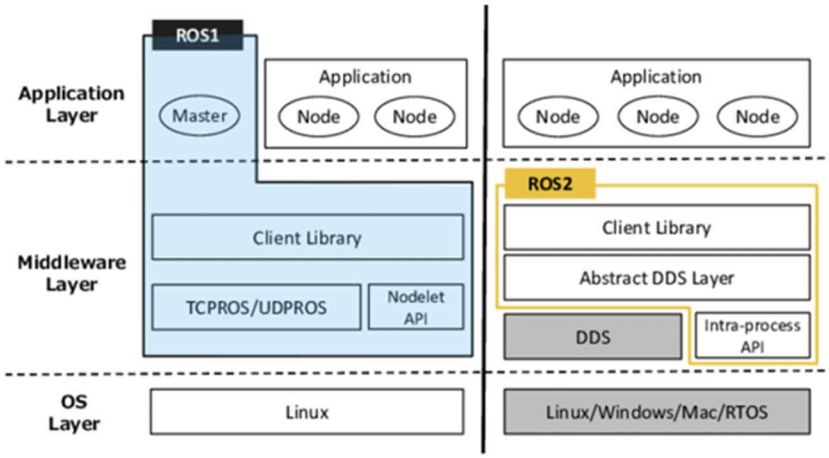
ROS1 vs ROS2 비교



-운영 체제

ROS1: Linux 전용

ROS2: Linux, Windows, macOS, RTOS 등 멀티 플랫폼 지원

-중앙 관리 시스템:

ROS1: Master 노드 필수 (중앙 집중형)

ROS2: Master 노드 없음 (분산형, discovery 자동화)

-노드 통신 방식

ROS1: **Master**가 Publisher-Subscriber 간 통신 중재

ROS2: DDS가 자동 discovery 및 직접 통신 수행

-확장성 및 실시간성:

ROS1: 제한적

ROS2: DDS 기반으로 확장성, 실시간성, 보안성 우수

ROS2 프로그래밍

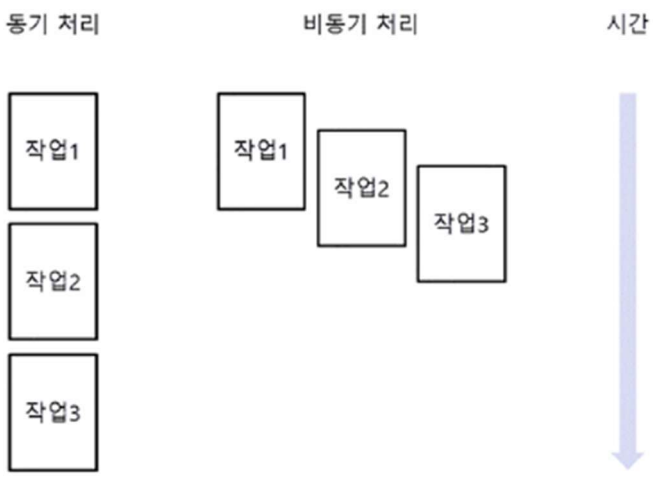
노드

ROS 시스템에서 기능을 수행하는 실행 단위 프로그램

Message (통신 방법)

토픽 (Topic) .msg

- 노드 간 통신 (publisher, subscriber)
- 비동기식 , 단방향
- 여러 Subscriber가 중복 구독 가능
- 지속적인 데이터



서비스 (Service) .srv

- 노드 간 통신

  (요청-응답 방식)

- 동기식
- 1대1 통신
- 짧은(처리빠른) 명령,데이터

액션 (Action) .action

- 동기/비동기 혼합형
- 장시간 실행되는 작업에 사용

프로그래밍 규칙

들여쓰기

- 공백 4칸을 사용
- 탭(Tab) 문자는 사용x

괄호 사용법

- 자료형에 따라 적절한 괄호를 사용합니다:

리스트: []

딕셔너리: {}

튜플: ()

문자열 표기

- 작은따옴표(') 사용

코드 검사 도구

- **ament_flake8**을 사용해 스타일을 자동으로 검사

Underlay vs Overlay

Underlay

- 기본 ROS2 설치 환경 또는 하위 워크스페이스
- 수정 불가

Overlay

- 기존 underlay 위에 덮어쓰는 상위 워크스페이스
- 새로운 패키지를 추가하거나 수정
- 수정 가능

패키지

package.xml

- 메타 정보 포함 (신분증 역할)

setup.cfg

- 선언적 방식
- 주로 사용되는 방식
- 간단하고 명확한 패키지 설정

```
# setup.cfg
[metadata]
name = my_python_pkg
version = 0.1.0
```

setup.py

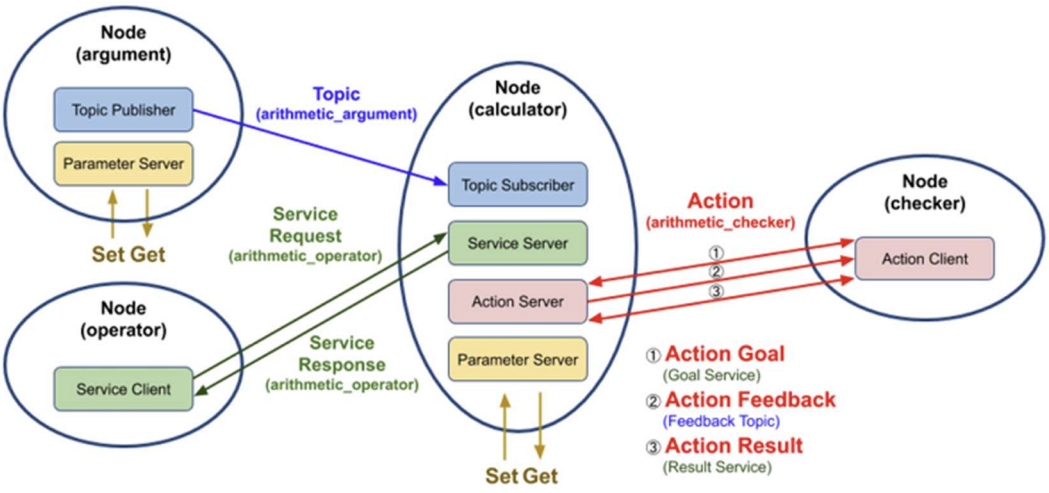
- 프로그래밍 방식
- 패키지 의존성을 해결 불가

```
# setup.py
from setuptools import setup

package_name = 'my_python_pkg'

setup(
    name=package_name,
    version='0.1.0',
```

패키지 구성



argument_node

토픽 Publish

현재 시간과 연산에 사용할 변수 a, b를 publish

operator_node

서비스 Client

연산자 (+, -, *, /)를 Service 요청으로 calculator_node에 전달함

calculator_node

토픽 Subscriber: argument_node → a, b 값 수신

서비스 Server: operator_node → 연산자 수신

연산 수행(결과값 반환) <-> checker_node

목표값 초과 시, 최종 결과 및 수식들을 액션 결과로 반환

checker_node

액션 Client

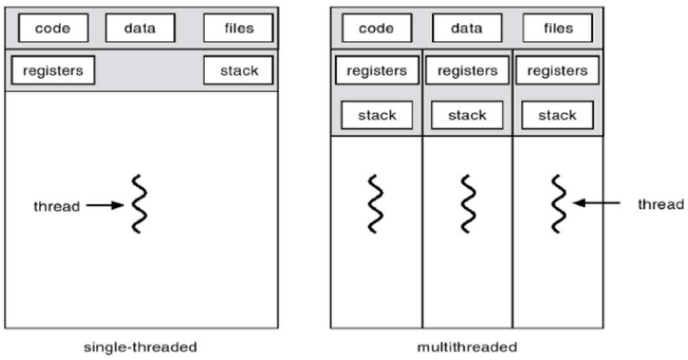
누적값이 도달해야 할 목표값(goal) 을 설정
calculator_node와 Action 통신

rclpy

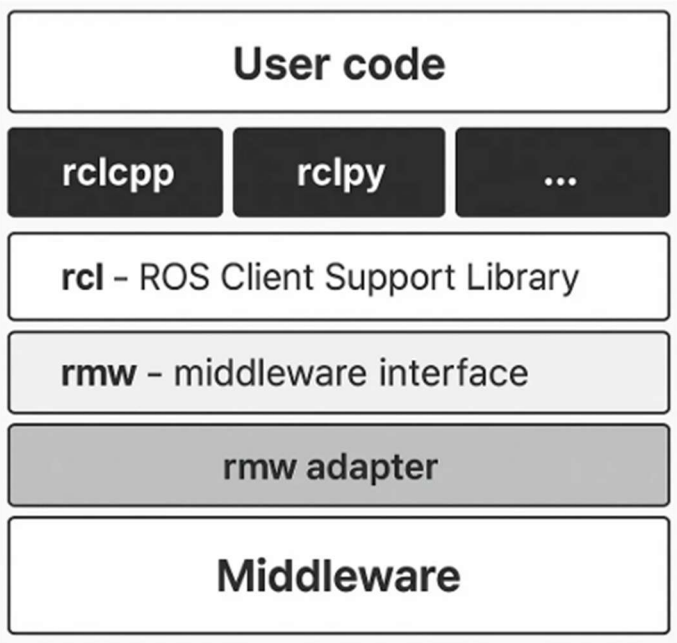
- ROS 2 의 Python 클라이언트 라이브러리

- Python을 이용해서 노드를 생성, ROS 2 통신 기능 사용 가능하게 해줌

- 멀티스레드 지원



ROS2 계층구조



User code

- 사용자가 작성한 코드

rcl (rclcpp, rclpy)

- 클라이언트 라이브러리

- 사용자 코드와 미들웨어를 연결

rmw

- 미들웨어 인터페이스

rmw adapter

Middleware

- 실제 메시지 전달과 QoS 설정을 처리 하는 계층