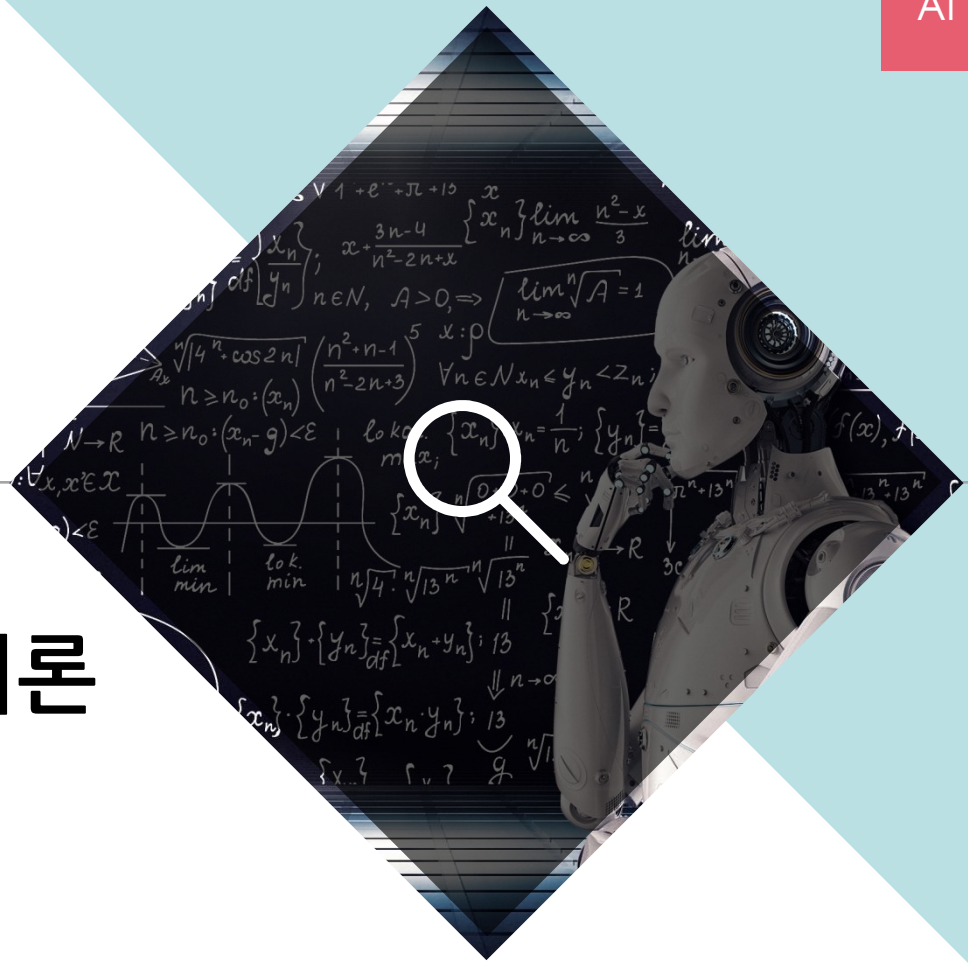
AI TRAINING SERIES



ROS2 와 Docker 이해하기 - 이론

Contents

I. ROS란 무엇인가?

II. ROS2의 주요 구성 요소

III. ROS2의 주요 특징

IV. Docker란 무엇인가?

V. Docker의 기본 구성 요소

VI. ROS2와 Docker

VII. Q&A

ROS

docker

I. ROS 란 무엇인가?

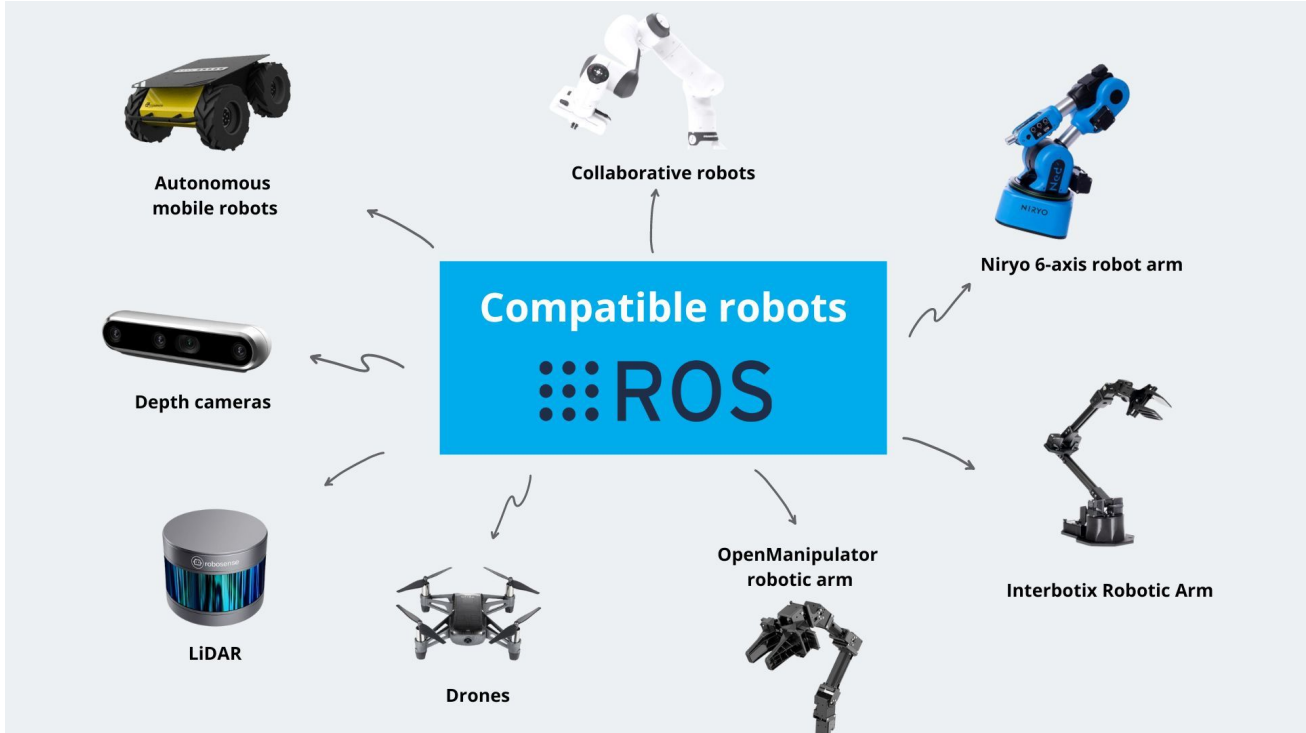
I. ROS 란 무엇인가?

로봇 소프트웨어 개발을 위한 오픈소스 미들웨어

- 하드웨어와 소프트웨어 간의 **통신을 쉽게** 할 수 있도록 지원
- 로봇 개발에 필요한 **기능들(센서, 모터 제어, 비전 처리 등)** 제공

주요 역할

- **하드웨어와 소프트웨어 상호작용 관리**
- 로봇 애플리케이션의 **효율적인 개발** 지원
- 다양한 **드라이버와 라이브러리** 제공



I. ROS 란 무엇인가?

ROS 역사와 배경 요약

- **2007년**: **Stanford AI Lab**와 **Willow Garage**에서 시작
    - 로봇 하드웨어와 소프트웨어 통합을 위한 미들웨어 개발
- **2010년**: **ROS 1** 공개
    - 오픈소스 프레임워크로 빠르게 확산, 연구용 로봇 및 프로토타입에 사용
- **2017년**: **ROS 2** 출시
    - 실시간 시스템, 보안, 분산 처리 기능 추가, 산업용 및 상용 로봇 지원
- **현재**: 로봇 개발의 **표준**으로 자리 잡은 오픈소스 플랫폼

I. ROS 란 무엇인가?

ROS1과 ROS2의 차이점

항목	ROS1	ROS2
통신 프로토콜	TCPROS, XML-RPC	DDS (실시간 통신 지원)
실시간 처리	실시간 지원 부족	실시간 시스템 지원
플랫폼 지원	주로 리눅스	Linux, Windows, macOS 지원
보안	보안 기능 부족	보안 기능 강화 (암호화, 인증 등)
확장성	제한적 확장성	모듈화된 아키텍처로 높은 확장성

I. ROS 란 무엇인가?

ROS2가 중요한 이유

- **실시간 시스템 지원**: 자율주행차, 산업용 로봇 등 실시간 처리가 필요한 시스템에 적합.
- **분산 시스템 안정성**: DDS 프로토콜로 네트워크 장애에도 안정적인 통신 제공.
- **다양한 플랫폼 지원**: Linux, Windows, macOS 등 다양한 운영체제에서 호환.
- **보안 강화**: 암호화, 인증 등 보안 기능 제공, 민감한 데이터 보호.
- **모듈화된 아키텍처**: 높은 확장성 및 시스템 수정 용이.
- **산업 및 상용 로봇에 적합**: 고도화된 시스템 및 상용 애플리케이션에 최적화.

Ⅱ. ROS2의 주요 구성 요소

Ⅱ. ROS2의 주요 구성 요소

ROS2의 핵심 기능 구조

1. **노드(Node)**
   a. 실행 단위인 프로세스. 로봇 애플리케이션을 작은 기능 단위로 분리해 관리합니다.

2. **메시지(Message)**
   a. 노드 간에 주고받는 데이터 형식(struct). 토픽이나 서비스, 액션 등에 담겨 전송됩니다.

3. **토픽(Topic)**
   a. 메시지를 발행(publish)하거나 구독(subscribe)하는 “이름표” 같은 역할로, 퍼블리셔와 서브스크라이버를 연결해 줍니다.

Ⅱ. ROS2의 주요 구성 요소

ROS2의 핵심 통신 패턴

1. **퍼블리셔/서브스크라이버 (Pub/Sub)**
   a. **용도**: 연속적인 데이터 스트리밍(예: 센서 값, 로봇 상태)
   b. **특징**: 느슨한 결합(loose coupling), 비동기 방식

2. **서비스 (Service)**
   a. **용도**: 단발성 요청–응답(예: “현재 위치 알려 줘”, “이 설정으로 변경해 줘”)
   b. **특징**: 동기 호출, 요청이 오면 응답 후 종료

3. **액션 (Action)**
   a. **용도**: 완료까지 시간이 걸리는 작업(예: 궤적 계획·실행, 맵 생성)
   b. **특징**: goal 요청 → 진행 피드백(feedback) → 완료 결과(result), 취소 지원

II. ROS2의 주요 구성 요소

추가적으로 알아둬야 하는 구성요소

1. **파라미터(Parameter)**: 런타임 설정값 저장소
   a. 런타임 설정값(컨트롤러 게인, 시뮬레이션 속도 등) 저장·관리

2. **TF**: 좌표계(transform) 간 변환 관리 모듈
   a. 서로 다른 좌표계(frame) 간 변환(transform) 관리

[ROS2 의 공식 튜토리얼 문서]

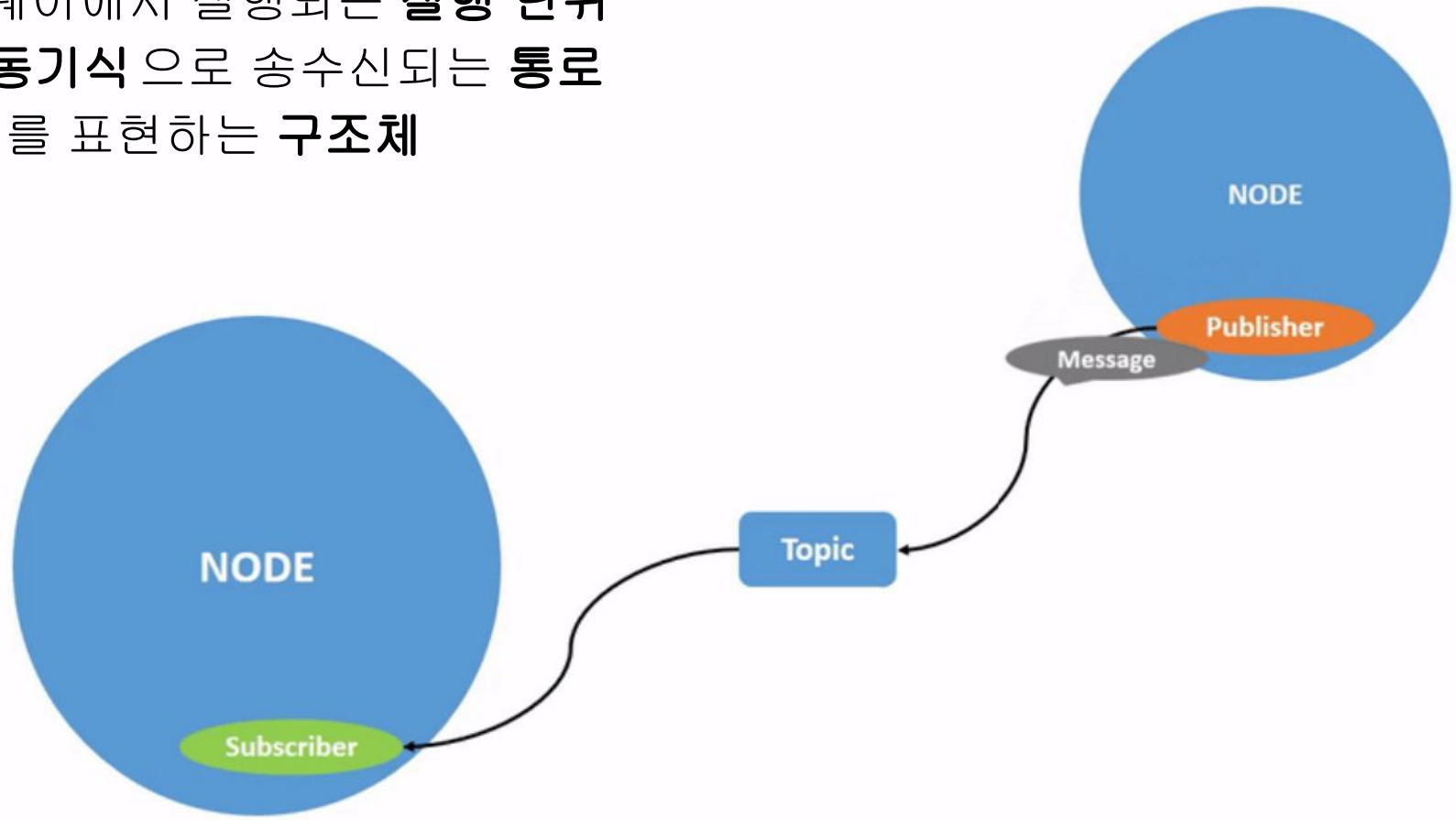
https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools.html

움직이는 그림으로 많은 예제들이 있으니 공부할 때 참고할 것.

II. ROS2의 주요 구성 요소

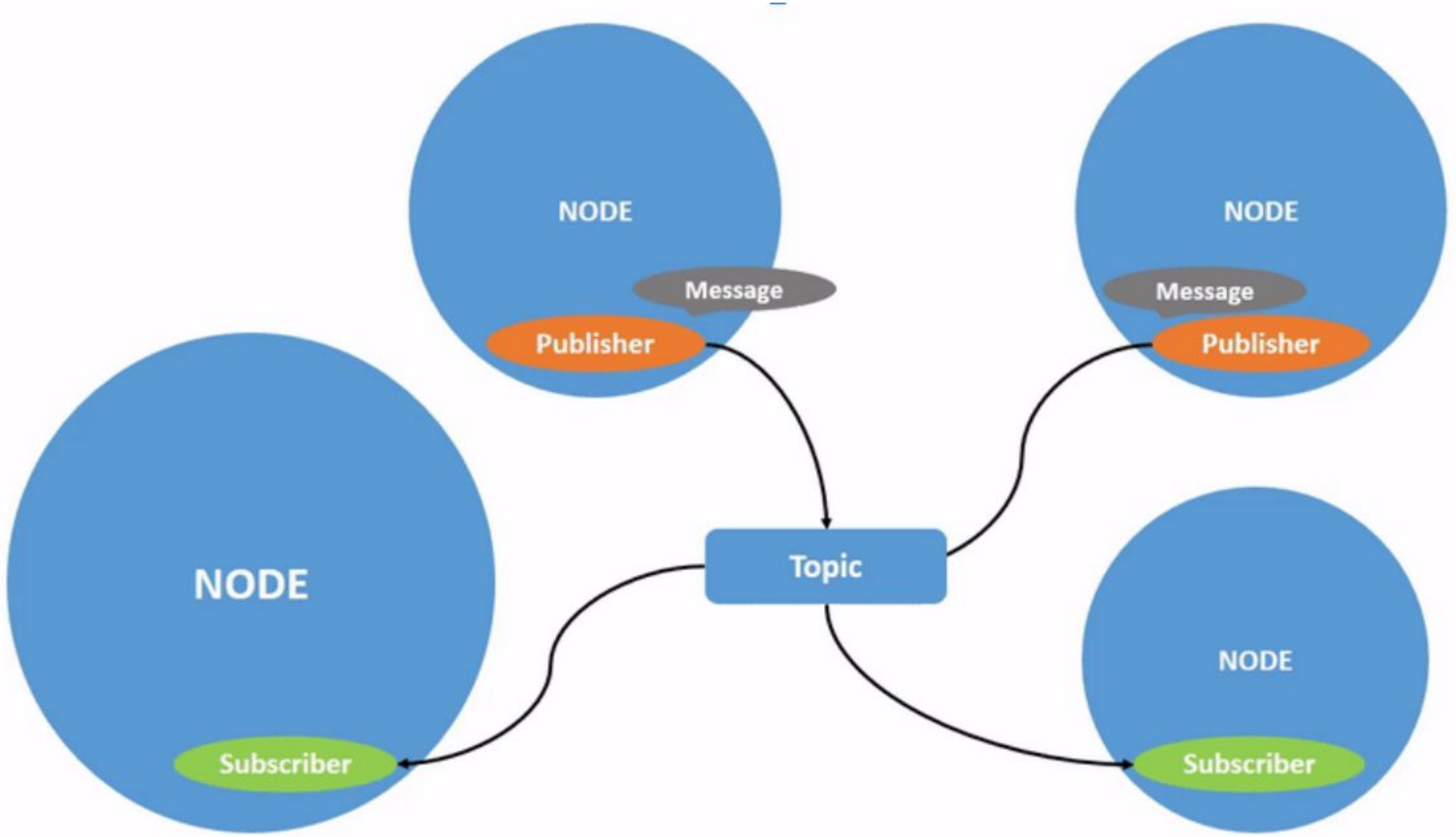
퍼블리셔/서브스크라이버 (Pub/Sub) 구조

- **노드(Node)**: 로봇 소프트웨어에서 실행되는 **실행 단위**
- **토픽(Topic)**: 데이터가 **비동기식**으로 송수신되는 **통로**
- **메시지(Message)**: 데이터를 표현하는 **구조체**



Ⅱ. ROS2의 주요 구성 요소

여러개의 노드가 동일한 **Topic**을 수신/ 전송할 수도 있다.



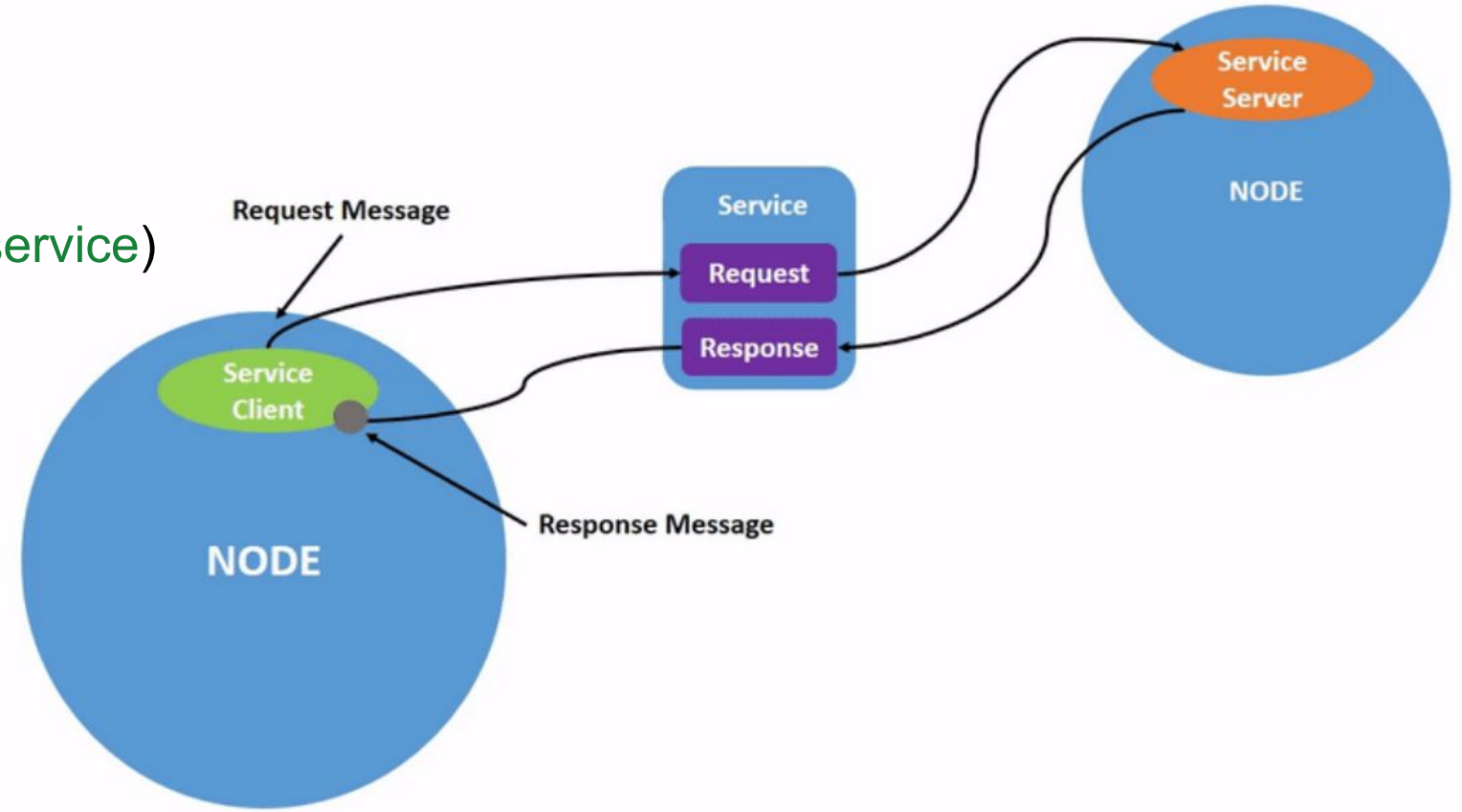
Ⅱ. ROS2의 주요 구성 요소

Service 노드 (Service Node)

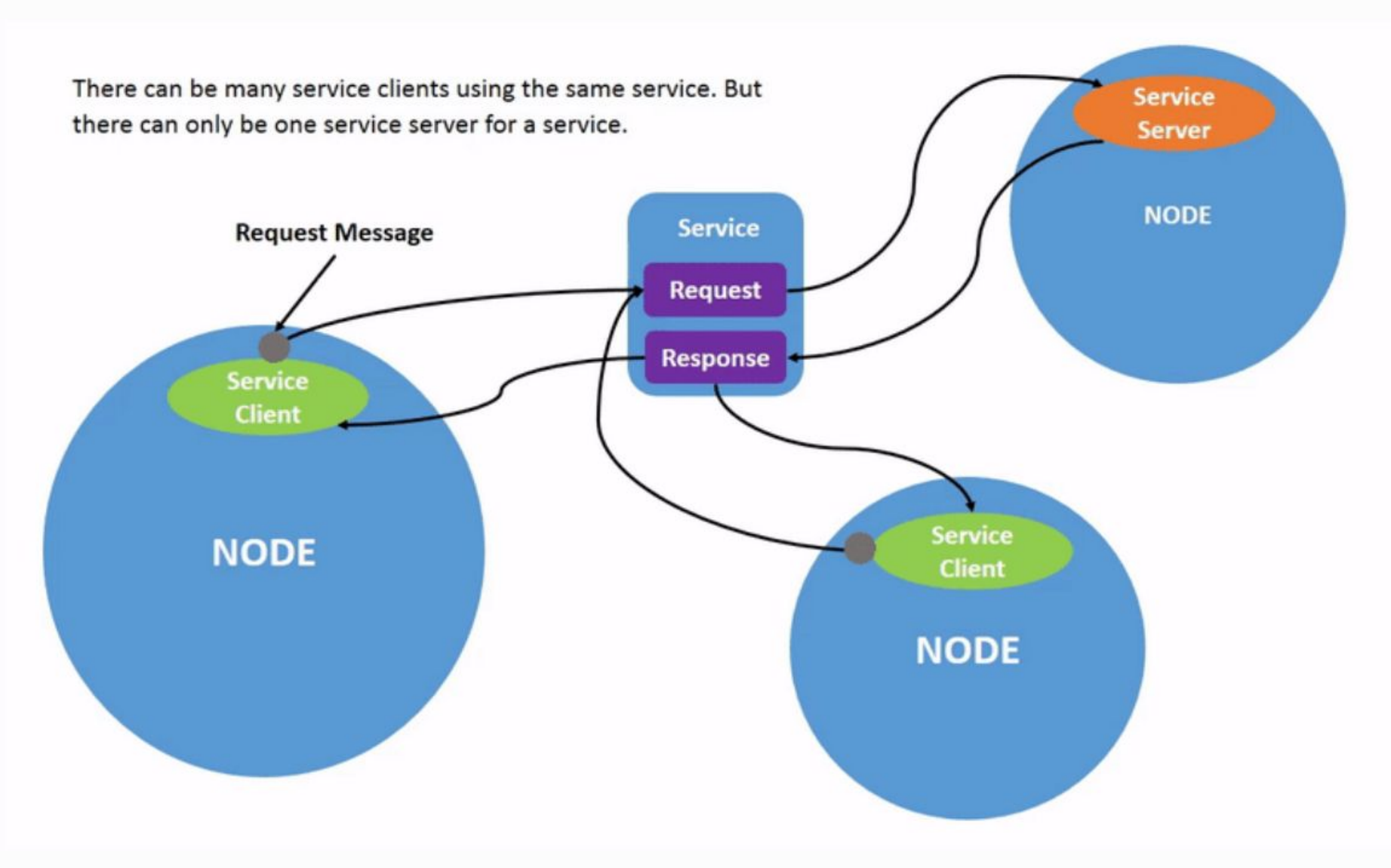
- 단발성 요청–응답 통신 제공
- 클라이언트가 요청(Request)을 보내면 서버가 처리 후 응답(Response)

특징:

- 동기식 호출
- 요청이 올 때까지 대기(wait_for_service)
- 한 번 처리 후 연결 종료



II. ROS2의 주요 구성 요소

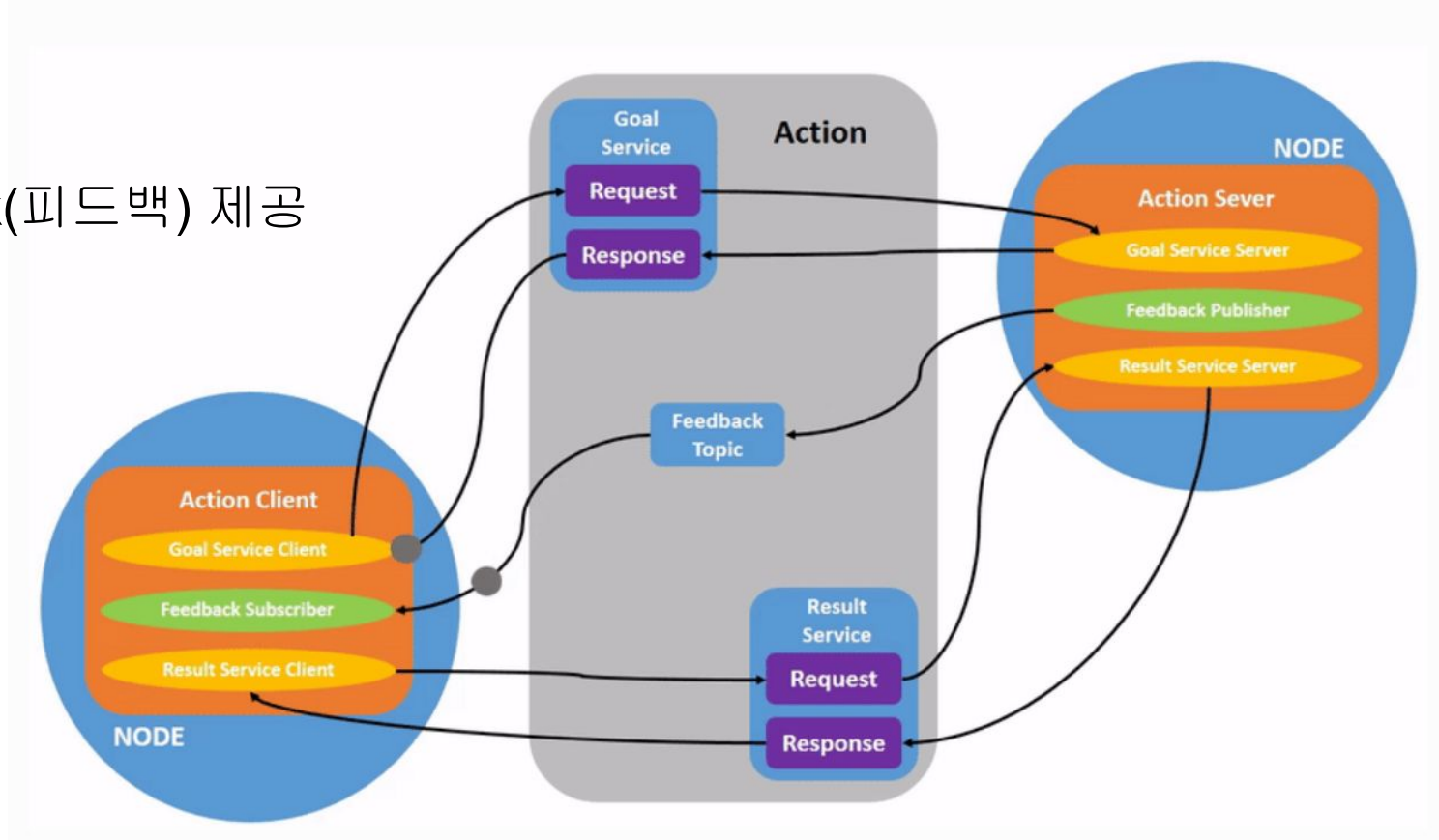


Ⅱ. ROS2의 주요 구성 요소

Action 노드 (Action Node)

- 완료까지 시간이 걸리는 작업을 비동기적으로 요청하고, 진행 상황을 모니터링하며 최종 결과를 받음

- **통신 패턴**:
  - 클라이언트가 Goal(목표) 전송
  - 서버가 수락 후 진행 Feedback(피드백) 제공
  - 완료 시 Result(결과) 반환
  - 필요 시 Goal 취소 요청 가능
- **특징**:
  - 비동기 처리
  - 진행 상태 확인(피드백)
  - 취소 지원



III. ROS2의 주요 특징

Ⅲ. ROS2의 주요 특징

DDS 기반 실시간 통신의 채택

- **산업 표준 데이터 중심 Publish/Subscribe 미들웨어**
  - OMG에서 정의한 오픈 표준(pub/sub)
  - 로봇·항공우주·자동차·금융·IoT 등 범용 활용
- **ROS2와의 통합**
  - RMW 인터페이스를 통해 eProsima Fast DDS, RTI Connext 등 선택 가능
  - QoS 설정, 자동 디스커버리, 멀티캐스트 전송 기능 활용
- **ROS1 대비 개선점**
  - ROS1: 자체 TCP/UDP 기반 → 실시간성·확장성·QoS 제어 한계
  - ROS2: DDS 기반 → 지연(latency), 신뢰성(reliability), 대역폭 제어 가능
- **적용 사례 : 표준 사용으로 인한 통합 및 연동이 용이하다**
  - 자율주행차 센서 데이터 브로드캐스트
  - 산업용 자동화 시스템 메시지 교환
  - 분산 시뮬레이션 및 멀티플랫폼 통신

III. ROS2의 주요 특징

DDS 기반 실시간 통신의 채택

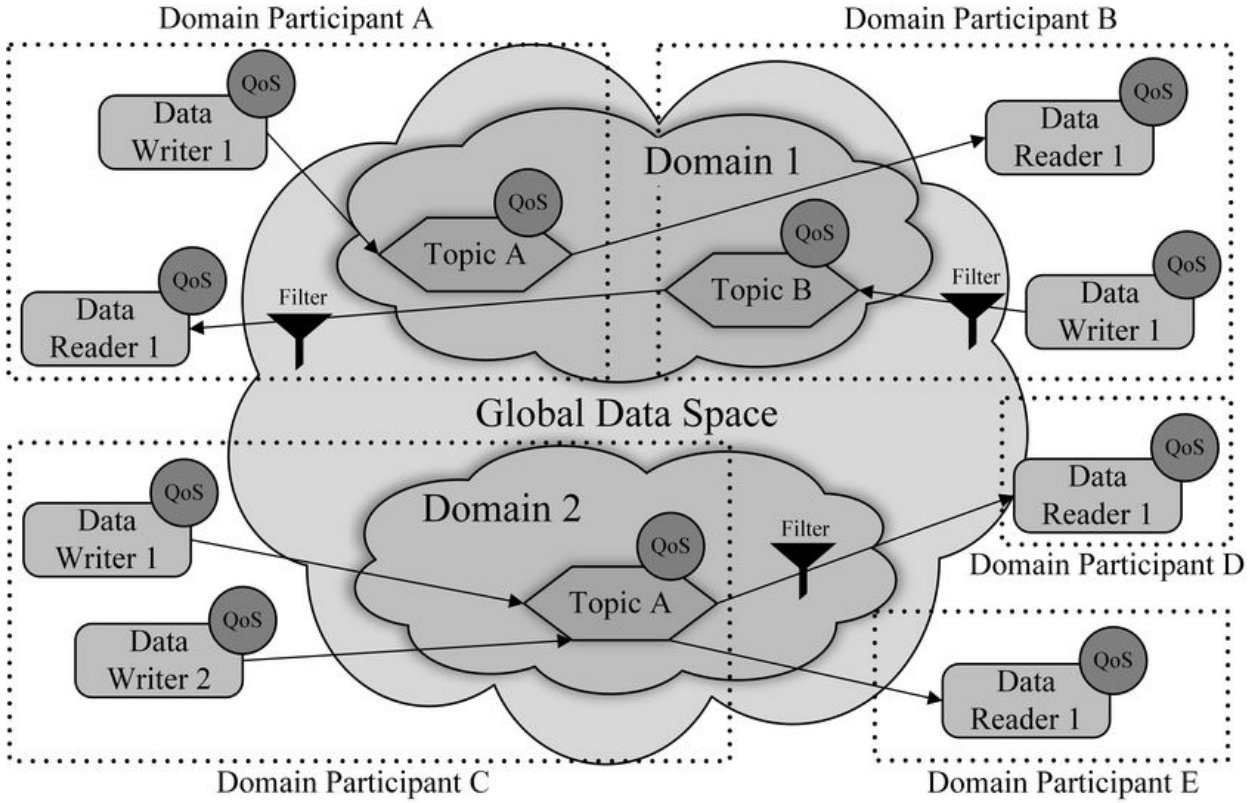
DDS(Data Distribution Service)는 분산 시스템에서

데이터 중심(pub/sub) 통신을 제공하는 미들웨어 표준입니다.

각 노드들은 **Participant** 단위로
DDS 네트워크에 참가,

Topic을 통해 데이터를 주고받음

이 모든 통신은 **Global Data Space**라는
추상 공간 위에서 이루어집니다.



Ⅲ. ROS2의 주요 특징

QoS(Data Distribution Service Quality of Service)

QoS는 DDS에서 데이터 전달 특성을 세밀하게 제어하기 위한 정책 집합입니다.

가장 자주 쓰는 QoS 3가지

- **Reliability:**
  - ***BEST_EFFORT*** **(빠르지만 손실 허용) vs. *RELIABLE* (재전송으로 손실 방지)**
- **Durability:**
  - ***VOLATILE*** **(메모리만) vs. *TRANSIENT_LOCAL* (발행 프로세스 내 저장)**
  - **→ 재시작 직후 최신값 받아야 한다면 TRANSIENT_LOCAL**
- **History:**
  - ***KEEP_LAST(N)*** **(최신 N개) vs. *KEEP_ALL* (모두)**
  - **→ 보통 상태 업데이트는 KEEP_LAST(1)**

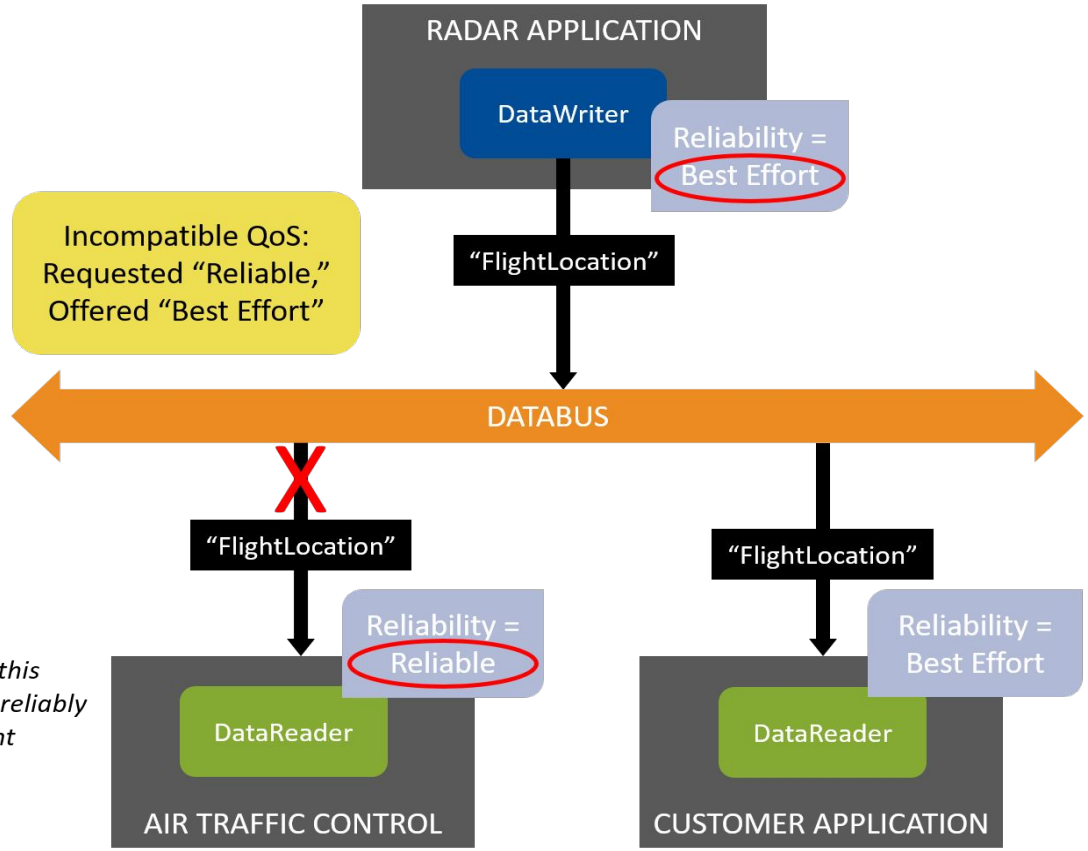
III. ROS2의 주요 특징

QoS 정책 설정의 예시

Best Effort: 메시지를 가능한 한 빨리 전송하지만, 손실이 발생해도 재전송하지 않음
Reliable: 메시지 손실 시 재전송을 시도하여 수신 보장

항공기 제어에 관련된 중요한 노드의 경우
메시지를 손실 할 경우 치명적인 문제를 일으키기 때문에 속도가 느리더라도,
확실한 메세지 수신을 보장하는
QOS정책을 적용

Customer Application 같은 메시지 손실이 치명적이지 않은 노드의 경우
QOS 정책을 느슨하게 가져감.



III. ROS2의 주요 특징

여러 Domain Participant가 Topic을 통해서

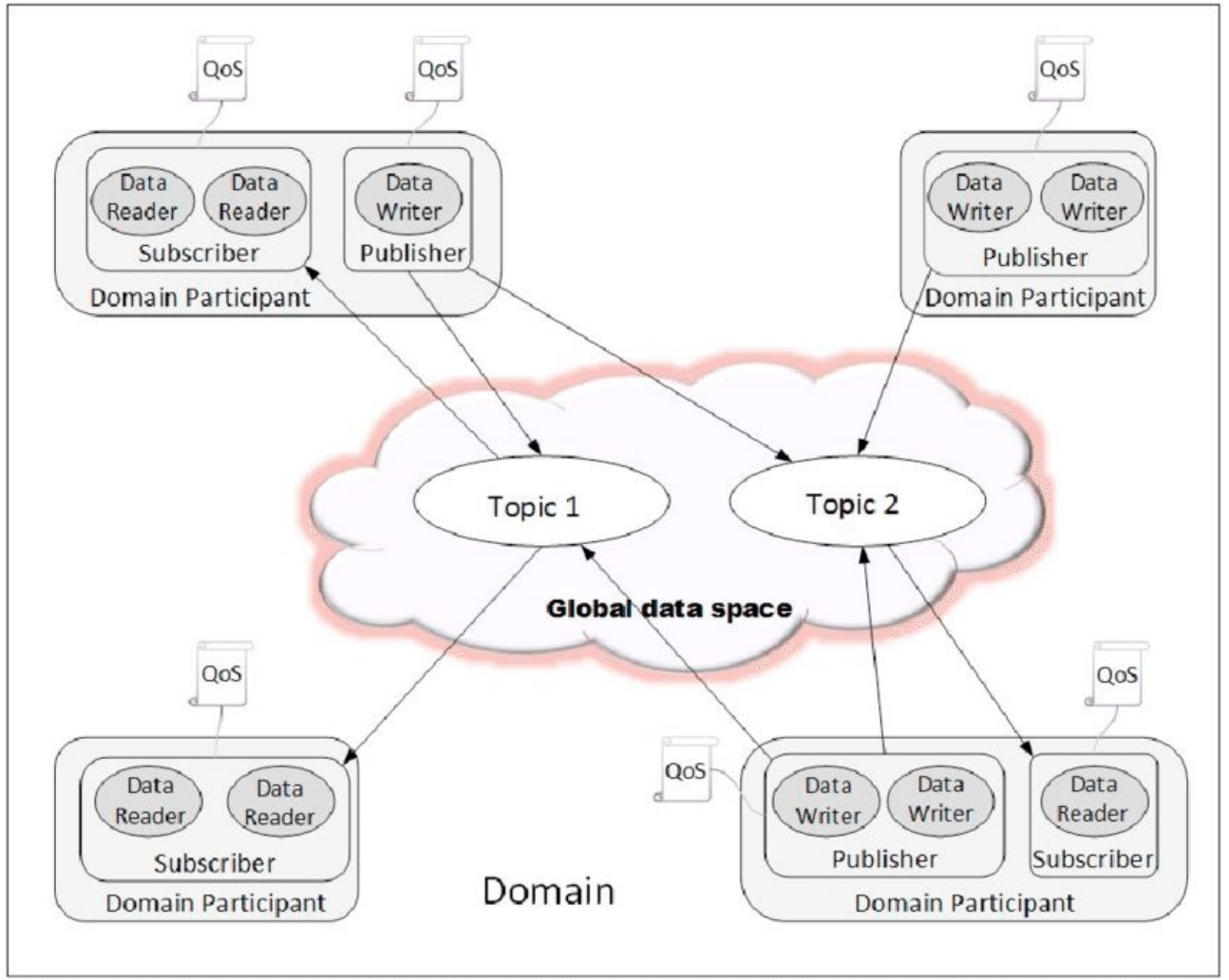
Global Data Space에 메시지를 게시 및 구독

1. Participant가 Topic별로 DataWriter(퍼블리셔)와 DataReader(서브스크라이버)를 생성

2. 퍼블리셔는 Global Data Space에 메시지를 쓰고, 구독자가 해당 Topic과 QoS 설정이 일치하는 경우 메시지를 읽음

3. Topic이 다르거나 QoS 미스매치일 때는 데이터가 무시됨

4. QoS 아이콘으로 Reliability, Durability 등 각 퍼블리셔·구독자의 정책 차이를 직관적으로 표현



Ⅲ. ROS2의 주요 특징

멀티 플랫폼 지원 (리눅스, 윈도우, macOS)

- **통합 빌드 시스템**
  - CMake + Colcon/Ament 사용
  - 한 번 작성 → 모든 OS에서 동일 빌드·배포
- **클라이언트 라이브러리**
  - `rclcpp`, `rclpy` 등 언어별 라이브러리 모두 지원
  - DDS 벤더(eProsima, RTI 등)도 각 플랫폼 제공
- **간편 설치**
  - Linux: `apt` (Debian/Ubuntu)
  - Windows: Chocolatey / MSI
  - macOS: Homebrew / Conda
- **컨테이너 & CI**
  - Docker / Windows Containers 이미지
  - GitHub Actions · Azure Pipelines로 크로스플랫폼 자동 테스트

IV. Docker란 무엇인가?

IV. Docker란 무엇인가?

- **가볍고 빠른 격리 환경**

  호스트 OS 커널을 공유해, 수초 만에 앱을 실행하는 '컨테이너' 제공

- **이미지 기반 패키지**

  애플리케이션 코드·라이브러리·설정 등을 하나의 파일(이미지)로 묶어 관리

- **Dockerfile로 자동화**

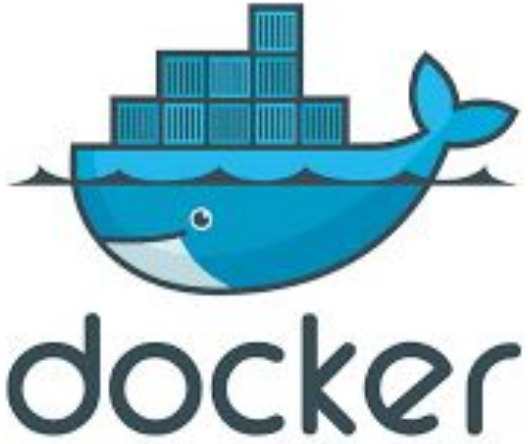
  텍스트 스크립트 한 줄로 환경 설정을 정의 -> 버전 관리·재현성 확보

- **개발→테스트→운영 일관성**

  "내 PC에서는 되는데 서버에서는 안 돼요" 문제 해소

- **공유와 협업 편리**

  Docker Hub 같은 공개 레지스트리에서 이미지 검색·다운로드 가능



IV. Docker란 무엇인가?

컨테이너와 가상 머신의 차이

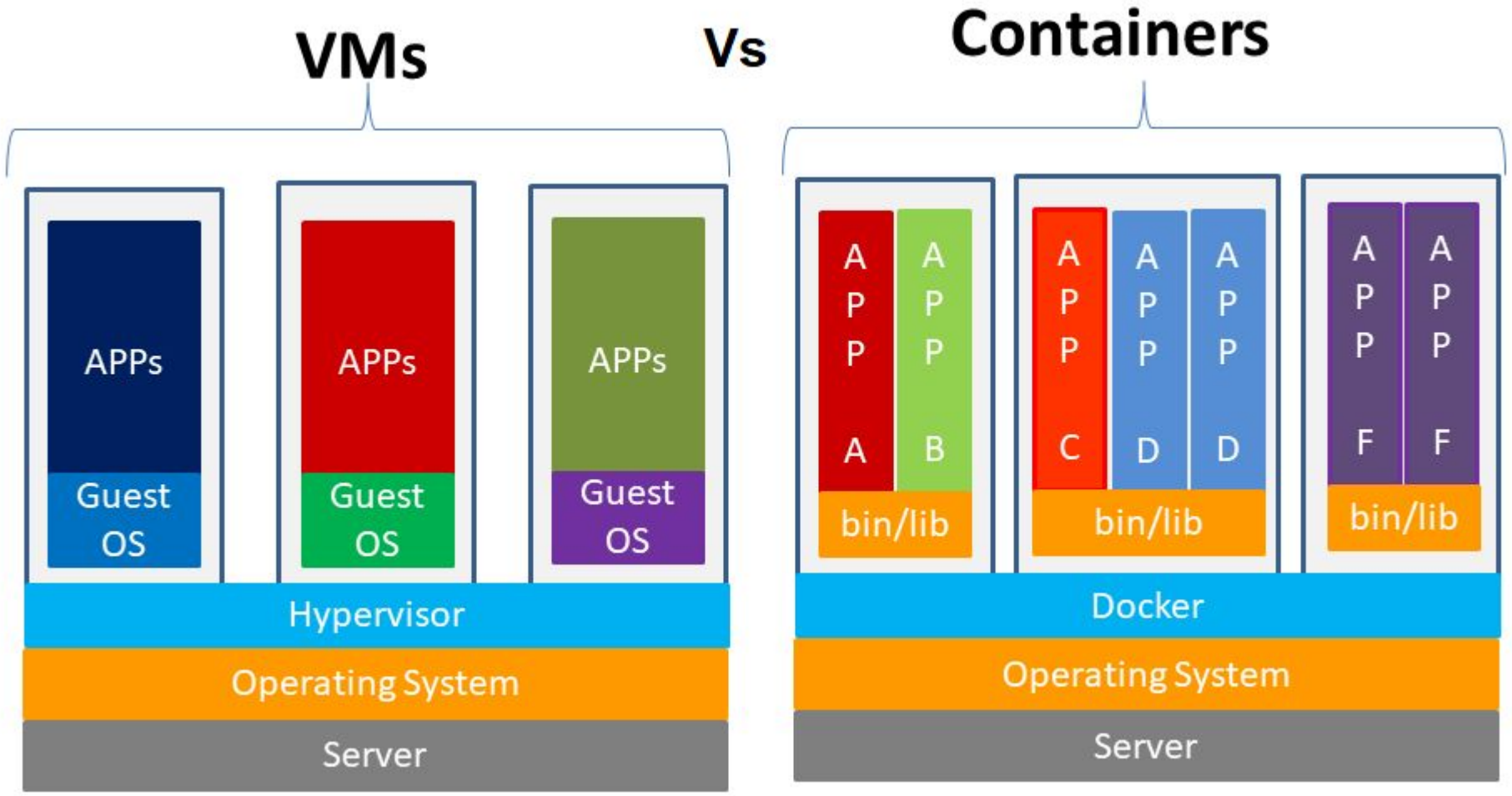
가상 머신(VM)

- 하이퍼바이저 위에 **별도의 운영체제(Guest OS)** 설치
- CPU·메모리·디스크 전용 할당 → 마치 독립된 서버 한 대와 같음
- 부팅 필요 → 수십 초~분 단위로 시작
- 이미지 크기 수 GB → 자원 부담

컨테이너(Container)

- 호스트 OS 커널 **공유** → 애플리케이션 실행에 필요한 라이브러리·파일만 격리
- 경량 프로세스 단위 격리 → 수십 MB 수준
- 부팅 과정 없이 프로세스 실행 → 수 밀리초~초 내 시작
- 이미지 크기 작아 빠른 다운로드·배포

IV. Docker란 무엇인가?



IV. Docker란 무엇인가?

구조 차이

- **가상 머신(VM):** 하이퍼바이저 위에 별도 OS 전체 설치 -> 마치 물리 서버 하나를 통째로 만드는 느낌
- **컨테이너:** 호스트 OS 커널 공유, 앱 실행에 필요한 파일 격리 -> 가볍게 프로세스 하나 띄우는 느낌

자원 사용량

- **VM:** 운영체제 전체가 메모리·디스크를 차지 -> 무겁고 비용 상승
- **컨테이너:** 애플리케이션 단위로 최소한의 파일만 포함 -> 자원 절약

시작 속도

- **VM:** OS 부팅 과정 필요 -> 수십 초~분 소요
- **컨테이너:** 프로세스 시작만 -> 수 밀리초~초 내 실행

이미지 크기

- **VM 이미지:** 수 GB
- **컨테이너 이미지:** 수십~수백 MB

IV. Docker란 무엇인가?

컨테이너의 장점 (경량, 빠른 배포)

경량(Lightweight)

- 호스트 OS 커널을 공유하여 별도의 운영체제 부팅 없이 애플리케이션만 격리
- 컨테이너 이미지 크기가 작아 필요한 파일만 포함 -> 다운로드·업로드 속도 빠름

빠른 배포(Fast Deployment)

- 컨테이너는 프로세스 수준에서 실행되므로, 수 초 이내에 기동 가능
- 코드 변경 후 재빌드·재배포 시간이 짧아 CI/CD 파이프라인에 최적

이식성(Portability)

- 개발 환경에서 만든 이미지가 테스트·운영 환경에서도 동일하게 동작
- “내 로컬에서는 되는데 서버에서는 안 돼요” 문제가 거의 사라짐

확장성(Scalability)

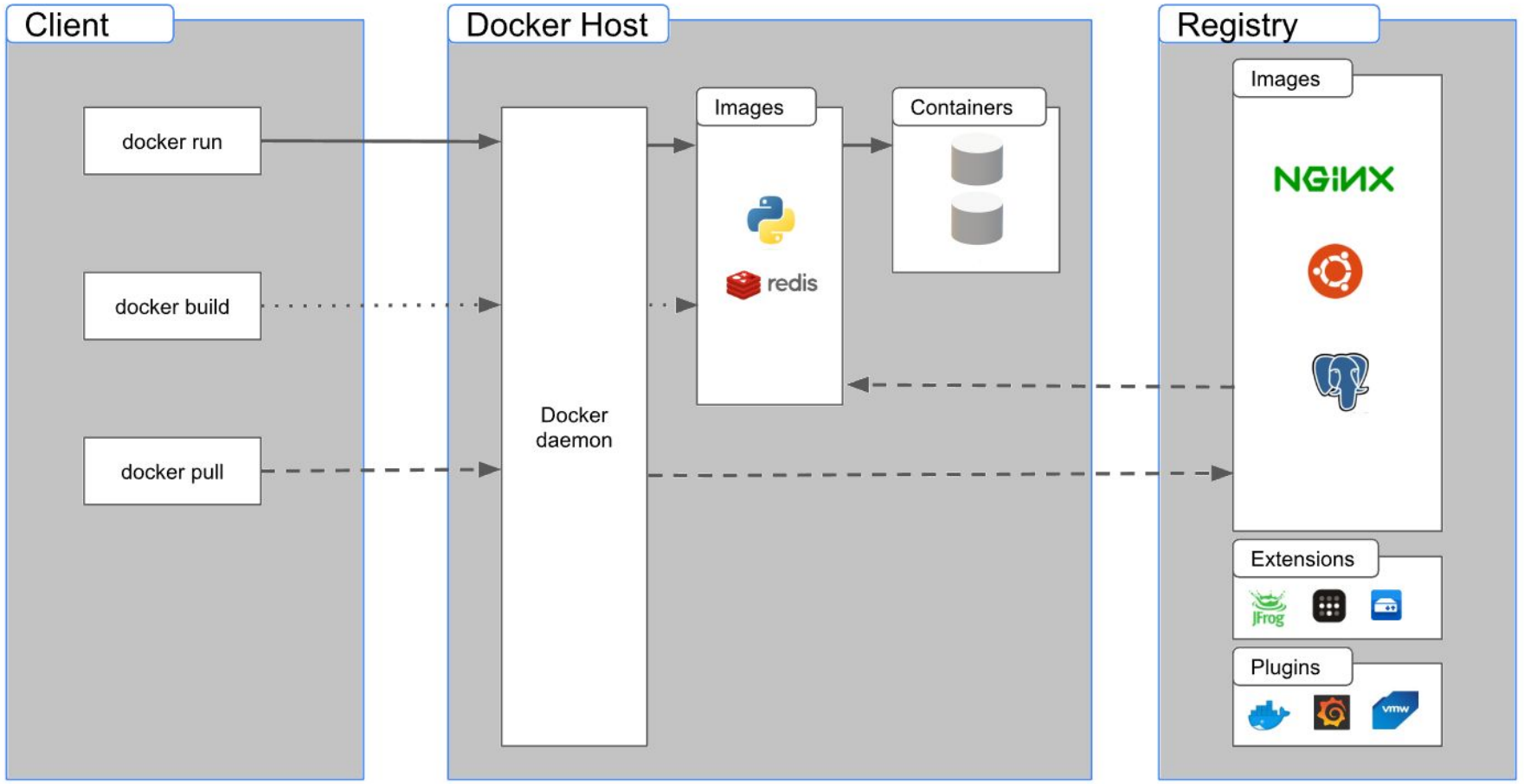
- 수평 확장이 용이해, 필요할 때 컨테이너 인스턴스를 쉽게 추가·제거 가능
- 마이크로서비스별로 독립 배포·관리할 수 있어 시스템 복잡도 감소 및 리소스 효율 향상

V. Docker의 기본 구성 요소

V. Docker의 기본 구성 요소

- **이미지 (Image)**
  - 애플리케이션 코드, 라이브러리, 설정 등을 하나로 묶은 읽기 전용 템플릿
  - Dockerfile로 정의 -> 재현 가능한 빌드
- **컨테이너 (Container)**
  - 이미지를 실행한 가벼운 인스턴스
  - 프로세스 격리 환경에서 독립 실행 -> 시작·중지가 빠름
- **Docker Hub**
  - 퍼블릭 레지스트리(저장소)
  - 공식·커뮤니티 이미지 검색, PUSH/PULL 지원
  - 프라이빗 레포지토리로 사설 이미지 관리 가능

V. Docker의 기본 구성 요소



VI.ROS2와 Docker

왜 Docker가 ROS 2 개발에 유용할까?

- **환경 일관성**: 누구나 동일한 이미지→“내 컴에선 돼요” 사라짐
- **의존성 격리**: 호스트에 라이브러리 뒤엉킴 없이 여러 버전 공존
- **재현성 높은 빌드**: Dockerfile만 있으면 몇 초 만에 완성
- **빠른 시작**: 컨테이너 수초 기동 vs. VM 수십 초
- **CI/CD 통합**: 이미지 빌드 -> 테스트 -> 배포 완전 자동화
- **멀티 플랫폼 테스트**: Ubuntu/Fedora/macOS 이미지 간편 전환

VI.ROS2와 Docker

Host OS

- 호스트 운영체제, Docker 엔진을 실행

Docker Container Runtime

- 네트워크, 볼륨, 네임스페이스 등 컨테이너 격리 기능

ROS 2 Base Image

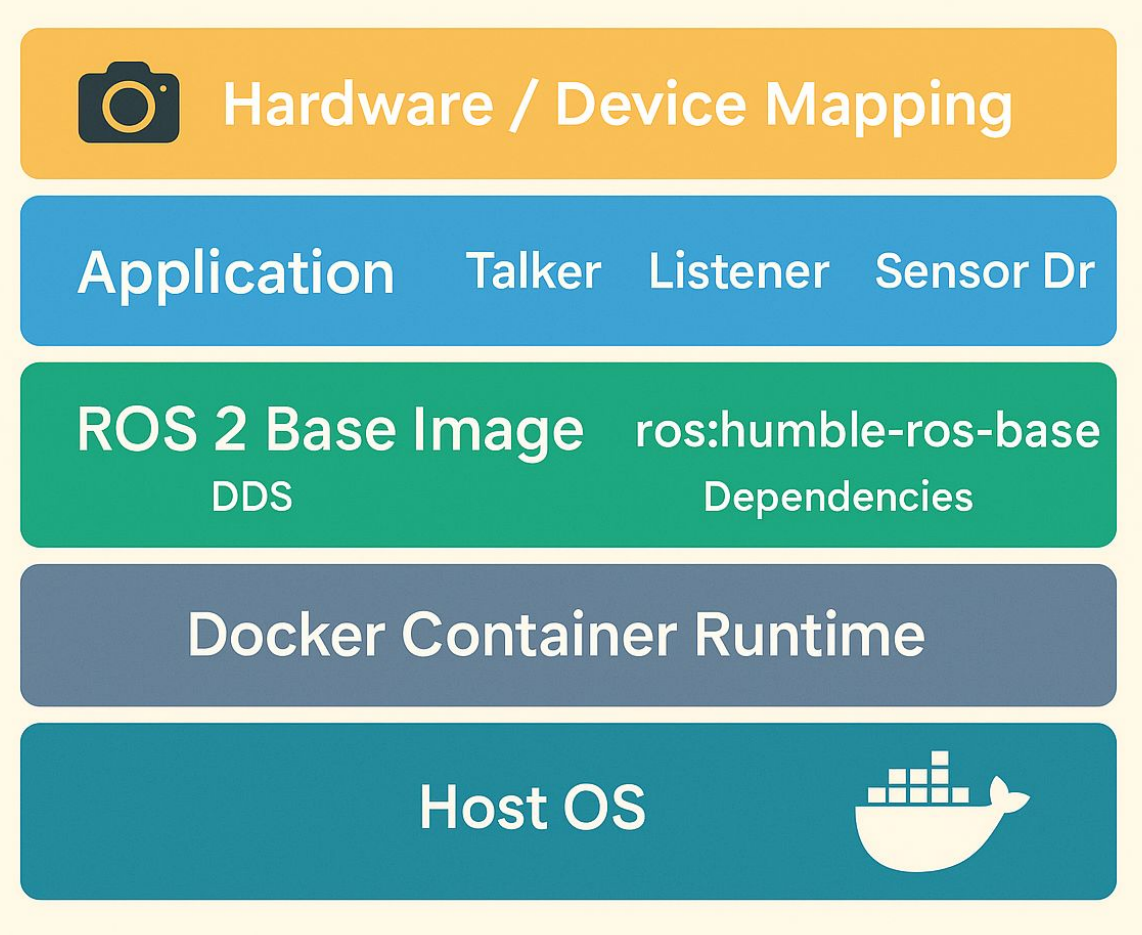
- ros:humble-ros-base (DDS, 공통 의존성 포함)

Application Layer

- 사용자 정의 노드(Talker, Listener, Sensor Driver)

Hardware / Device Mapping

- 카메라, 직렬 포트(/dev/ttyUSB0) 등의 실제 디바이스



THANK YOU