DSThinkPad:~$ export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
daesung@DSThinkPad:~$ ros2 run demo_nodes_cpp listener
[INFO] [1726321069.077129032] [listener]: I heard: [Hello World: 1]
[INFO] [1726321070.076433730] [listener]: I heard: [Hello World: 2]
[INFO] [1726321071.076209668] [listener]: I heard: [Hello World: 3]
[INFO] [1726321072.076097277] [listener]: I heard: [Hello World: 4]
```

Bottom Terminal:

```
daesung@DSThinkPad:~$ export RMW_IMPLEMENTATION=rmw_fastrtps_cpp
daesung@DSThinkPad:~$ ros2 run demo_nodes_cpp talker
[INFO] [1726321069.076213597] [talker]: Publishing: 'Hello World: 1'
[INFO] [1726321070.075811119] [talker]: Publishing: 'Hello World: 2'
[INFO] [1726321071.075798058] [talker]: Publishing: 'Hello World: 3'
[INFO] [1726321072.075727431] [talker]: Publishing: 'Hello World: 4'
```

In both terminals, the command `export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp` or `rmw_fastrtps_cpp` is highlighted with a red box, followed by the command `ros2 run demo_nodes_cpp` and either `listener` or `talker`.

Data Distribution Service

■ DDS 사용 예시

❖ Domain 변경

The screenshot displays three terminal windows on a Linux system, each showing the execution of ROS nodes. The top window shows a 'talker' node publishing messages. The middle window shows the activation of the 'humble' ROS distribution and the execution of a 'listener' node. The bottom window shows another 'listener' node receiving messages from the 'talker'. All command lines and output messages are highlighted with red boxes.

```
daesung@DSThinkPad: ~
daesung@DSThinkPad:~$ export ROS_DOMAIN_ID=11
daesung@DSThinkPad:~$ ros2 run demo_nodes_cpp talker
[INFO] [1726321663.590614075] [talker]: Publishing: 'Hello World: 1'
[INFO] [1726321664.590892112] [talker]: Publishing: 'Hello World: 2'

daesung@DSThinkPad: ~ 80x5
daesung@DSThinkPad:~$ humble
ROS2 Humble is activated!
ROS_Domain is 23
daesung@DSThinkPad:~$ ros2 run demo_nodes_cpp listener

daesung@DSThinkPad: ~ 80x7
daesung@DSThinkPad:~$ export ROS_DOMAIN_ID=11
daesung@DSThinkPad:~$ ros2 run demo_nodes_cpp listener
[INFO] [1726321663.591261132] [listener]: I heard: [Hello World: 1]
[INFO] [1726321664.591626064] [listener]: I heard: [Hello World: 2]
```

3 ROS2 Basics

❖ 패키지 설치 및 확인

- sudo apt update
- sudo apt install <pkg-name>
 - ✓ ex) sudo apt install ros-humble-turtlesim
- ros2 pkg list
- ros2 pkg executables turtlesim

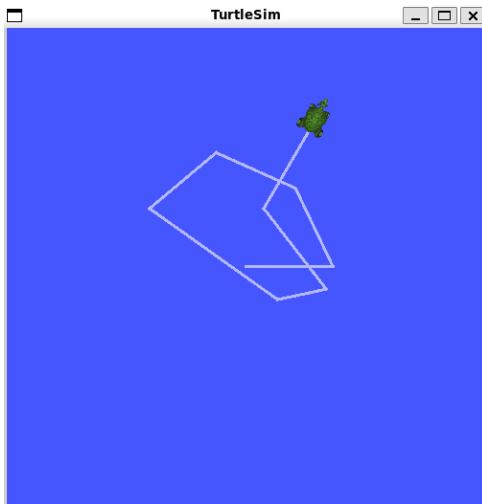
```
daesung@DSThinkPad: ~
daesung@DSThinkPad: ~ 80x56
daesung@DSThinkPad:~$ ros2 pkg list
action_msgs
action_tutorials_cpp
action_tutorials_interfaces
action_tutorials_py
actionlib_msgs
ament_cmake
ament_cmake_auto
ament_cmake_copyright
ament_cmake_core
ament_cmake_cppcheck
ament_cmake_cpplint
ament_cmake_export_definitions
ament_cmake_export_dependencies
ament_cmake_export_include_directories
ament_cmake_export_interfaces
ament_cmake_export_libraries
ament_cmake_export_link_flags
ament_cmake_export_targets
ament_cmake_flake8
```

```
daesung@DSThinkPad: ~
daesung@DSThinkPad: ~ 80x56
daesung@DSThinkPad:~$ ros2 pkg executables turtlesim
turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node
daesung@DSThinkPad:~$
```

❖ 패키지의 노드 실행

```
daesung@DSThinkPad: ~
daesung@DSThinkPad: ~ 80x56
daesung@DSThinkPad:~$ ros2 pkg executables turtlesim
turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node
daesung@DSThinkPad:~$
```

- ros2 run turtlesim turtlesim_node

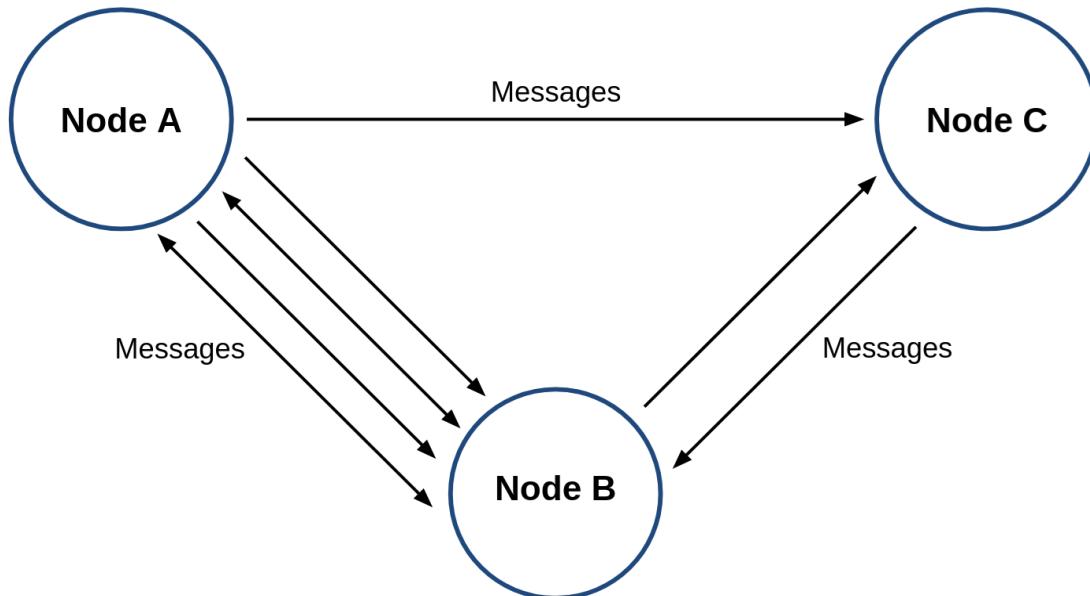


- ros2 run turtlesim turtle_teleop_key

```
daesung@DSThinkPad: ~ 64x13
daesung@DSThinkPad:~$ 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 rotation.
'Q' to quit.
```

❖ 노드와 메시지 통신

- 노드들은 서로 유기적으로 Message로 연결되어 사용됨
- 수행하고자 하는 태스크가 많아질수록 메시지로 연결되는 노드가 늘어나며 시스템이 확장



❖ ROS Interface

- ROS의 노드 간에 데이터를 주고받을 때에는 토픽, 서비스, 액션이 사용되는데 이 때 사용되는 데이터의 형태
- ROS 인터페이스에는 ROS 2에 새롭게 추가된 IDL(interface definition language)과 ROS 1부터 ROS 2까지 널리 사용 중인 msg, srv, action이 있음
- 토픽, 서비스, 액션은 각각 msg, srv, action에 대한 interface를 사용하고 있으며 Integer, Floating Point, Boolean과 같은 단순 자료형을 기본 단위로 메시지 안에 메시지를 가진 간단한 데이터 구조 및 메시지들이 나열된 배열과 같은 구조로도 사용하고 있음

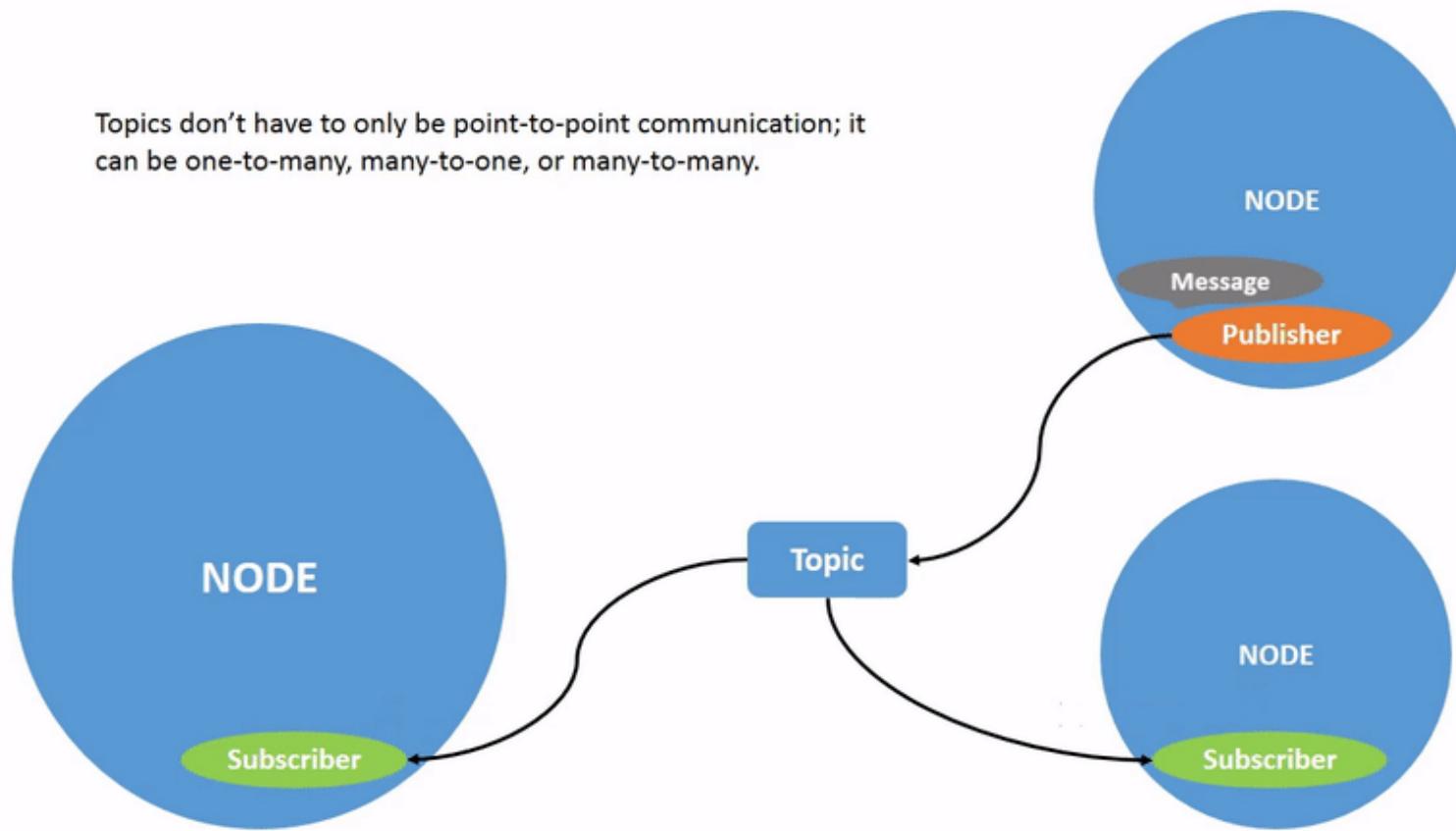
❖ ROS Interface

- [단순 자료형]
 - ✓ 정수(integer), 부동 소수점(floating point), 불(boolean)
 - ✓ https://github.com/ros2/common_interfaces/tree/foxy/std_msgs
- [메시지 안에 메시지를 품고 있는 간단한 데이터 구조]
 - ✓ geometry_msgs-msgs/Twist의 `Vector3 linear`
 - ✓ https://github.com/ros2/common_interfaces/blob/foxy/geometry_msgs/msg/Twist.msg
- [메시지들이 나열된 배열과 같은 구조]
 - ✓ sensor_msgs-msgs/LaserScan 의 `float32[] ranges`
 - ✓ https://github.com/ros2/common_interfaces/blob/foxy/sensor_msgs/msg/LaserScan.msg

4 TOPIC

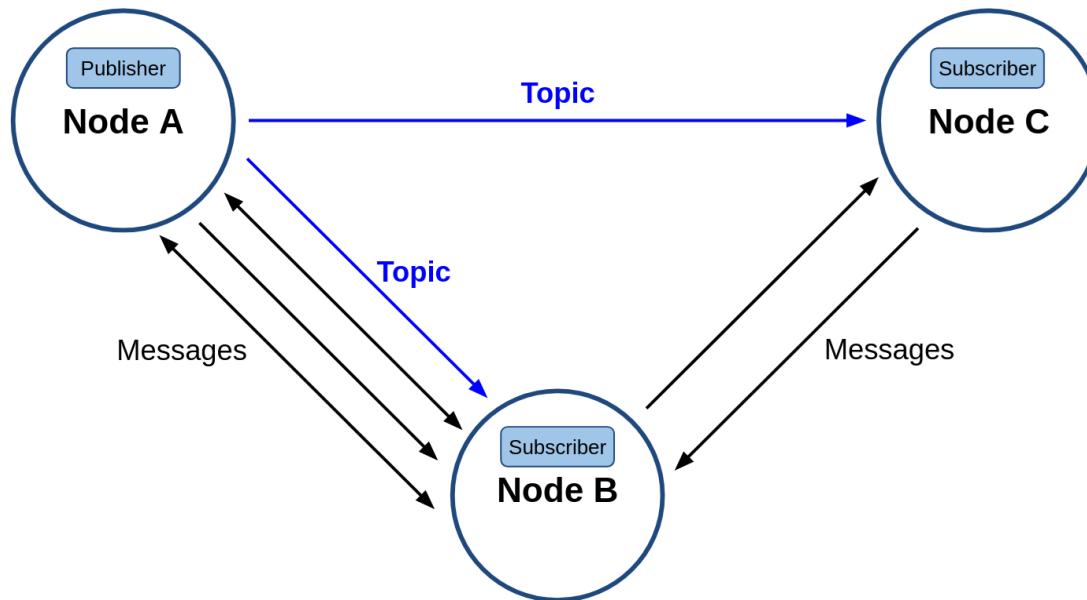
❖ 노드 간 Publish & Subscribe

Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.



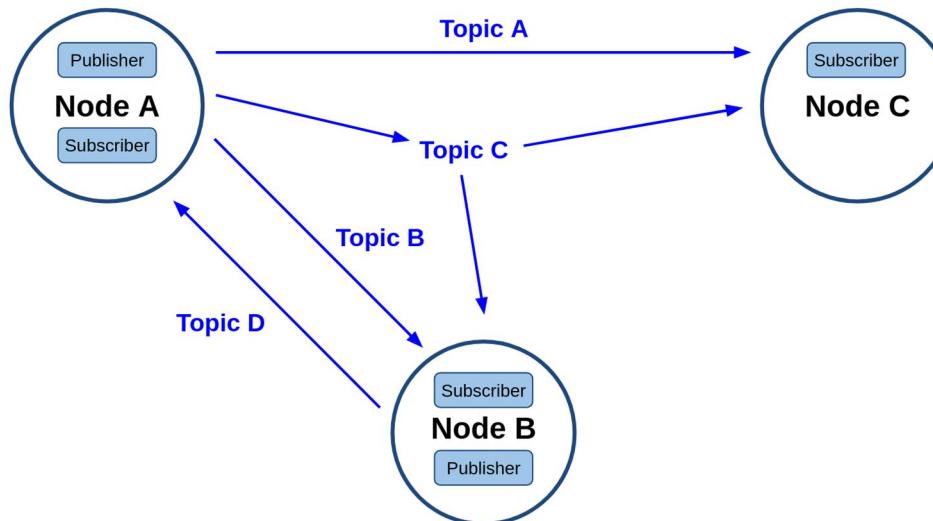
❖ Topic 사용 방법

- 비동기식 단방향 메시지 송수신 방식으로 msg 메시지 형태의 메시지를 발간하는 Publisher와 메시지를 구독하는 Subscriber 간의 통신
- 1:N, N:1, N:N 통신도 가능하며 ROS 메시지 통신에서 가장 널리 사용



❖ Topic 사용 방법

- Node A 처럼 하나의 이상의 토픽을 발행할 수 있을 뿐만이 아니라 `Publisher` 기능과 `Subscriber` 역할을 동시에 수행 가능하며, 자신 토픽을 셀프 구독할 수도 있음
- 통상적으로 70% 이상이 토픽방식을 사용. 기본 특징으로 비동기성과 연속성을 가져서 센서 값 전송 등 항상 정보를 주고 받는 노드 부분에 주로 사용

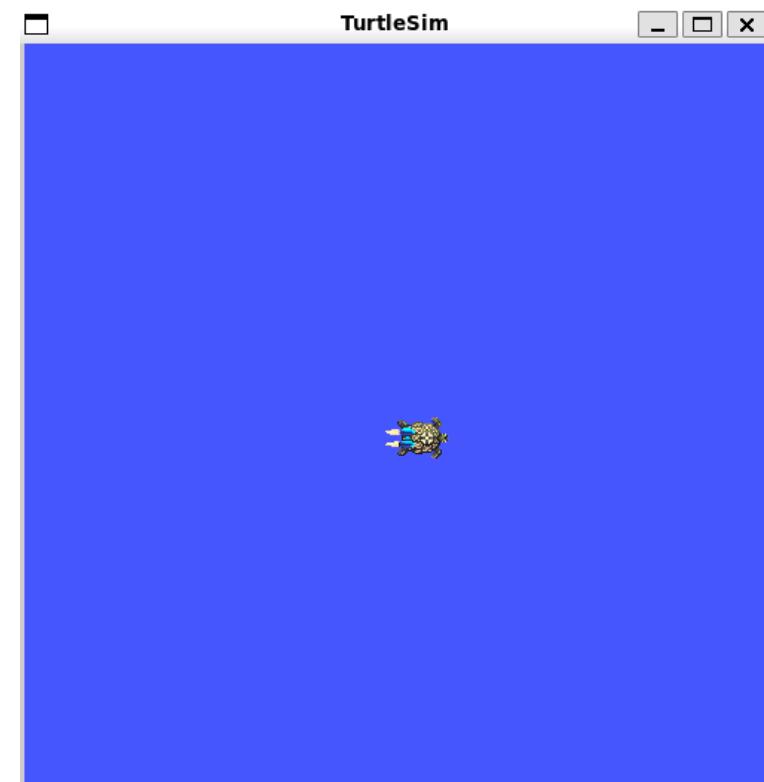


❖ Topic List

- Turtlesim 실행 후 topic list 조회

```
daesung@DSThinkPad: ~
daesung@DSThinkPad: ~ 61x13
(base) daesung@DSThinkPad:~$ humble
ROS2 Humble is activated!
ROS_Domain is 23
(base) daesung@DSThinkPad:~$ ros2 run turtlesim turtlesim_node
QStandardPaths: wrong permissions on runtime directory /run/user/1000/, 0755 instead of 0700
[INFO] [1725799900.510379729] [turtlesim]: Starting turtlesim
with node name /turtlesim
[INFO] [1725799900.513816222] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
[

daesung@DSThinkPad: ~ 61x12
(base) daesung@DSThinkPad:~$ humble
ROS2 Humble is activated!
ROS_Domain is 23
(base) daesung@DSThinkPad:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
(base) daesung@DSThinkPad:~$
```



❖ Topic Information

- Turtlesim 실행 후 노드 정보 및 topic 정보 조회

```
daesung@DSThinkPad:~$ 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
  (기타 자료 생략)
```

❖ Topic List & Type

- Turtlesim 실행 후 topic list 조회 및 type 확인

```
daesung@DSThinkPad: ~ 61x12
(base) daesung@DSThinkPad:~$ humble
ROS2 Humble is activated!
ROS Domain is 23
(base) daesung@DSThinkPad:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
(base) daesung@DSThinkPad:~$ ros2 topic type /turtle1/pose
turtlesim/msg/Pose
(base) daesung@DSThinkPad:~$
```

❖ Topic Data

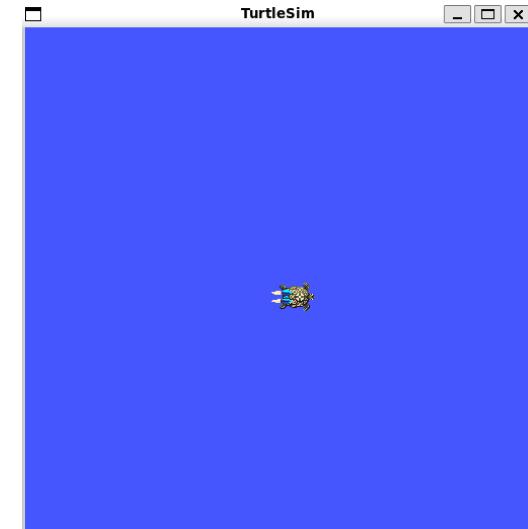
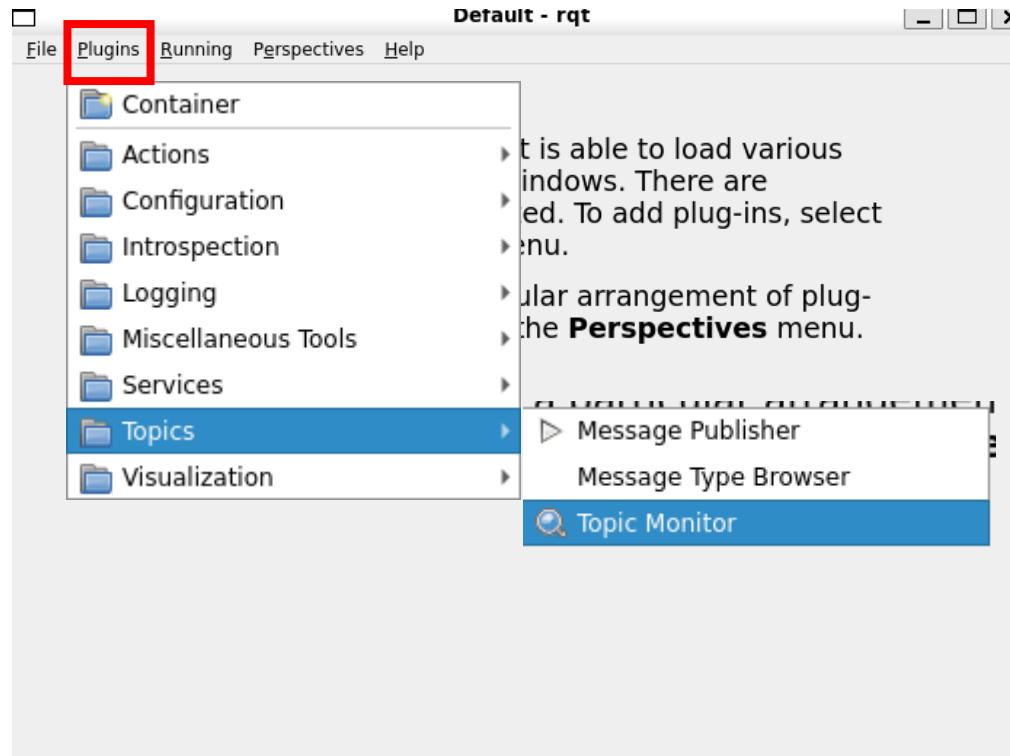
- Turtle1이 발행하는 pose Topic 확인을 위해 turtlesim/msg/Pose 확인
- ros2 interface show turtlesim/msg/Pose

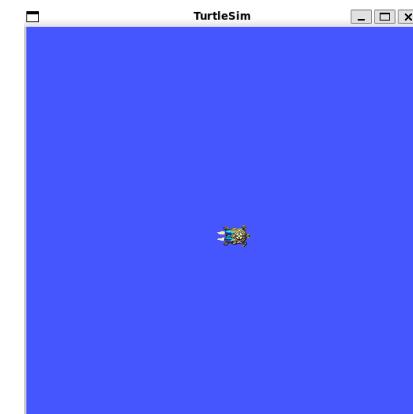
```
[daesung@DSThinkPad: ~ 61x12]
Type: turtlesim/msg/Pose
Publisher count: 1
Subscription count: 0
(base) daesung@DSThinkPad:~$ ros2 interface show turtlesim/ms
g/Pose
float32 x
float32 y
float32 theta

float32 linear_velocity
float32 angular_velocity
(base) daesung@DSThinkPad:~$
```

❖ Topic with rqt

- Turtlesim 실행 후 rqt를 이용한 Topic 조회
- Plugins → Topics → Topic Monitor





❖ Topic List & Type (rqt)

- Topic list 및 topic type 확인 가능

Topic Monitor

Topic	Type	Bandwidth
/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	
/turtle1/rotate_absolute/_action/feedback	turtlesim/action/RotateAbsolute_FeedbackMessage	
/turtle1/rotate_absolute/_action/status	action_msgs/msg/GoalStatusArray	

- Plugins → Topics → Message Type Browser

Default - rqt

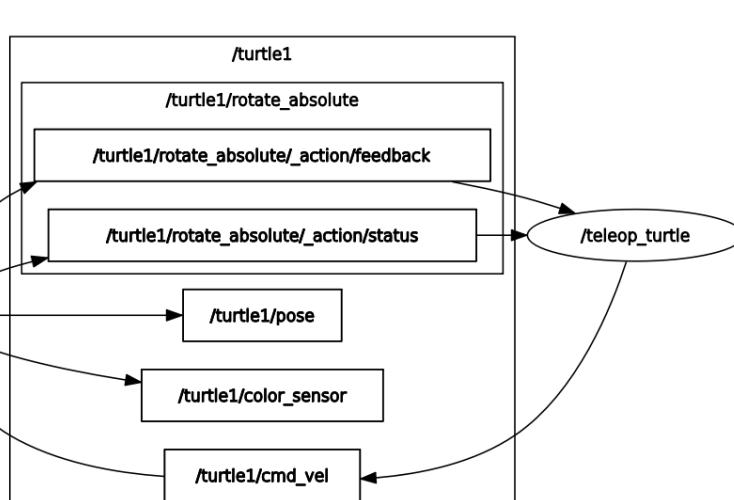
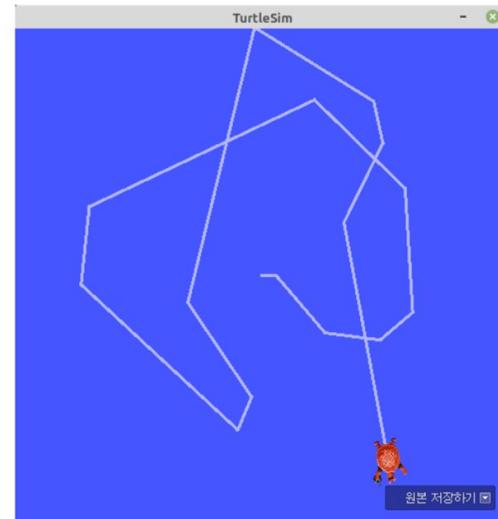
Message Type Browser

Message: turtlesim

Tree	Type	Path
Msg Root	turtlesim/Pose	turtlesim/Pose
x	float	turtlesim/Pose/x
y	float	turtlesim/Pose/y
theta	float	turtlesim/Pose/theta
linear_velocity	float	turtlesim/Pose/linear_velocity
angular_velocity	float	turtlesim/Pose/angular_velocity

❖ Topic - example

- ros2 run turtlesim turtlesim_node
- ros2 run turtlesim turtle_teleop_key
- rqt_graph



```
daesung@DSThinkPad: ~ 50x9
daesung@DSThinkPad:~$ 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 rotation.
'Q' to quit.

daesung@DSThinkPad: ~ 60x25
daesung@DSThinkPad:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
daesung@DSThinkPad:~$
```

■ Node & Topic

❖ Node information

- /turtlesim

```
daesung@DSThinkPad: ~ 68x41
daesung@DSThinkPad:~$ 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/SetParametersAtomically
  Service Clients:
  Action Servers:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
  Action Clients:
```

■ Node & Topic

❖ Node information

- /teleop_turtle

```
daesung@DSThinkPad: ~ 68x41
daesung@DSThinkPad:~$ ros2 node info /teleop_turtle
/teleop_turtle
Subscribers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
Publishers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /rosout: rcl_interfaces/msg/Log
  /turtle1/cmd_vel: geometry_msgs/msg/Twist
Service Servers:
  /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
Service Clients:
Action Servers:
Action Clients:
  /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
```

❖ Topic 상태조회 명령

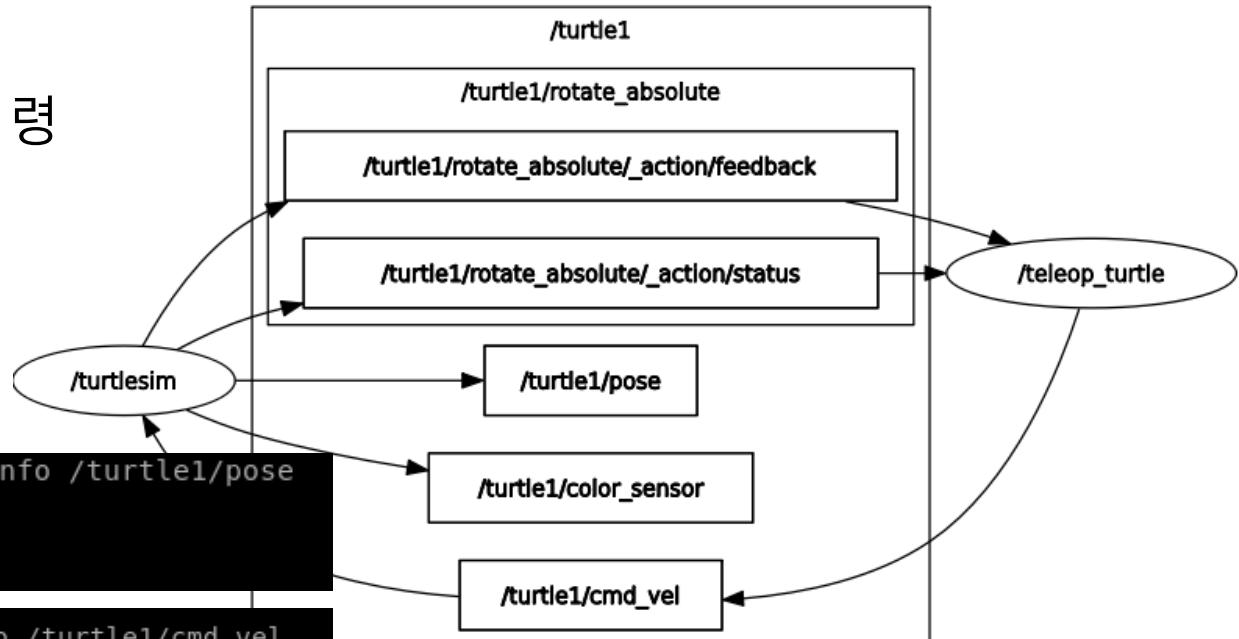
- topic info
- topic echo
- topic bw
- topic hz

```
daesung@DSThinkPad:~$ ros2 topic info /turtle1/pose
Type: turtlesim/msg/Pose
Publisher count: 1
Subscription count: 0
```

```
daesung@DSThinkPad:~$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 1
```

```
daesung@DSThinkPad:~$ ros2 topic bw /turtle1/cmd_vel
Subscribed to [/turtle1/cmd_vel]
48 B/s from 2 messages
    Message size mean: 52 B min: 52 B max: 52 B
33 B/s from 2 messages
    Message size mean: 52 B min: 52 B max: 52 B
```

```
daesung@DSThinkPad:~$ ros2 topic hz /turtle1/cmd_vel
average rate: 0.495
min: 1.372s max: 2.666s std dev: 0.64686s window: 2
```

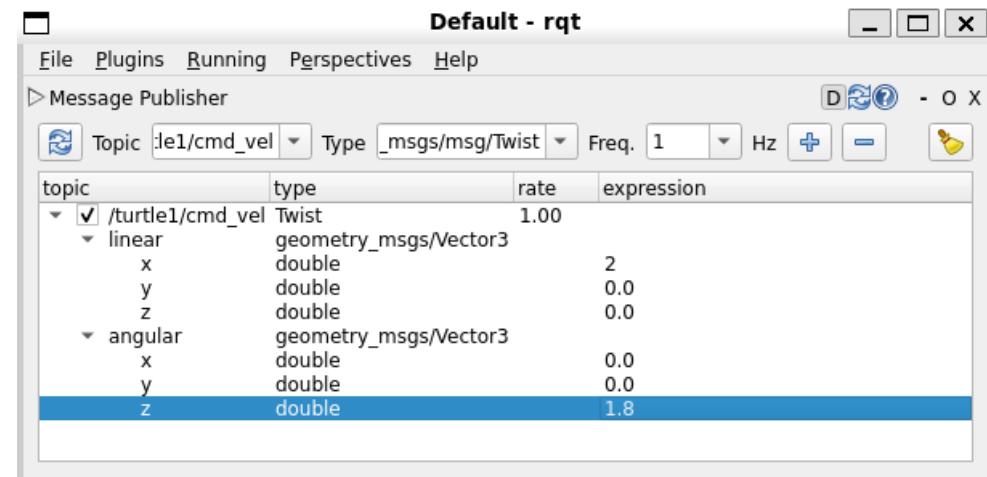
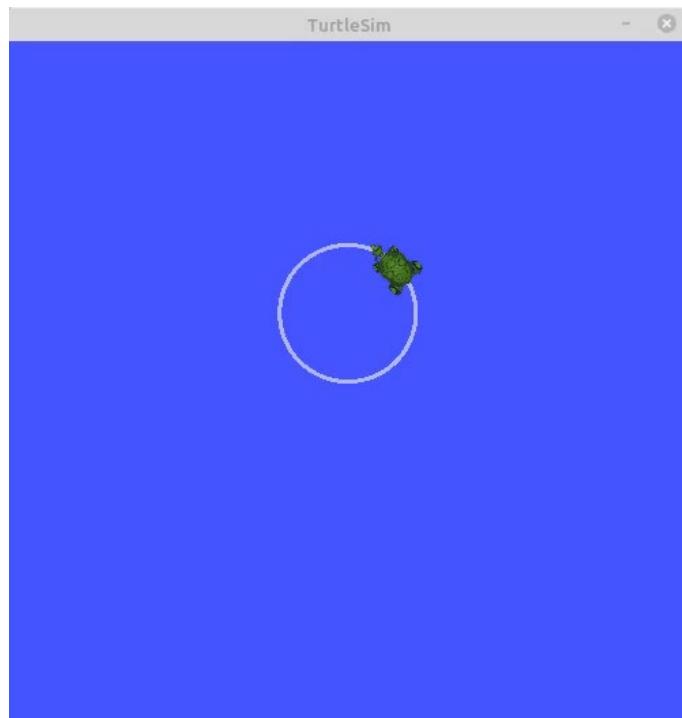


```
daesung@DSThinkPad:~$ ros2 topic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
```

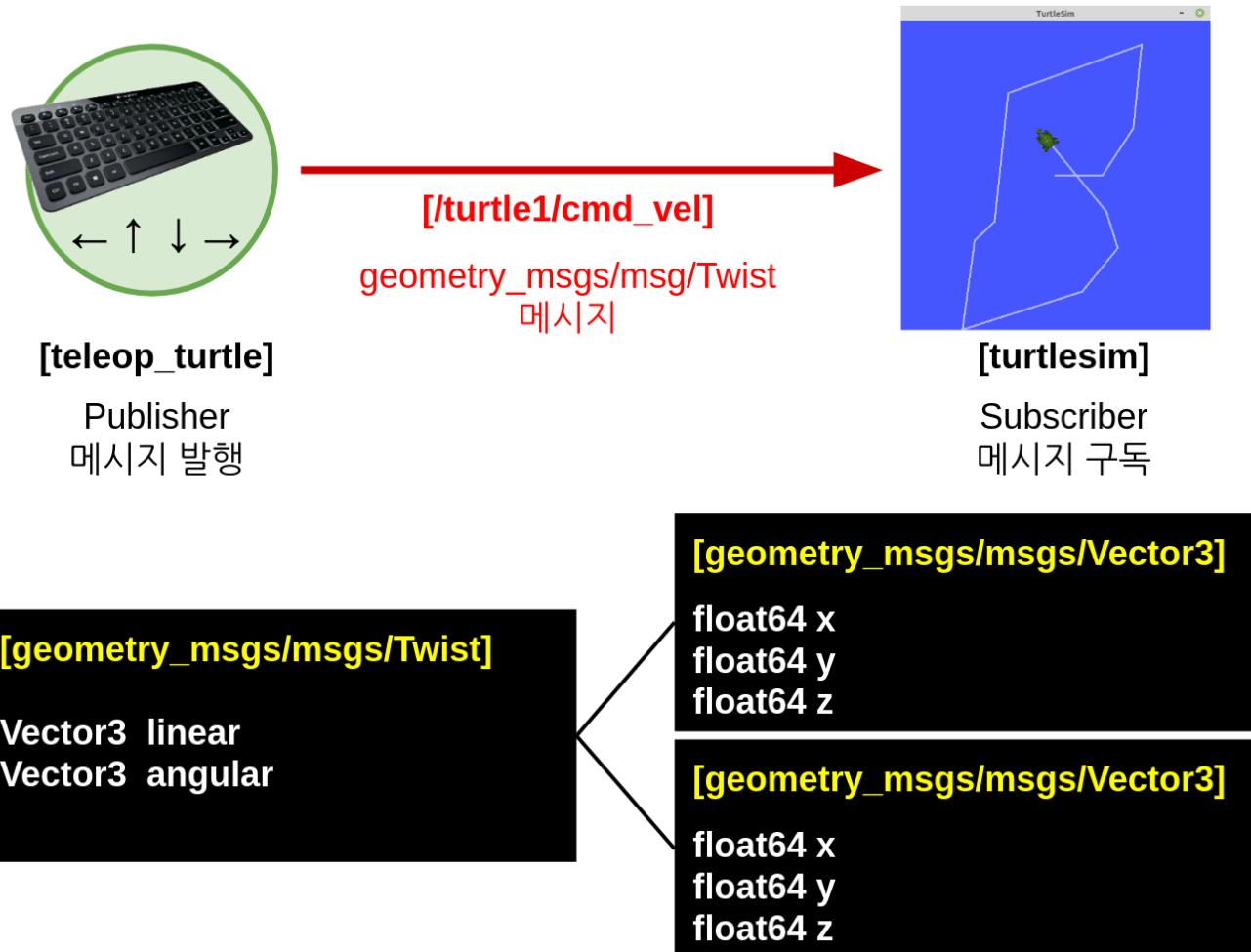
■ Topic 발행

❖ ros2 topic pub

```
$ ros2 topic pub --rate 1 /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}}"
```



❖ Message Interface



❖ Message Interface

- 토픽은 고유의 인터페이스를 가지고 있음
 - ✓ 메시지 인터페이스라 부르며, msg 파일
 - ✓ /turtle1/cmd_vel Topic은 geometry_msgs/msg/Twist 형태
- Twist 데이터 형태를 보면 Vector3 linear와 Vector3 angular로 구성
 - ✓ 메시지 안에 메시지를 품고 있는 것으로 Vector3 형태에 linear이라는 이름의 메시지와 Vector3 형태에 angular이라는 이름의 메시지, 즉 2개의 메시지가 존재
 - ✓ Vector3는 다시 float64 형태에 x, y, z 값이 존재함
- geometry_msgs/msg/Twist 메시지 형태는 float64 자료형의 linear.x, linear.y, linear.z, angular.x, angular.y, angular.z라는 이름의 메시지
 - ✓ 병진 속도 3개, 회전 속도 3개를 표현

❖ Message Interface

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

```
Vector3 linear
```

```
Vector3 angular
```

```
$ ros2 interface show geometry_msgs/msg/Vector3
```

```
float64 x
```

```
float64 y
```

```
float64 z
```

❖ Message Interface

```
$ ros2 interface list
```

Messages:

action_msgs/msg/GoalInfo
action_msgs/msg/GoalStatus
action_msgs/msg/GoalStatusArray

(생략)

Services:

action_msgs/srv/CancelGoal
composition_interfaces/srv/ListNodes

(생략)

Actions:

action_tutorials_interfaces/action/Fibonacci
example_interfaces/action/Fibonacci

(생략)

❖ Message Interface

```
$ ros2 interface packages  
action_msgs  
action_tutorials_interfaces  
actionlib_msgs  
builtin_interfaces  
(생략)
```

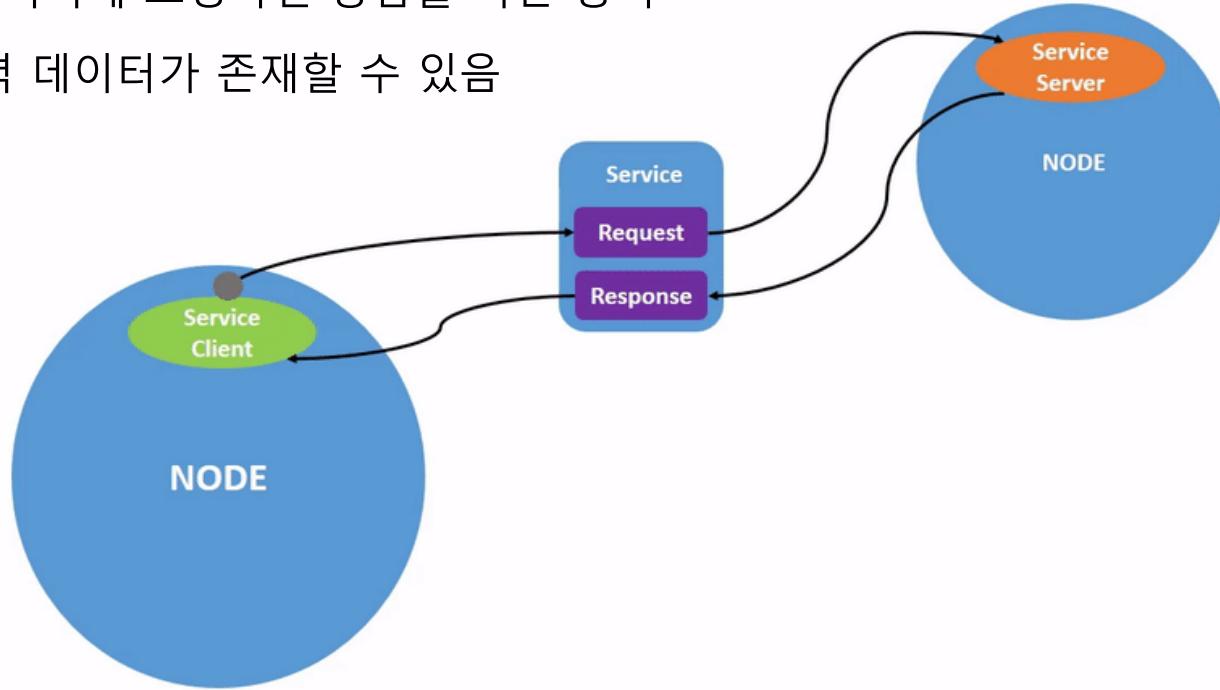
```
$ ros2 interface package turtlesim  
turtlesim/srv/Spawn  
turtlesim/msg/Color  
turtlesim/msg/Pose  
turtlesim/srv/TeleportAbsolute  
turtlesim/srv/Kill  
turtlesim/action/RotateAbsolute  
turtlesim/srv/SetPen  
turtlesim/srv/TeleportRelative
```

5 Service

■ 노드와 메시지 통신

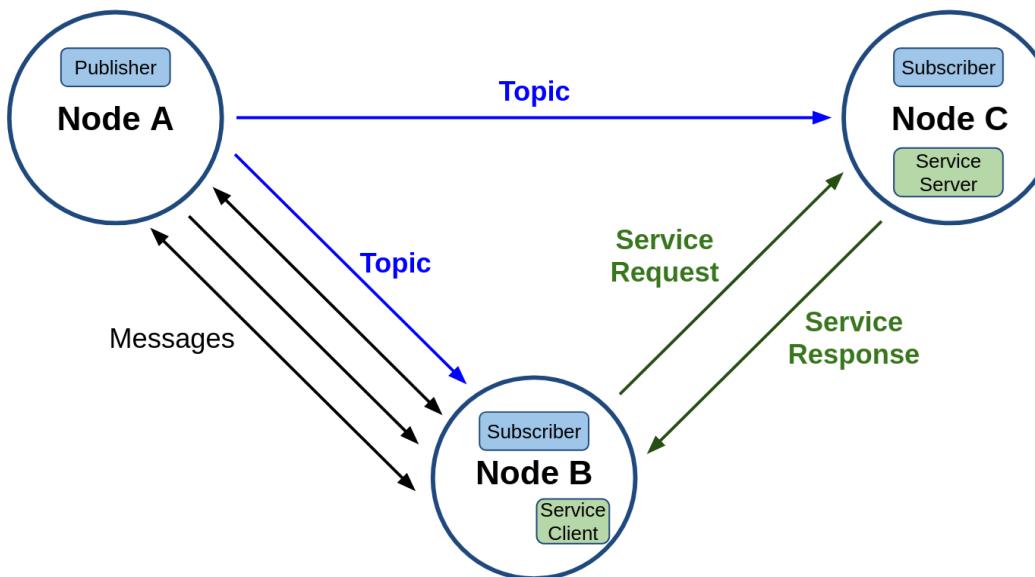
❖ Service

- 두개의 노드가 데이터를 주고 받는 방식
- 클라이언트가 서버에 요청하면 응답을 하는 방식
- 입력 또는 출력 데이터가 존재할 수 있음



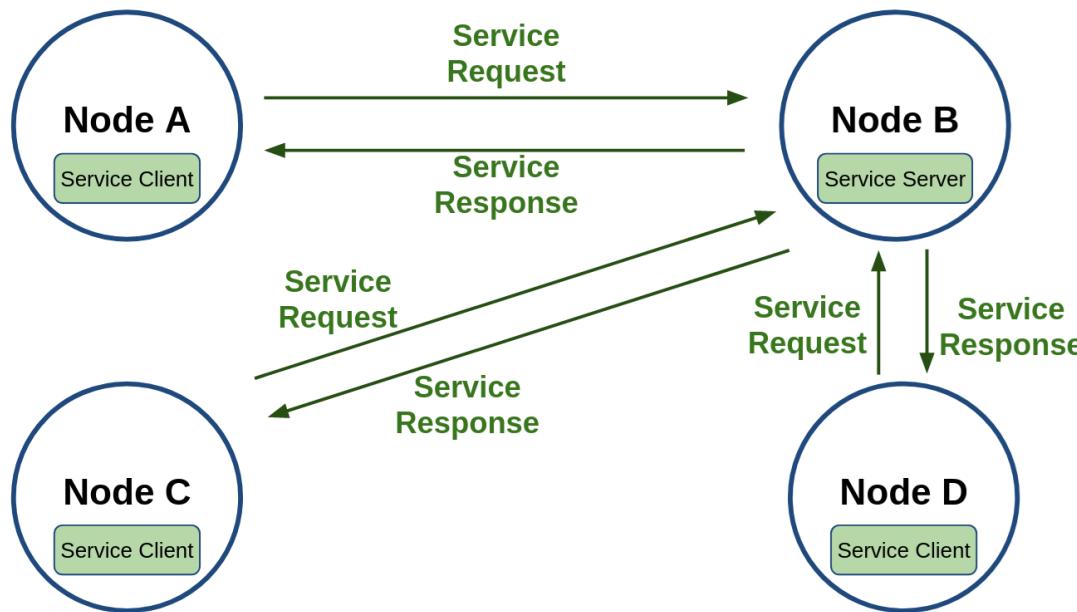
❖ Service

- 동기식 양방향 메시지 송수신 방식으로 서비스의 요청(Request)을 하는 쪽을 Service client, 서비스의 응답(Response)을 하는 쪽을 Service server라 함
- 서비스는 작업을 요청 하는 클라이언트, 요청 작업을 수행 후 결과를 전달하는 서버 간의 통신. 요청 및 응답(Request/Response) 또한 앞서 언급한 메시지의 변형으로 srv 메시지라 함



❖ Service

- 동일 서비스에 대해 복수의 클라이언트를 가질 수 있도록 설계
 - ✓ 서비스 응답은 서비스 요청이 있었던 서비스 클라이언트에 대해서만 응답
 - ✓ Node C의 Service Client가 Node B의 Service Server에게 서비스 요청을 했다면 Node B의 Service Server는 요청 받은 서비스를 수행한 후 Node C의 Service Client에게만 서비스 응답



❖ Service 목록 확인

```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 service list
/clear
/kill
/reset
/spawn
/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
daesung@DSThinkPad:~$
```

❖ Service 형태 확인

```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 service type /clear
std_srvs/srv/Empty
daesung@DSThinkPad:~$ ros2 service type /kill
turtlesim/srv/Kill
daesung@DSThinkPad:~$ ros2 service type /spawn
turtlesim/srv/Spawn
daesung@DSThinkPad:~$ ros2 service list -t
/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/SetParametersAtomically]
```

❖ Service 찾기

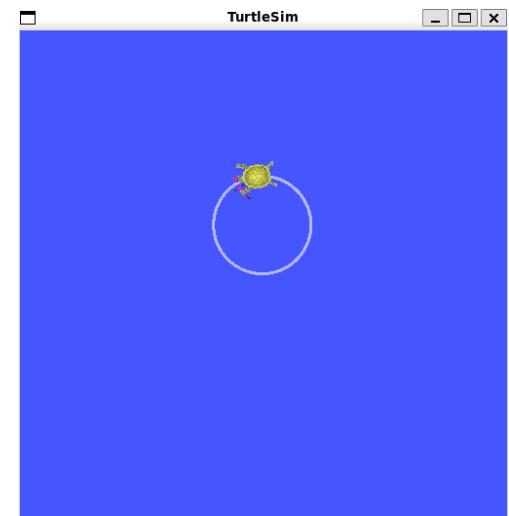
```
$ ros2 service find std_srvs/srv/Empty  
/clear  
/reset
```

```
$ ros2 service find turtlesim/srv/Kill  
/kill
```

❖ Service 요청 실습

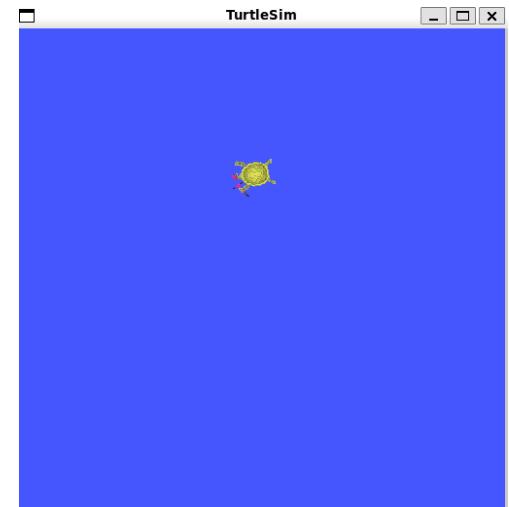
```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 topic pub --rate 1 /turtle1/cmd_vel
geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=1.8))

publishing #2: 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))
```



```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 service call /clear std_srvs/srv/Empty
waiting for service to become available...
requester: making request: std_srvs.srv.Empty_Request()

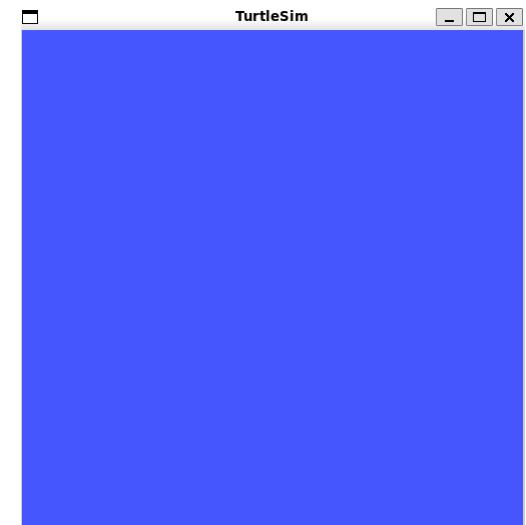
response:
std_srvs.srv.Empty_Response()
```



❖ Service 요청 실습

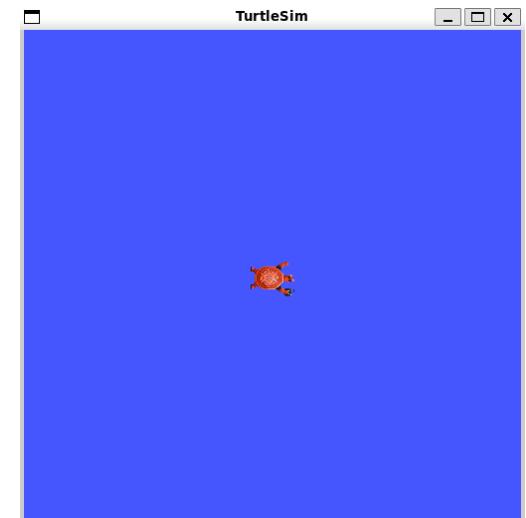
```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 service call /kill turtlesim/srv/Kill "name: 'turtle1'"
waiting for service to become available...
requester: making request: turtlesim.srv.Kill_Request(name='tur
tle1')

response:
turtlesim.srv.Kill_Response()
```



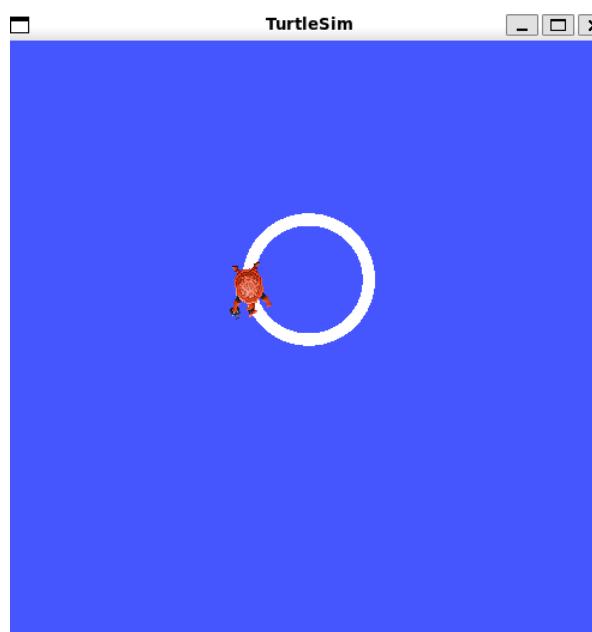
```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 service call /reset std_srvs/srv/Empty
requester: making request: std_srvs.srv.Empty_Request()

response:
std_srvs.srv.Empty_Response()
```



❖ Service 요청 실습

```
daesung@DSThinkPad:~$ ros2 service call /turtle1/set_pen turtle  
sim/srv/SetPen "{r: 255, g: 255, b: 255, width: 10}"  
waiting for service to become available...  
requester: making request: turtlesim.srv.SetPen_Request(r=255,  
g=255, b=255, width=10, off=0)  
  
response:  
turtlesim.srv.SetPen_Response()
```



❖ Service 요청 실습

```
$ ros2 service call /kill turtlesim/srv/Kill "name: 'turtle1'"
```

```
requester: making request: turtlesim.srv.Kill_Request(name='turtle1')
```

```
response:
```

```
turtlesim.srv.Kill_Response()
```

```
$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 5.5, y: 9, theta: 1.57, name: 'leonardo'}"
```

```
requester: making request: turtlesim.srv.Spawn_Request(x=5.5, y=9.0, theta=1.57, name='leonardo')
```

```
response:
```

```
turtlesim.srv.Spawn_Response(name='leonardo')
```

```
$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 5.5, y: 7, theta: 1.57, name: 'raffaello'}"
```

```
requester: making request: turtlesim.srv.Spawn_Request(x=5.5, y=7.0, theta=1.57, name='raffaello')
```

```
response:
```

```
turtlesim.srv.Spawn_Response(name='raffaello')
```

```
$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 5.5, y: 5, theta: 1.57, name: 'michelangelo'}"
```

```
requester: making request: turtlesim.srv.Spawn_Request(x=5.5, y=5.0, theta=1.57, name='michelangelo')
```

```
response:
```

```
turtlesim.srv.Spawn_Response(name='michelangelo')
```

```
$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 5.5, y: 3, theta: 1.57, name: 'donatello'}"
```

```
requester: making request: turtlesim.srv.Spawn_Request(x=5.5, y=3.0, theta=1.57, name='donatello')
```

```
response:
```

```
turtlesim.srv.Spawn_Response(name='donatello')
```

❖ Service 요청 실습

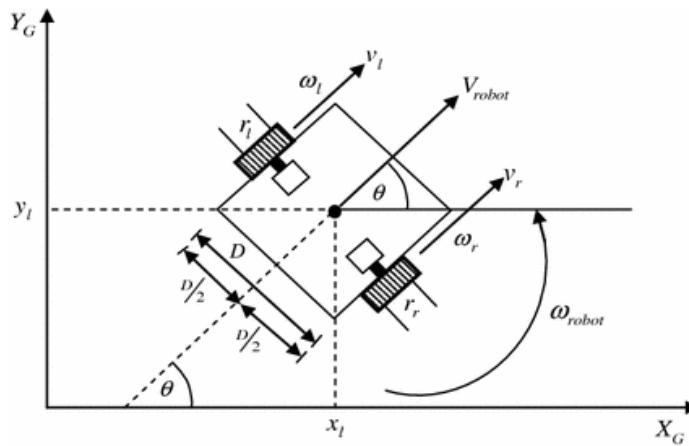


```
$ ros2 topic list
/donatello/cmd_vel
/donatello/color_sensor
/donatello/pose
/leonardo/cmd_vel
/leonardo/color_sensor
/leonardo/pose
/michelangelo/cmd_vel
/michelangelo/color_sensor
/michelangelo/pose
/new_turtle/cmd_vel
/new_turtle/pose
/parameter_events
/raffaello/cmd_vel
/raffaello/color_sensor
/raffaello/pose
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
/turtle2/cmd_vel
/turtle2/pose
/turtle3/cmd_vel
/turtle3/color_sensor
/turtle3/pose
/turtle4/cmd_vel
/turtle4/color_sensor
/turtle4/pose
```

❖ rqt를 이용한 Service type 조회

- ros2 interface show turtlesim/srv/TeleportAbsolute

```
(base) daesung@DSThinkPad:~$ ros2 interface show turtlesim/srv/TeleportAbsolute
float32 x
float32 y
float32 theta
---
(base) daesung@DSThinkPad:~$
```



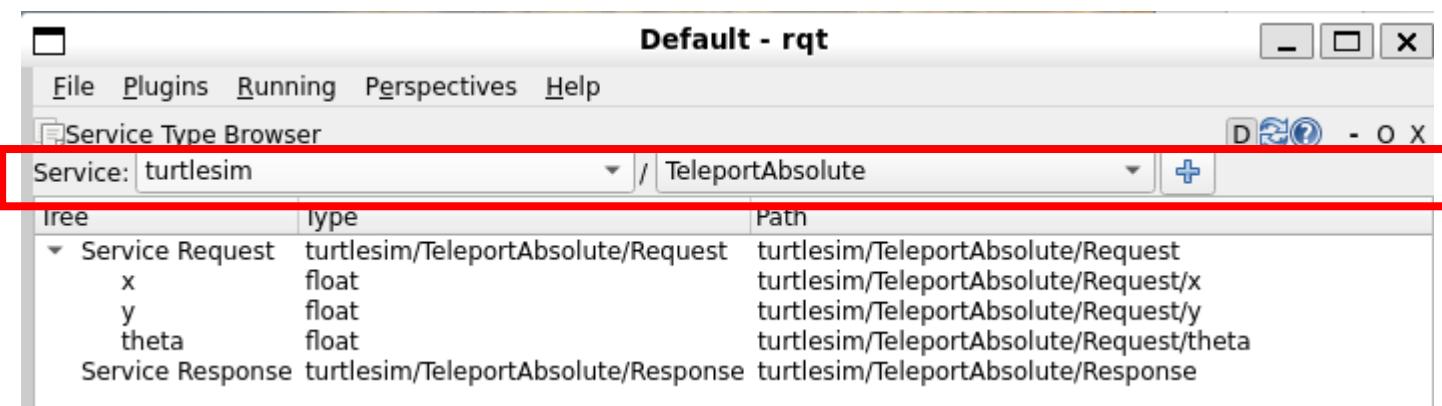
[theta]

Degrees → radians

$$\times \text{by } \frac{\pi}{180}$$

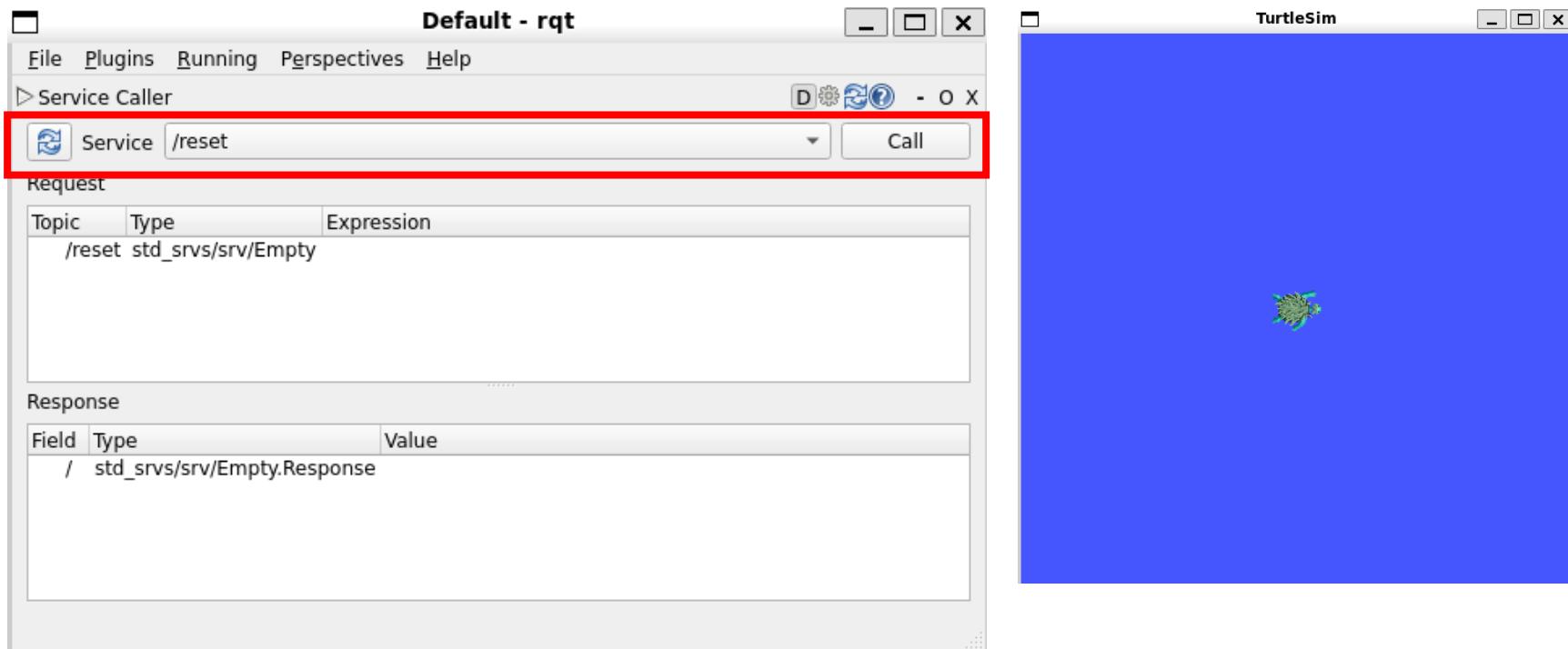
❖ rqt를 이용한 Service type 조회

- turtlesim/srv/TeleportAbsolute Service 내부 Data 구조 확인
- rqt를 이용한 Service 조회
- Plugins → Services → Service Type Browser



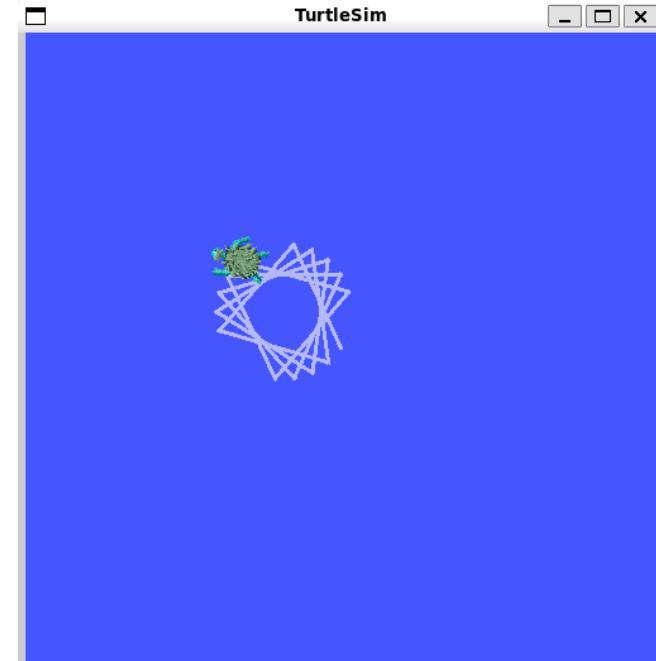
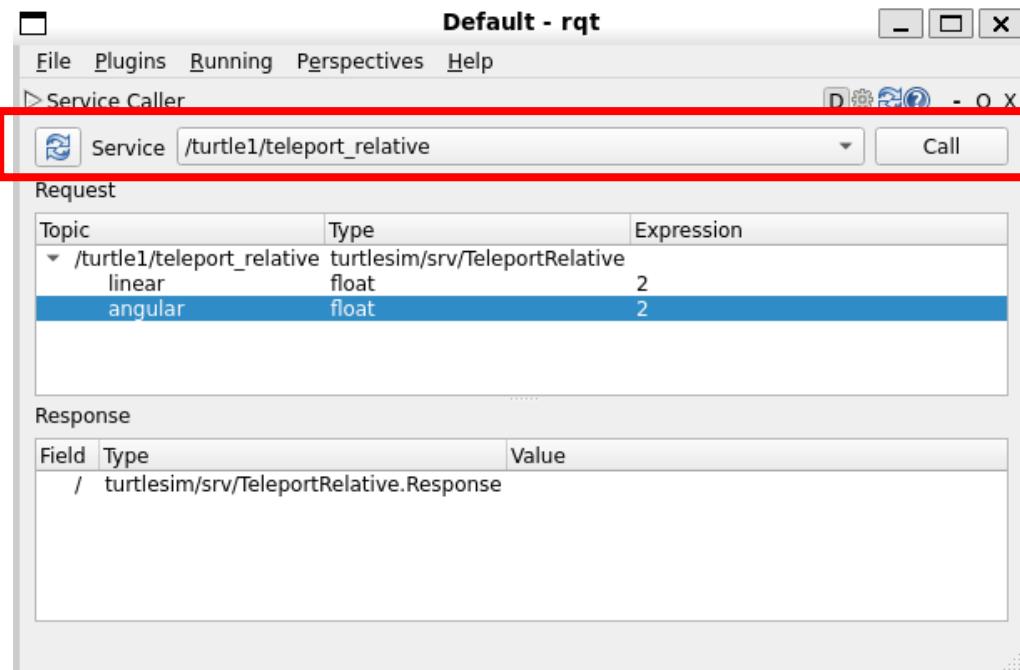
❖ rqt를 이용한 Service call

- rqt 실행
- Plugins → Services → Service Caller



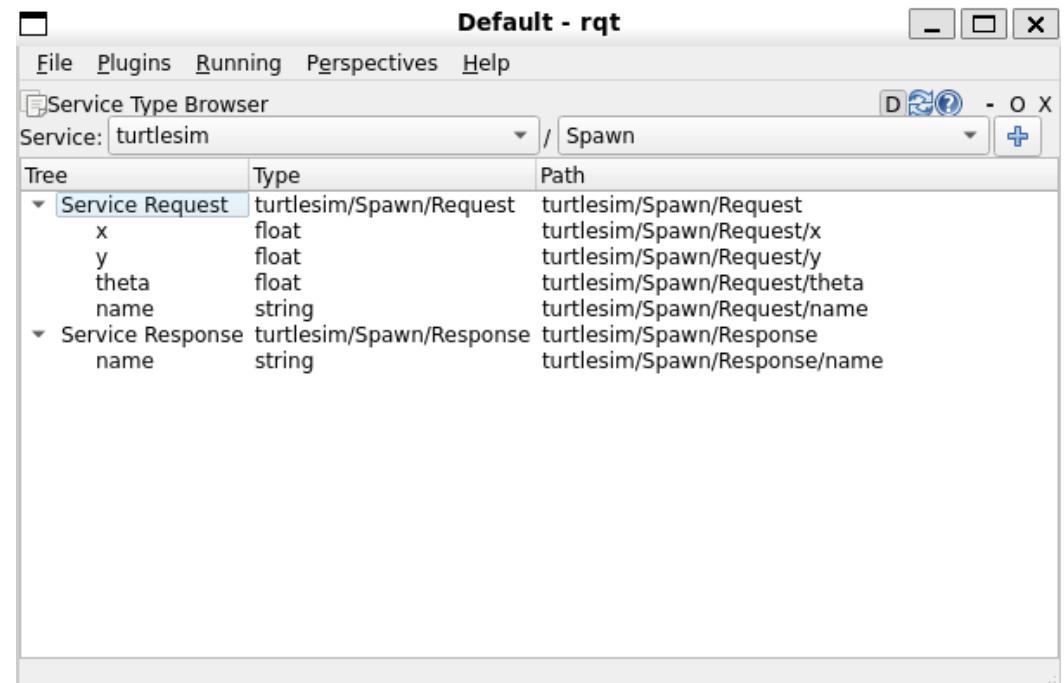
❖ rqt를 이용한 Service call

- rqt 실행
- Plugins → Services → Service Caller
- /turtle1/teleport_relative



❖ Service Interface

- 서비스 또한 토픽과 마찬가지로 별도의 인터페이스를 가지고 있음
 - 서비스 인터페이스라 부르며, 파일로는 srv 파일로 표현
 - 서비스 인터페이스는 메시지 인터페이스의 확장형이라고 볼 수 있음
-
- /spawn 서비스를 예시로 설명



❖ Service Interface

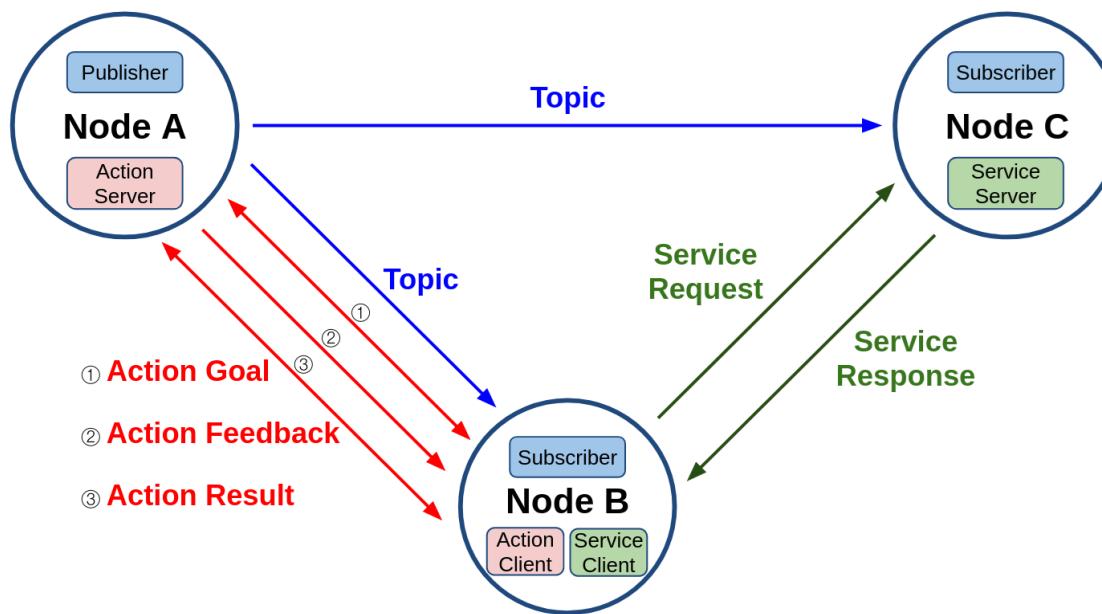
```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 interface show turtlesim/srv/Spawn
float32 x
float32 y
float32 theta
string name # Optional. A unique name will be created and returned if this is empty
---
string name
```

- float32 형태의 x, y, theta 그리고 string 형태로 name 두개의 데이터가 존재
- `---`는 구분자, 서비스 인터페이스는 메시지 인터페이스와는 달리 서비스 요청 및 응답(Request/Response) 형태로 구분
- x, y, theta, name 은 서비스 요청(서비스 클라이언트 → 서비스 서버)
- 서비스 서버는 지정된 서비스를 수행하고 name 데이터를 서비스 클라이언트에 전송

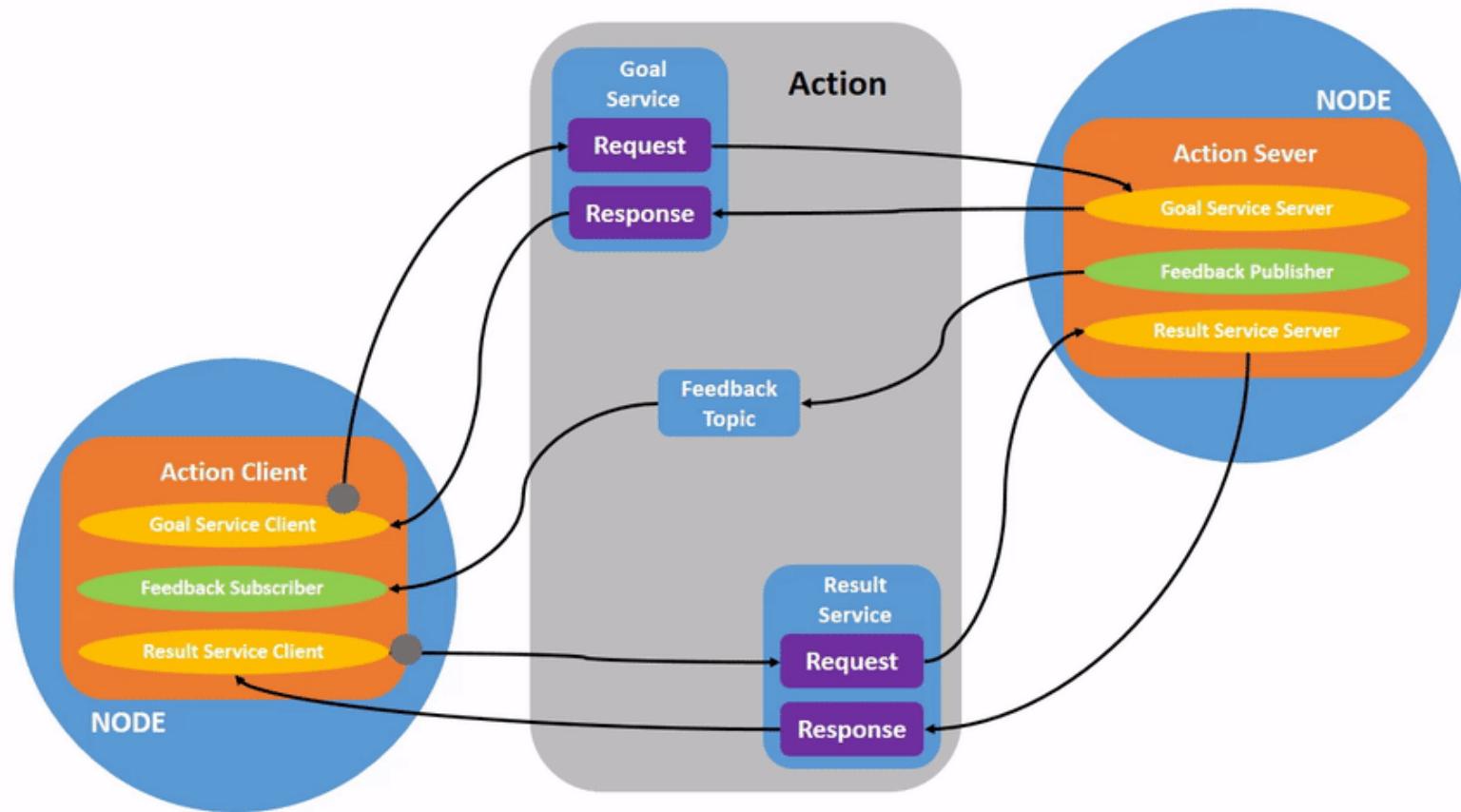
6 Action

❖ Action

- 액션은 토픽(topic)과 서비스(service)의 혼합
- 액션 목표 및 액션 결과를 전달하는 방식은 서비스와 같음
- 액션 피드백은 토픽과 같은 메시지 전송 방식



❖ Action



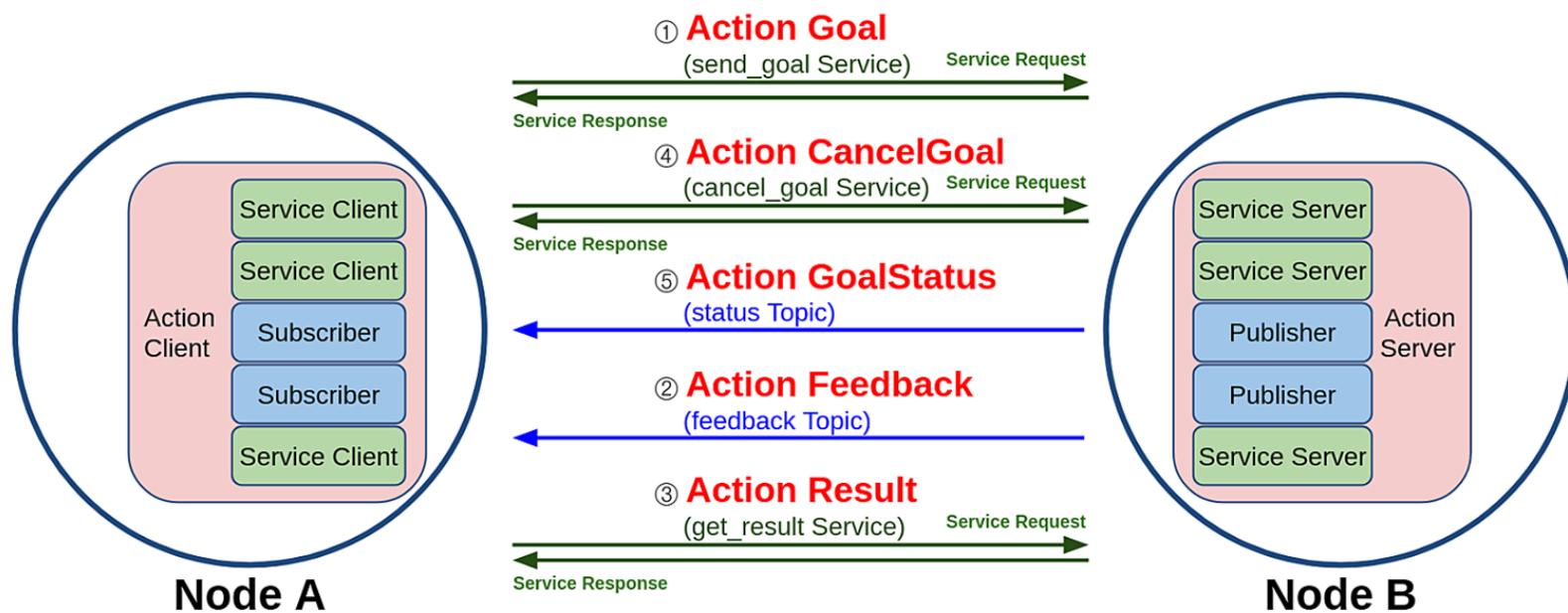
❖ Action

- 액션(Action)은 비동기식(Topic)+동기식(Service) 양방향 메시지 송수신 방식
 - ✓ 액션 목표 Goal를 지정하는 Action client
 - ✓ 액션 목표를 받아 태스크를 수행하면서 중간 결과 값에 해당되는 액션 피드백(Feedback)
 - ✓ 최종 결과 값에 해당되는 액션 결과(Result)를 전송
- ROS1은 토픽만을 사용
- ROS2에서는 액션 목표, 액션 결과, 액션 피드백에 대한 데이터를 교환. 토픽과 서비스 혼합



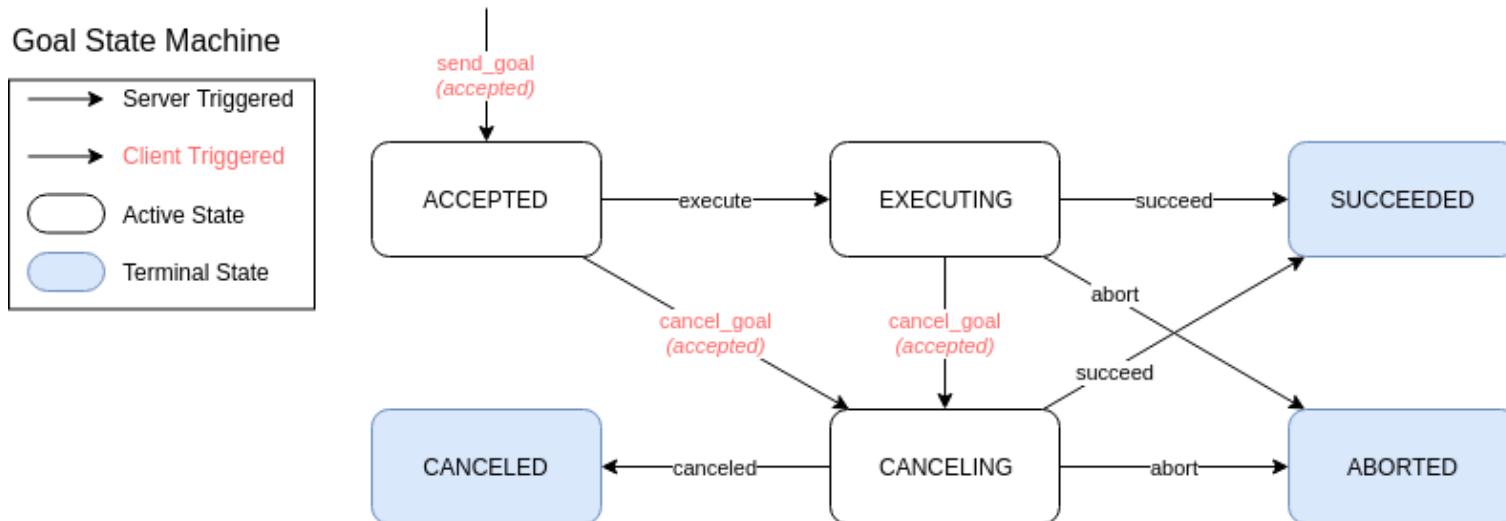
❖ Action

- Action Client = Service Client 3개 + Topic Subscriber 2개
- Action Server = Service Server 3개 + Topic Publisher 2개
- 액션 목표/피드백/결과(goal/feedback/result) 데이터는 msg 및 srv 인터페이스의 변형으로 action 인터페이스라 함



❖ Action

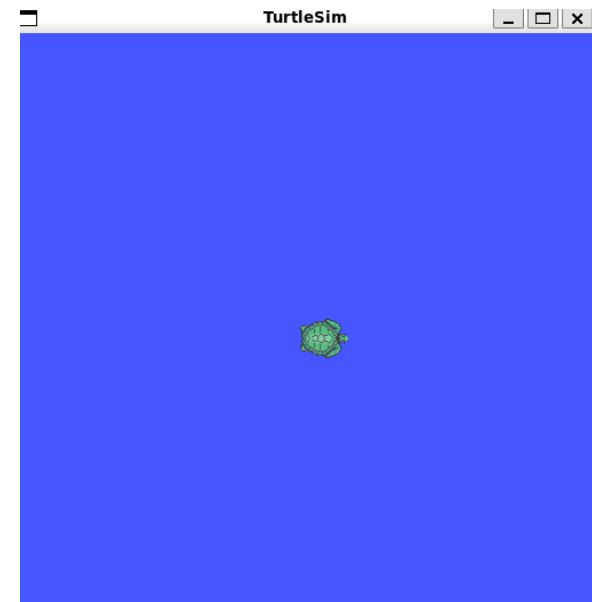
- ROS1은 비동기 방식 이용. 이로 인해 원하는 타이밍에 적절한 액션을 수행하기 어려움
- ROS 2에서 목표 상태(goal state)라는 개념 등장.
- 목표 상태는 목표 값을 전달 한 후의 State-Machine을 구동하여 액션의 프로세스를 모니터링.



❖ Action Server & Client Test

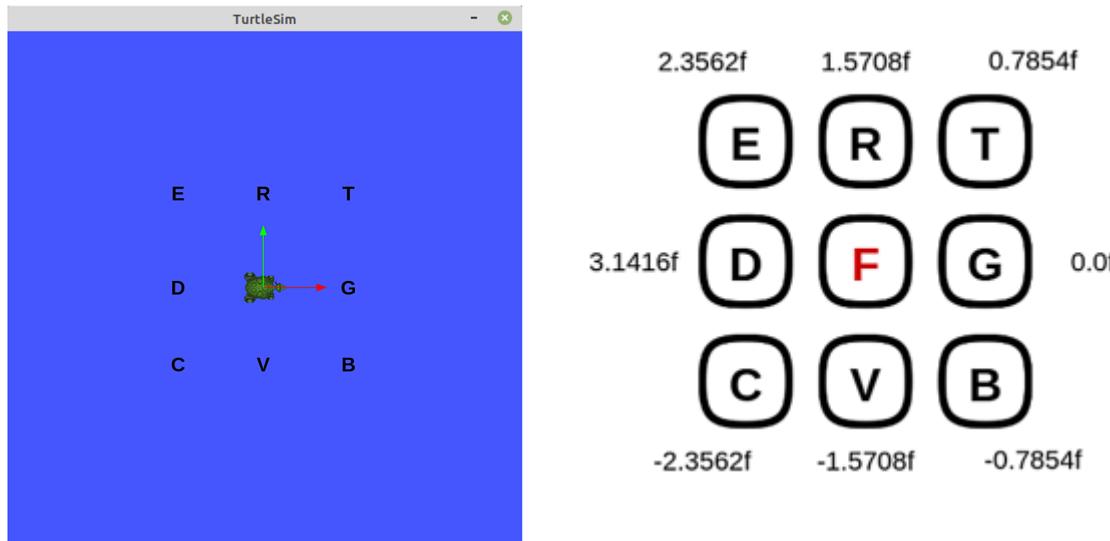
- turtlesim 활용
- turtlesim_node 및 turtle_teleop_key 사용
- 방향키가 아닌 F키 주변에 위치하고 있는 G, B, V, C, D, E, R, T 키 사용

```
[daesung@DSThinkPad:~]$ 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 rotation.
'Q' to quit.
```



❖ Action Server & Client Test

- rotate_absolute Action 수행에 있어 액션의 목표 값을 전달
- 다음 그림과 같이 각 버튼은 거북이를 절대 각도로 회전하도록 목표 값 설정
- F키를 누르면 전달한 목표 값을 취소하여 동작을 바로 멈춤
- G키는 theta 값이 0.0이며, rotate_absolute 액션의 기준 각도
- 다른 키는 위치별로 0.7854 radian씩 반시계 방향으로 회전



❖ Action Server & Client Test

- 액션 목표는 도중에 취소할 수도 있으며 turtlesim_node에도 상황을 터미널 창에 표시
- 액션 목표의 취소 없이 목표 theta 값에 도달하면 다음과 같이 표시

[INFO]: Rotation goal completed successfully

- 액션 목표 값에 도달하기 전에 F키를 눌러 취소하게 되면 turtlesim_node의 터미널 창에 아래와 같은 상태를 표시하고 거북이는 동작을 멈춤

[INFO]: Rotation goal canceled

Node & Action

❖ Node information

- /turtlesim

```
daesung@DSThinkPad: ~ 68x41
daesung@DSThinkPad:~$ 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/SetParametersAtomically
  Service Clients:
    Action Servers:
      /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
    Action Clients:
```

Node & Action

❖ Node information

- /teleop_turtle

```
daesung@DSThinkPad: ~ 68x41
daesung@DSThinkPad:~$ ros2 node info /teleop_turtle
/teleop_turtle
Subscribers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
Publishers:
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /rosout: rcl_interfaces/msg/Log
  /turtle1/cmd_vel: geometry_msgs/msg/Twist
Service Servers:
  /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
Service Clients:
  Action Servers:
  Action Clients:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
```

❖ Action 목록 및 정보

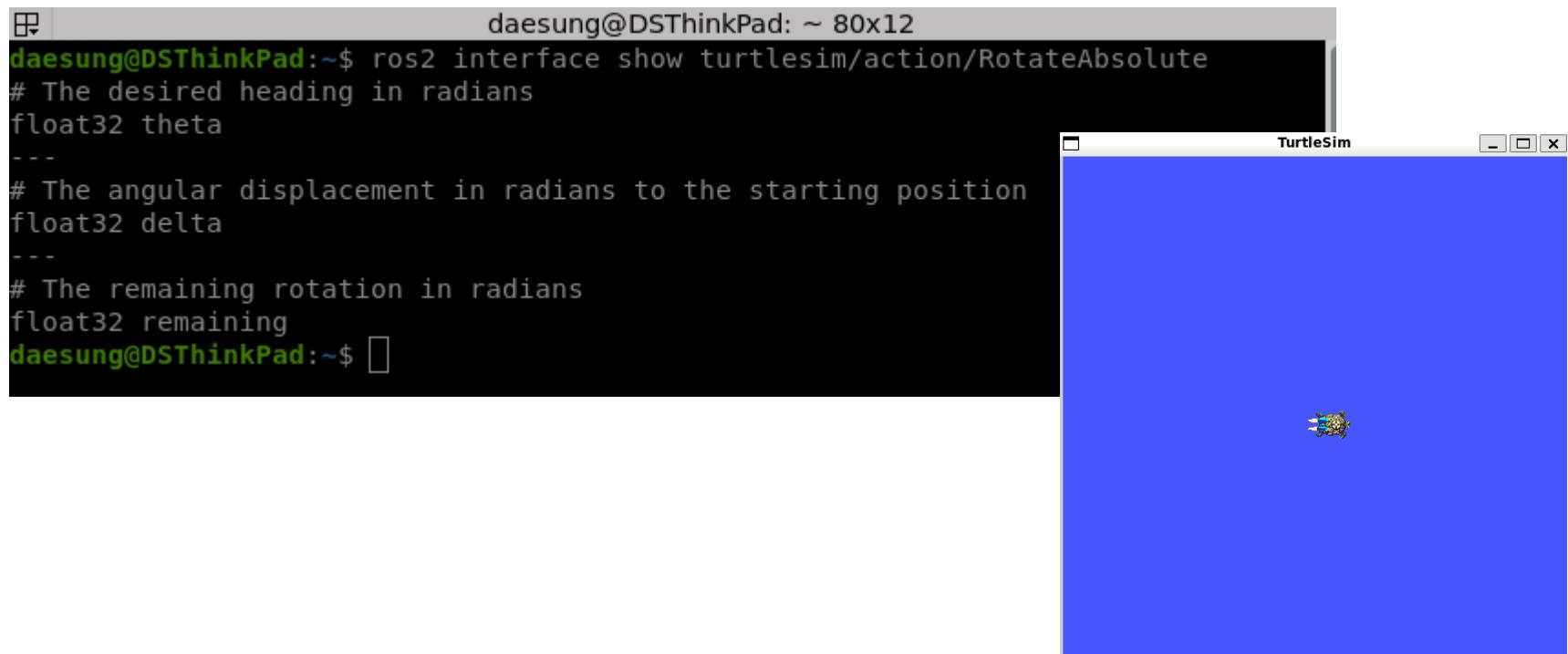
- ROS2 action list & interface type

The image shows three terminal windows from a Linux desktop environment. A window titled 'TurtleSim' displays a small yellow turtle icon on a blue background. The other two windows show command-line output:

- Top Terminal:** Shows the command `ros2 action list -t` and its output: `/turtle1/rotate_absolute [turtlesim/action/RotateAbsolute]`.
- Middle Terminal:** Shows the command `ros2 run turtlesim turtle_teleop_key` and its instructions for controlling the turtle with arrow keys and specific rotation keys.
- Bottom Terminal:** Shows the command `ros2 action info /turtle1/rotate_absolute` and its details: Action: /turtle1/rotate_absolute, Action clients: 1 (teleop_turtle), Action servers: 1 (/turtlesim).

❖ Action 목록 및 정보

- ROS2 action interface type
- [theta] 목표 각도 값, [delta] Action 시작 위치에서의 각도 변위, [remaining] 목표 각도로 이동까지 남은 각도의 값으로 feedback 옵션을 사용해야 나타남



The image shows a terminal window on the left and a TurtleSim window on the right.

Terminal Output:

```
daesung@DSThinkPad: ~ 80x12
daesung@DSThinkPad:~$ ros2 interface show turtlesim/action/RotateAbsolute
# The desired heading in radians
float32 theta
---
# The angular displacement in radians to the starting position
float32 delta
---
# The remaining rotation in radians
float32 remaining
daesung@DSThinkPad:~$
```

TurtleSim Window:

A window titled "TurtleSim" is displayed. It shows a small yellow turtle icon centered on a solid blue background. The window has standard Linux-style window controls (minimize, maximize, close) at the top right corner.

❖ Action 목표 전달

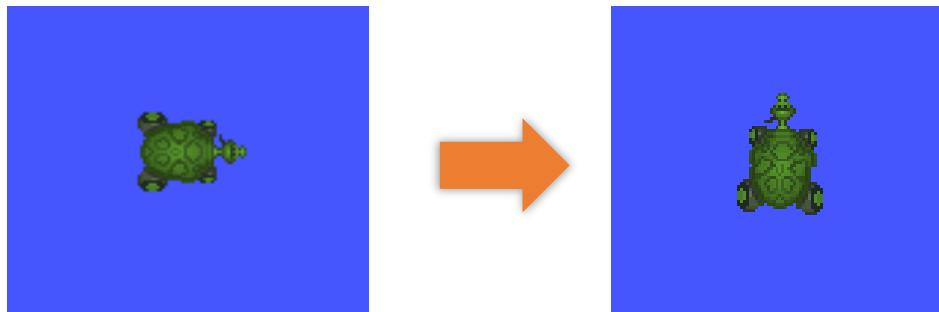
```
$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.5708}"
```

```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.5708}"
Waiting for an action server to become available...
Sending goal:
    theta: 1.5708

Goal accepted with ID: 62810cdefd664ccfae3a2e38fe15f9b6

Result:
    delta: -1.5520000457763672

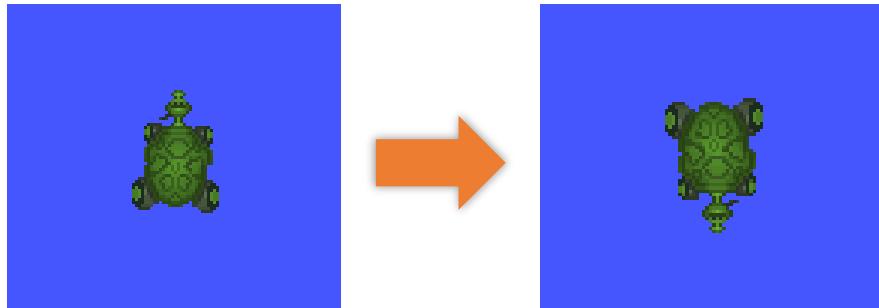
Goal finished with status: SUCCEEDED
```



❖ Action 목표 전달(feedback 옵션 사용)

```
$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute  
"{'theta: -1.5708}" --feedback
```

```
Feedback:  
    remaining: -0.06679999828338623  
  
Feedback:  
    remaining: -0.05079996585845947  
  
Feedback:  
    remaining: -0.034799933433532715  
  
Feedback:  
    remaining: -0.018799901008605957  
  
Result:  
    delta: 3.1040000915527344  
  
Goal finished with status: SUCCEEDED
```



❖ Action Interface

- 액션 또한 별도의 인터페이스를 가지고 있으며 액션 인터페이스라 칭함
- 파일은 Action파일로 표기되며, 메시지 및 서비스 인터페이스의 확장형임
- 실습에서 사용한 /turtle1/rotate_absolute 로 설명

Tree	Type	Path
▼ Action Goal	turtlesim/RotateAbsolute/Goal	turtlesim/RotateAbsolute/Goal
theta	float	turtlesim/RotateAbsolute/Goal/theta
▼ Action Result	turtlesim/RotateAbsolute/Result	turtlesim/RotateAbsolute/Result
delta	float	turtlesim/RotateAbsolute/Result/delta
▼ Action Feedback	turtlesim/RotateAbsolute/Feedback	turtlesim/RotateAbsolute/Feedback
remaining	float	turtlesim/RotateAbsolute/Feedback/remaining

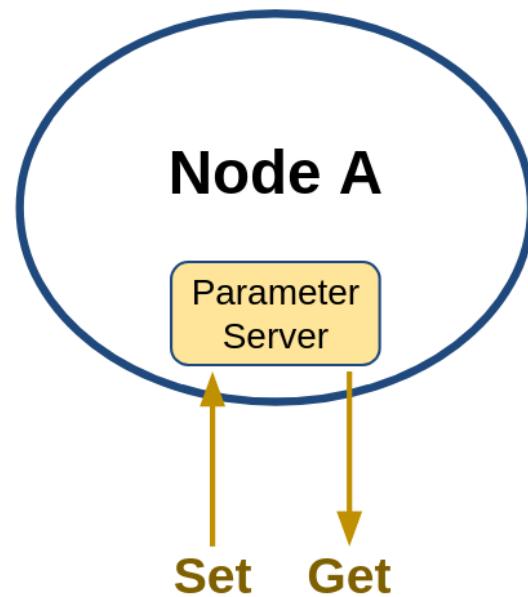
❖ Action Interface

```
daesung@DSThinkPad: ~ 80x12
daesung@DSThinkPad:~$ ros2 interface show turtlesim/action/RotateAbsolute
# The desired heading in radians
float32 theta
---
# The angular displacement in radians to the starting position
float32 delta
---
# The remaining rotation in radians
float32 remaining
daesung@DSThinkPad:~$ 
```

- turtlesim/action/RotateAbsolute.action
- float32 형태의 theta, delta, remaining 라는 세개의 데이터가 있음
- `---` 구분자로 목표(goal), 결과(result), 피드백(feedback)으로 구분

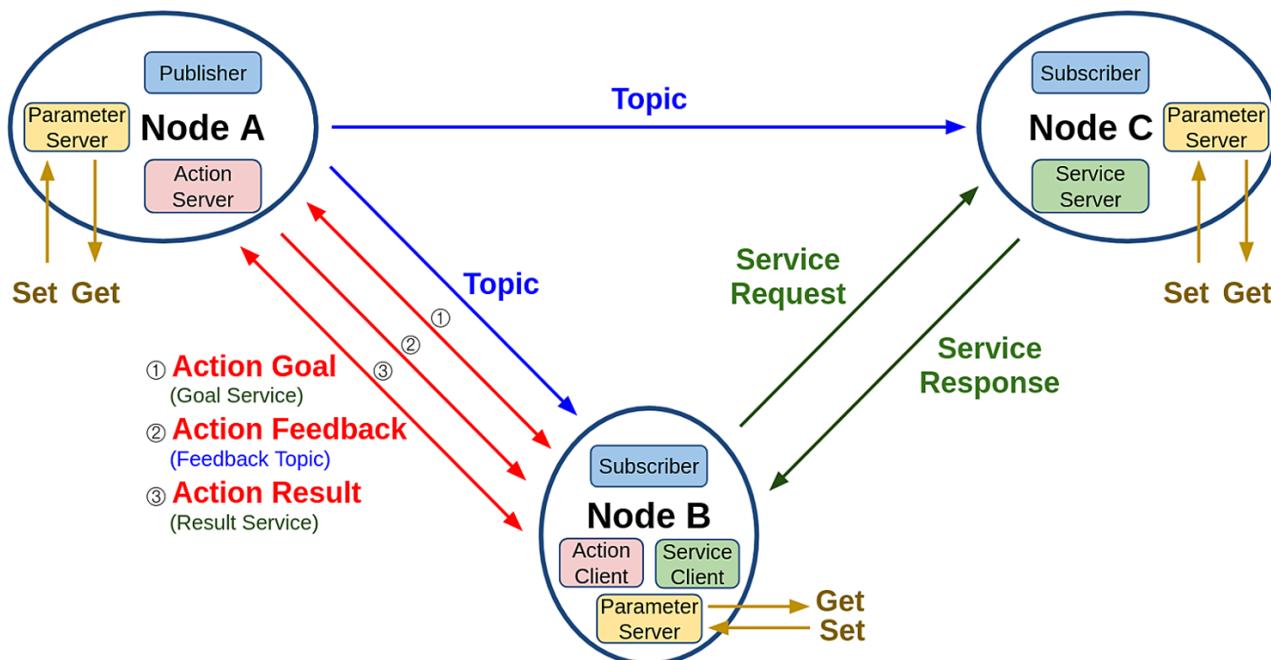
7 Parameter

- ❖ Parameter



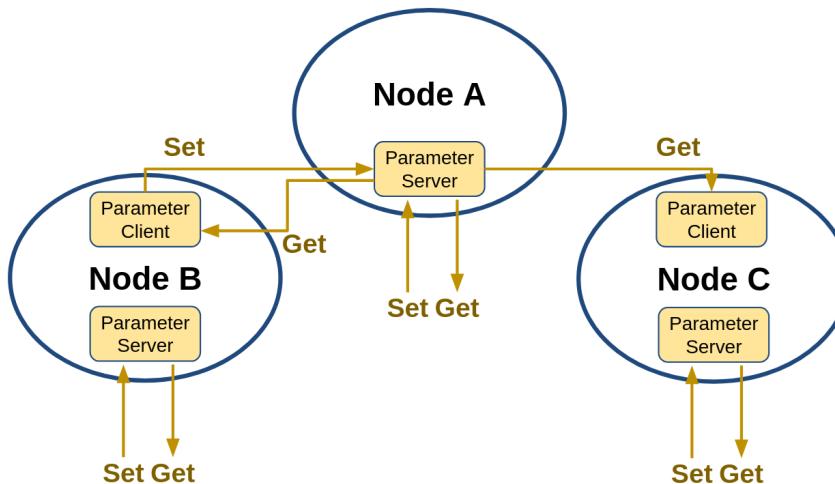
❖ Parameter

- 각 노드의 Parameter server를 실행시켜 외부의 Parameter client 간의 통신으로
파라미터를 변경하는 것으로 서비스와 동일
- 단 노드 내 매개변수 또는 글로벌 매개변수를 서비스 통신 방법을 사용하여
외부에서 Set/Get 할 수 있게 함



❖ Parameter

- 파라미터 관련 기능은 RCL(ROS Client Libraries)의 기본 기능으로 모든 노드가 자신만의 Parameter server를 가지고 있음
- 각 노드는 Parameter client도 포함시킬 수 있어서 자기 자신의 파라미터 및 다른 노드의 파라미터를 읽고 쓸 수 있음
- 추가 프로그래밍이나 컴파일 없이 능동적으로 변화 가능한 프로세스 생성 가능
- 각 파라미터는 yaml 파일 형태의 파라미터 설정 파일로 되어 있음



❖ Parameter 목록 확인

- turtlesim 활용
- turtlesim_node 및 turtle_teleop_key 사용

```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 param list
/teleop_turtle:
    qos_overrides./parameter_events.publisher.depth
    qos_overrides./parameter_events.publisher.duration
    qos_overrides./parameter_events.publisher.history
    qos_overrides./parameter_events.publisher.reliability
    scale_angular
    scale_linear
    use_sim_time
/turtlesim:
    background_b
    background_g
    background_r
    qos_overrides./parameter_events.publisher.depth
    qos_overrides./parameter_events.publisher.duration
    qos_overrides./parameter_events.publisher.history
    qos_overrides./parameter_events.publisher.reliability
    use_sim_time
```

❖ Parameter 내용 확인

```
$ ros2 param describe /turtlesim background_b
```

```
daesung@DSThinkPad: ~ 63x29
daesung@DSThinkPad:~$ ros2 param describe /turtlesim background_b
Parameter name: background_b
Type: integer
Description: Blue channel of the background color
Constraints:
  Min value: 0
  Max value: 255
  Step: 1
daesung@DSThinkPad:~$
```

❖ Parameter 값 읽기

```
$ ros2 param get /turtlesim background_r
```

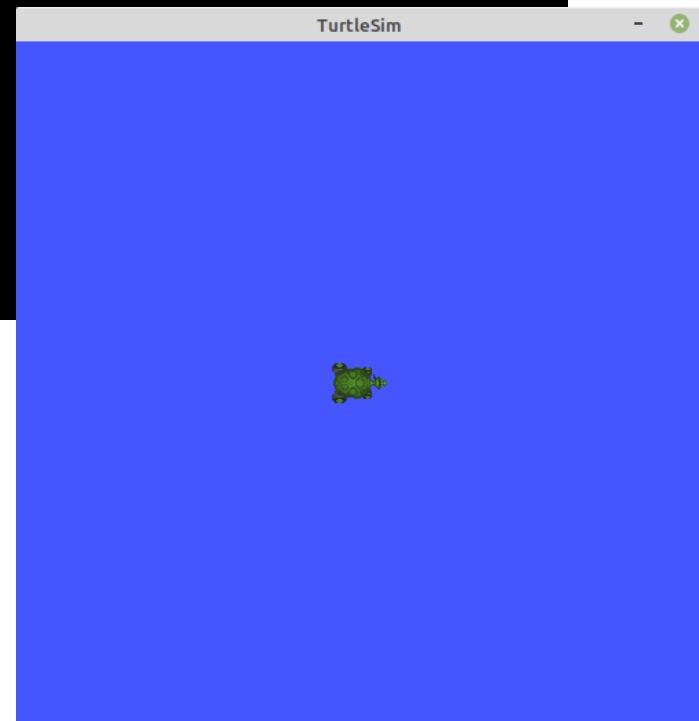
```
Integer value is: 69
```

```
$ ros2 param get /turtlesim background_g
```

```
Integer value is: 86
```

```
$ ros2 param get /turtlesim background_b
```

```
Integer value is: 255
```



❖ Parameter 값 쓰기

```
$ ros2 param set /turtlesim background_r 148
```

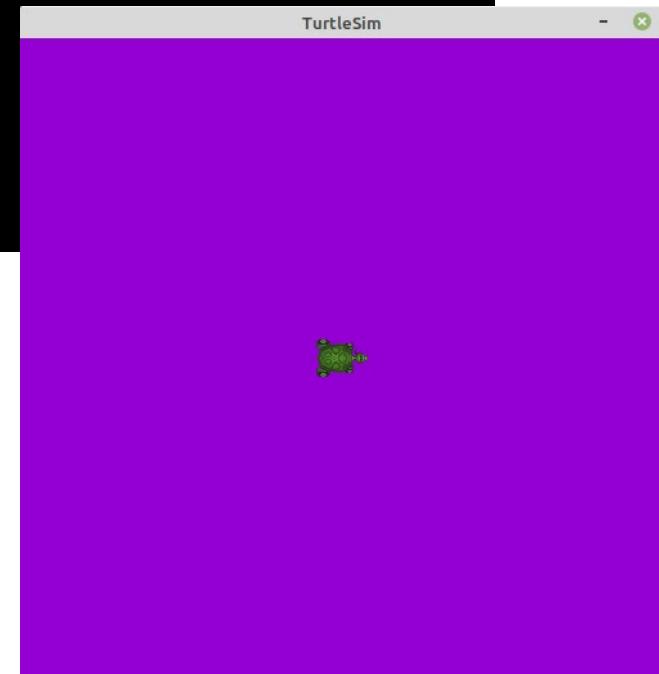
Set parameter successful

```
$ ros2 param set /turtlesim background_g 0
```

Set parameter successful

```
$ ros2 param set /turtlesim background_b 211
```

Set parameter successful



- ❖ Parameter 저장 값 확인 및 재사용

```
$ ros2 param set /turtlesim background_r 148
```

```
Set parameter successful
```

```
$ ros2 param set /turtlesim background_g 0
```

```
Set parameter successful
```

```
$ ros2 param set /turtlesim background_b 211
```

```
Set parameter successful
```

❖ Parameter 저장 값 확인 및 재사용

- ros2 param set으로 파라미터의 값을 바꾸어 사용할 수 있으나 turtlesim 노드를 종료하고 다시 시작하면 모든 파라미터는 초기 값으로 다시 설정
- 현재 파라미터를 저장하고 다시 불러오기 위해 ros2 param dump 명령어 사용
- 현재 폴더에 해당 노드 이름으로 설정 파일이 yaml 형태로 저장

```
$ ros2 param dump /turtlesim  
Saving to: ./turtlesim.yaml
```

```
$ ros2 run turtlesim turtlesim_node --ros-args --params-file ./turtlesim.yaml
```

8 Topic/Service/Action 비교

Topic/Service/Action 비교

	토픽 (topic)	서비스 (service)	액션 (action)
연속성	연속성	일회성	복합 (토픽+서비스)
방향성	단방향	양방향	양방향
동기성	비동기	동기	동기 + 비동기
다자간 연결	1:1, 1:N, N:1, N:N (publisher:subscriber)	1:1 (server:client)	1:1 (server:client)
노드 역할	발행자 (publisher) 구독자 (subscriber)	서버 (server) 클라언트 (client)	서버 (server) 클라언트 (client)
동작 트리거	발행자	클라언트	클라언트
인터페이스	msg 인터페이스	srv 인터페이스	action 인터페이스
CLI 명령어	ros2 topic ros2 interface	ros2 service ros2 interface	ros2 action ros2 interface
사용 예	센서 데이터, 로봇 상태, 로봇 좌표, 로봇 속도 명령 등	LED 제어, 모터 토크 On/Off, IK/FK 계산, 이동 경로 계산 등	목적지로 이동, 물건 파지, 복합 태스크 등

Topic/Service/Action 비교

	msg 인터페이스	srv 인터페이스	action 인터페이스
확장자	*.msg	*.srv	*.action
데이터	토픽 데이터 (data)	서비스 요청 (request) --- 서비스 응답 (response)	액션 목표 (goal) --- 액션 결과 (result) --- 액션 피드백 (feedback)
형식	fieldtype1 fieldname1 fieldtype2 fieldname2 fieldtype3 fieldname3	fieldtype1 fieldname1 fieldtype2 fieldname2 --- fieldtype3 fieldname3 fieldtype4 fieldname4	fieldtype1 fieldname1 fieldtype2 fieldname2 --- fieldtype3 fieldname3 fieldtype4 fieldname4 --- fieldtype5 fieldname5 fieldtype6 fieldname6
사용 예	[geometry_msgs/msg/Twist] Vector3 linear Vector3 angular	[turtlesim/srv/Spawn.srv] float32 x float32 y float32 theta string name --- string name	[turtlesim/action/RotateAbsolute.action] float32 theta --- float32 delta --- float32 remaining



Q & A
