ROS2 day4 hw1 결과보고서

2025407012/로봇학부/송연우

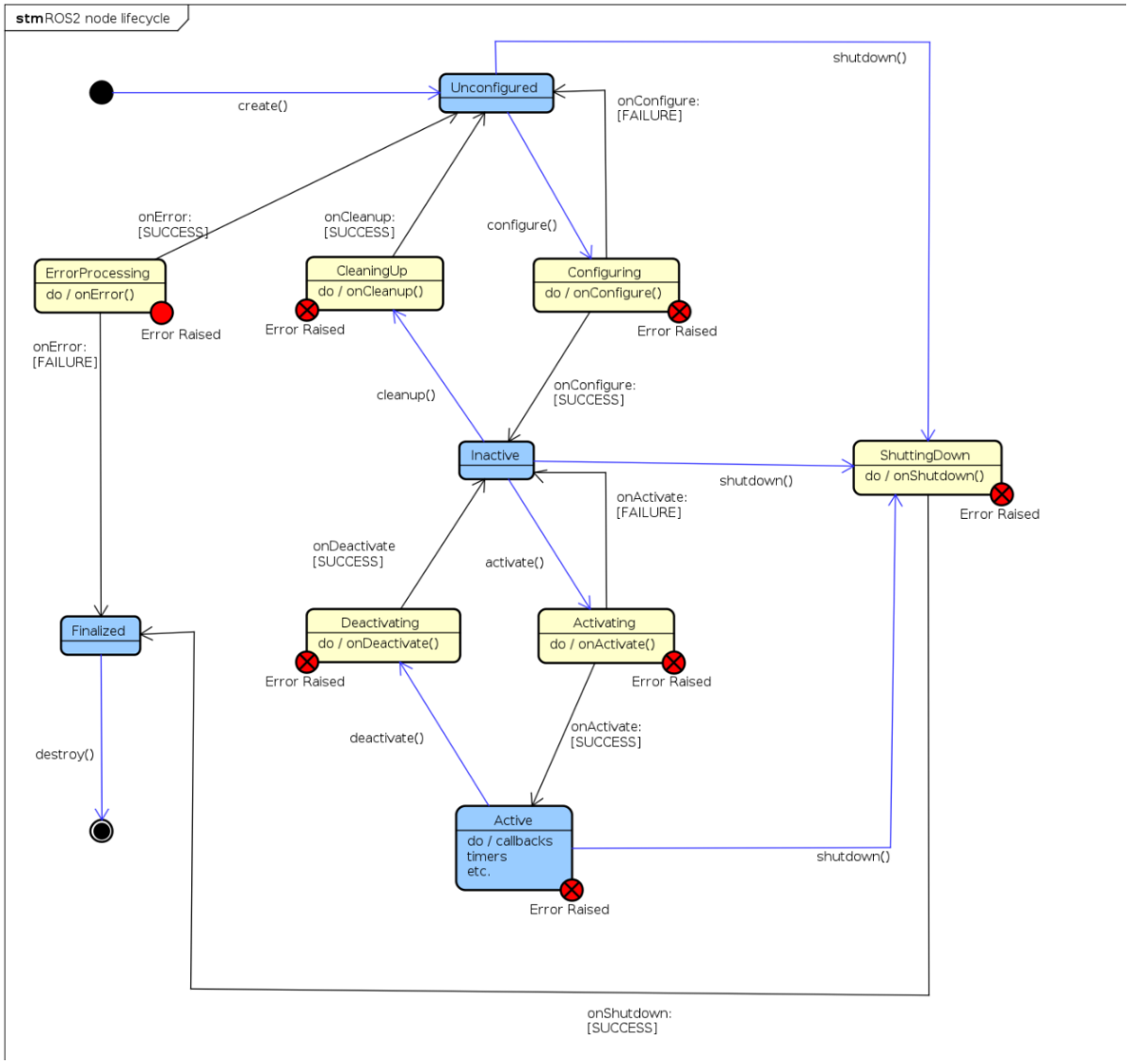
목차

1. managed node

관리되는 노드는 일반 노드와 달리 수명을 가지고, 활성화/비활성화와 같은 여러 상태에 존재할 수 있습니다. 이러한 노드는 로봇의 작동 과정에서 그때그때 필요한 노드만을 활성화시키기 위해 사용되며, 효율적인 동작에 기여합니다.

ros2에서 관리되는 노드는 총 4가지의 주요 상태를 가지며 4가지 상태 사이에서 6개의 전환 상태를 추가로 가집니다. 아래 사진에서 파란색으로 표시되는 부분이 주요 상태이며, 노란색으로 표시되는 부분이 추가적으로 가지는 상태입니다.

생명주기



각 주요 상태에는 Unconfigured, Inactive, Active, Finalized 가 있으며 각자의 상태에서 이동할 수 있는 전환 상태가 정해져 있습니다.

2. Qos

Qos 는 서비스 품질의 약자로 ros2에서는 publisher, subscriber, service server, service client 에 대해 Qos 를 설정할 수 있습니다. 기본적인 Qos 종류는 총 8가지로 History, Depth, Reliability, Durability, Deadline, Lifespan, Liveliness, Lease Duration 이 있습니다.
다음은 각 프로필에 대한 설명입니다.
- History: 최대 N 개의 메시지만을 저장하거나, 모든 메시지를 저장할 수 있습니다.
- Depth: 기록을 마지막으로 유지하기로 설정했을 경우에만 적용되며, 설정하는 깊이만큼의 메시지를 저장합니다.
- Reliability: 메시지를 전달하는 쪽에 초점을 둘지 메시지의 신뢰성에 초점을 둘 지 정할 수 있습니다.
- Durability: 일시적으로 늦게 가입한 구독자에 대한 메시지를 유지하거나, 아예 유지하지 않도록 설정할 수 있습니다.
- Deadline: 토픽에 후속적인 메시지가 게시되는 데 걸리는 최대 예상 시간을 설정할 수 있습니다.
- Lifespan: 메시지가 만료된 것으로 간주되지 않도록 메시지를 게시/수신할 때까지 걸리는 최대 시간입니다.
- Liveliness: 노드 중 하나가 메시지를 게시하면 자동으로 해당 게시자가 활성화 상태라고 인지하거나, 수동으로 호출 후 활성화 상태를 인지하도록 하는 방식을 선택할 수 있습니다.
- Lease Duration: 시스템에서 게시자가 활성 상태에서 벗어났다고 간주하기 전까지의 최대 기간입니다.

사용자는 기본적으로 제공되는 설정을 이용하거나, 직접 설정을 변경해 사용할 수 있습니다.

3. 구현 내용

위 내용을 학습하고 나서, lifecycle 을 가지는 노드에서 publisher, subscriber 에 Qos 를 설정하는 코드를 작성했습니다. 먼저 lifecycle 에서는 publisher 부분에서의 6개 상태를, Qos 는 reliability, durability 부분을 설정했습니다.

1) hpp

(1) publisher
종속성으로는 <chrono>, <functional>, <memory>, <string>, "rclcpp/rclcpp.hpp", "std_msgs/msg/string.hpp", "rclcpp_lifecycle/lifecycle_node.hpp" 등을 사용했습니다.

```
13  class Pub : public rclcpp_lifecycle::LifecycleNode
```

rclcpp_lifecycle 을 상속하는 자식 클래스인 LifecycleNode 를 상속한 Pub 클래스를 선언했습니다. 해당 클래스의 public 함수로는 각각의 주요 상태 사이의 중간 상태 5가지의 함수가 있습니다. private 에서는 메시지를 발행할 timer_callback 함수가 있습니다.

(2) subscriber

```
13  class Sub : public rclcpp::Node
```

publisher 와 마찬가지로 종속성을 가지며 rclcpp 를 상속하는 Node 클래스를 부모 클래스로 가집니다. 나머지 부분은 일반적인 subscriber 의 헤더파일과 동일하게 public 함수로는 Sub(생성자)를 가지고 private 부분에서 topic_callback 함수와 subscriber 를 선언합니다.

2) cpp

(1) publisher
생성자에서는 보통의 publisher 와는 다르게 노드 이름과 count 값을 초기화하고 다른 코드는 넣지 않습니다. publisher 와 타이머를 초기화하고 메시지를 발행하는 것은 active 상태에서 실행되므로, 생성자에서는 해당 작업을 건너뛰고 이를 on_configuare 함수에서 실행합니다.

```
6     Pub::Pub()
7         : rclcpp_lifecycle::LifecycleNode("pub"), count(0){}
```

on_configure 함수는 unconfigure 과 inactive 사이 중간 단계로 코드는 다음과 같습니다.
1.

```
    auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
```

먼저 종속성 설정을 위한 변수 qos_profile 을 선언하고 , depth 관련 설정을 10으로 해 메시지를 10개까지 저장하도록 설정합니다.

2.

```
qos_profile.reliability(RMW_QOS_POLICY_RELIABILITY_RELIABLE);
```

다음으로 신뢰성보다는 전달을 보장하도록 설정합니다.

3.

```
qos_profile.durability(RMW_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL);
```

내구성의 경우에는 최근의 것만 저장하도록 합니다.

4.

```
publisher = this->create_publisher<std_msgs::msg::String>("topicname", qos_profile);
```

Qos 설정 후 해당 변수인 qos_profile 을 발행자 생성 함수에 매개변수로 넣습니다.

5.

```
timer = this->create_wall_timer(1s, std::bind(&Pub::timer_callback, this));
```

타이머로 몇 초 간격으로 timer_callback 함수를 불러올 지 설정합니다.

6.

```
RCUTILS_LOG_INFO_NAMED(get_name(), "Configuring from state %s.", state.label().c_str());
```

전 단계에서부터 전환되었다는 메시지를 출력합니다.

7.

```
return rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn::SUCCESS;
```

위 작업이 성공적으로 끝나면 다음 값을 반환합니다.

8. 다음으로 on_activate()함수입니다.

on_activate()함수에서는 내장되어있는 메소드인 on_activate()를 실행한 후,

```
publisher->on_activate();
```

메시지 출력과 값 반환이 이루어집니다. on_activate()는 publisher 를 활성화시킵니다.

9. on_deactivate()함수입니다.

```
publisher->on_deactivate();
```

마찬가지로 on_deactivate()함수를 실행시킨 후 나머지를 on_activate()와 동일하게 실행합니다.

10. on_cleanup()함수입니다.

```
timer.reset();
```

여기서는 타이머와 발행자를 리셋합니다.

```
publisher.reset();
```

11.
on_shutdown 에서는 on_cleanup() 함수와 같은 작업(타이머, 발행자 리셋)을 실행합니다.

12. on_error 함수입니다.

```
RCUTILS_LOG_INFO_NAMED(get_name(), "something went wrong!");
```

에러메시지를 출력하고

```
return rclcpp_lifecycle::node_interfaces::LifecycleNodeInterface::CallbackReturn::FAILURE;
```

FAIURE 를 반환합니다.

13. timer_callback()함수입니다.

```
auto message = std_msgs::msg::String();
```

메시지를 생성한 후,

```
message.data = "Say Hello #" + std::to_string(count++);
```

값을 정해줍니다. 이때 발행자가 활성화 상태가 아니라면

```
RCLCPP_INFO(
    get_logger(), "Lifecycle publisher is currently inactive. Messages are not published.");
```

메시지가 발행되지 않고 있음을 출력합니다.

활성화 상태라면,

```
RCLCPP_INFO(get_logger(), "Lifecycle publisher is active. Publishing: [%s]", message.data.c_str());
```

메시지를 출력해 줍니다.

나머지 코드는 main 함수로, 일반 노드에서와 달리 spin()함수에서의 변화가 있습니다.

```
rclcpp::spin(pub->get_node_base_interface());
```

get_node_base_interface() 함수는 노드의 기능에 접근할 수 있는 클래스 객체를 가져오는 역할을 합니다. 따라서 이 함수를 사용함으로서 lifecycle node 설정이 가능해집니다.

(2) subscriber

1. 구독자에서는 일반 노드와 같은 구성을 가지고 있으며, qos_profile 변수를 사용한 depth 설정 코드만이 변경되었습니다.

```
auto qos_profile = rclcpp::QoS(rclcpp::KeepLast(10));
```

2. 생성자입니다.
qos_profile 변수와 구독자를 생성합니다.

```
subscriber = this->create_subscription<std_msgs::msg::String>(
```

3.

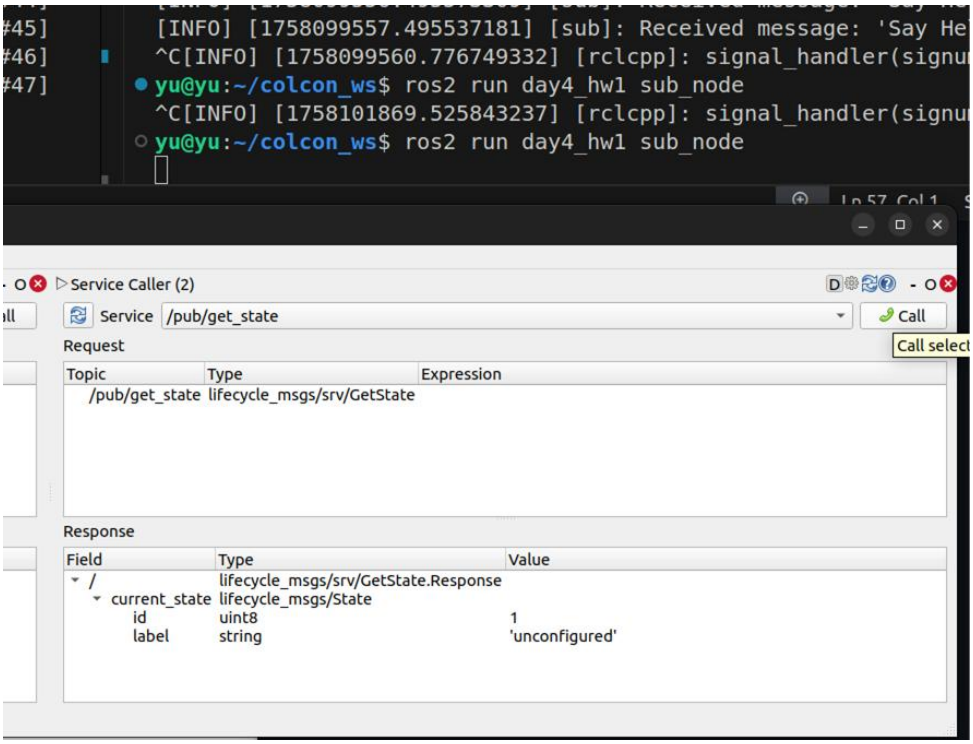
```
RCLCPP_INFO(this->get_logger(), "Received message: '%s'", msg->data.c_str());
```

topic_callback 함수입니다. 수신한 메시지를 터미널에 출력합니다.

나머지 코드는 main 함수입니다. 구독자 노드의 경우에는 일반 노드의 코드로 이루어져있습니다.

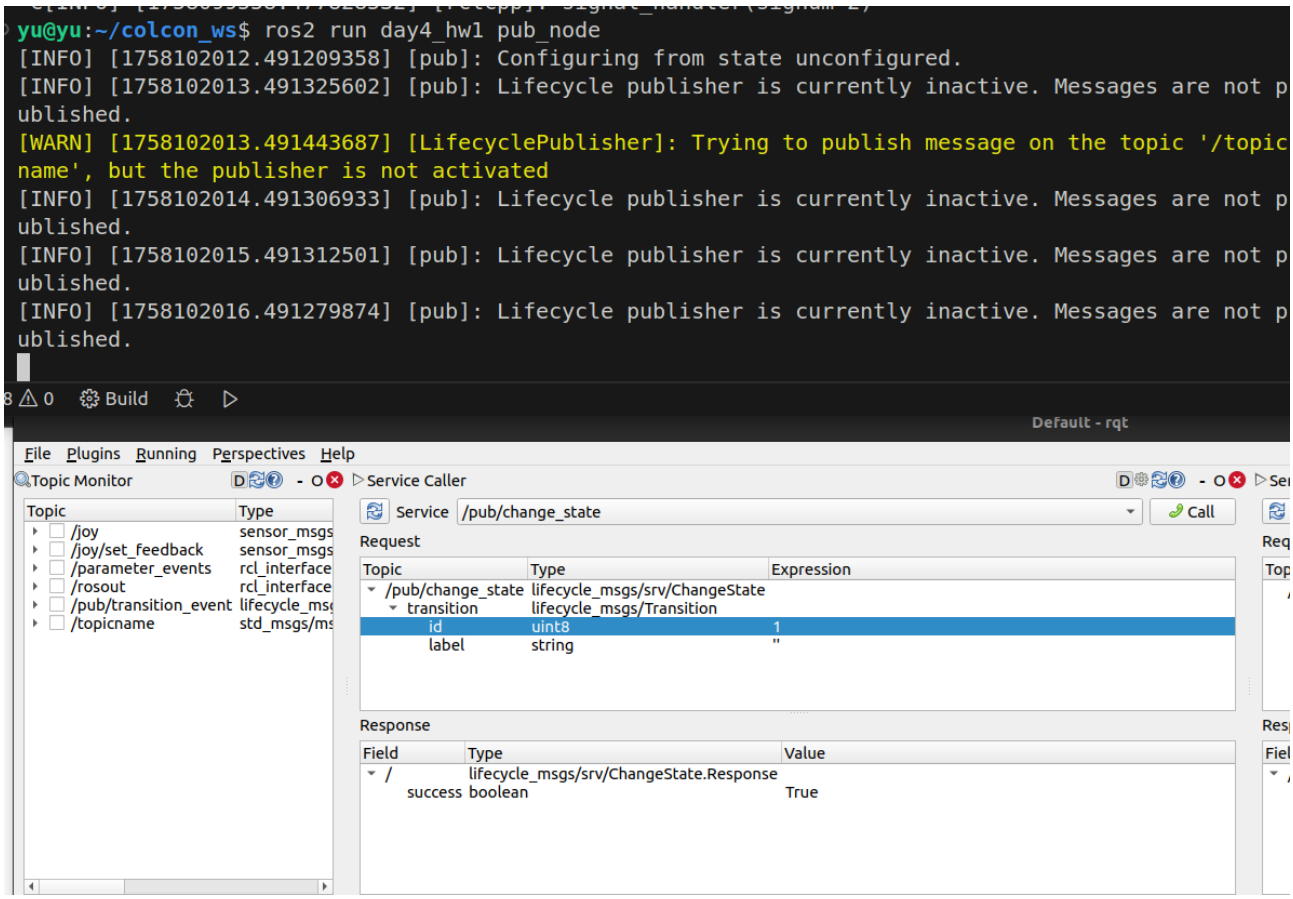
```
rclcpp::spin(node);
```

3) 실행 화면
다음은 터미널로 rqt 를 실행한 후 service call 로 발행자의 상태를 변화시키는 부분입니다. Lifecycle 클래스를 사용하면 위 기능을 사용할 수 있습니다.

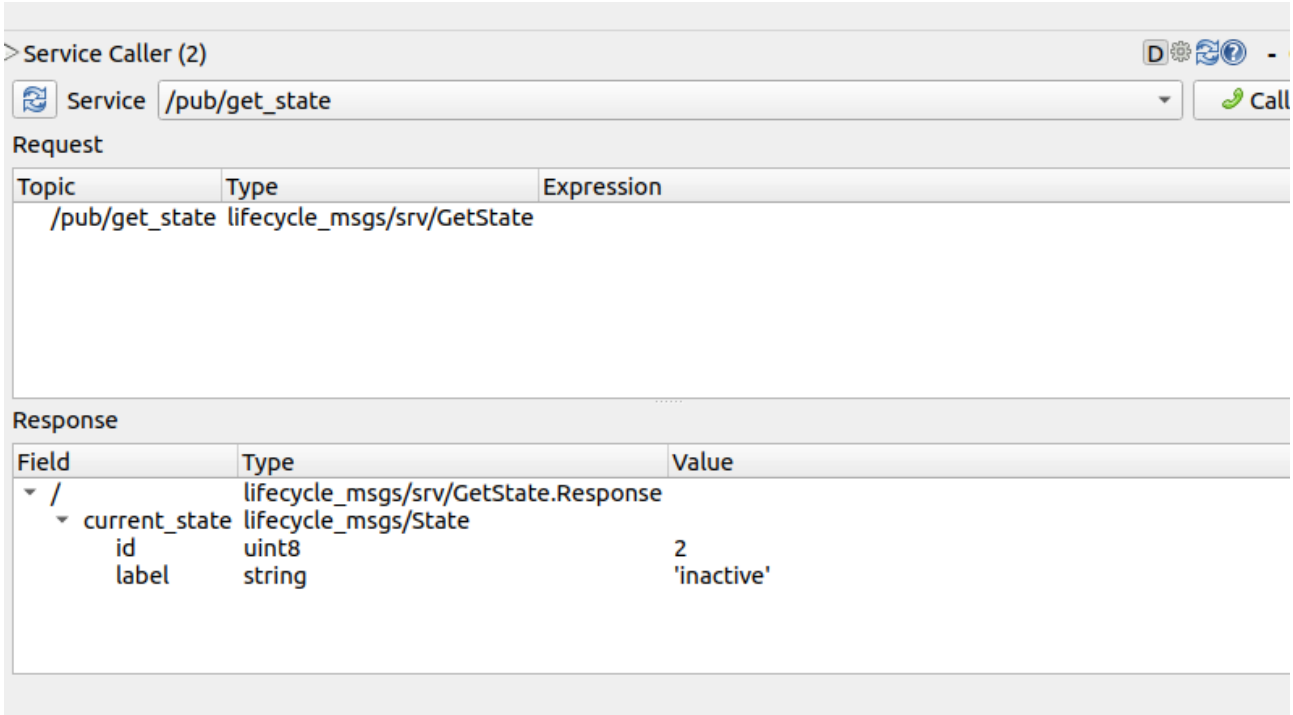


처음 발행자와 구독자 노드를 실행시키고 나면 발행자에서 메시지를 자동으로 발행하지 않습니다. service 창의 *pub/*get_state 에서 call 을 누르면 현재 노드의 상태(unconfigured)를 표시해 줍니다.

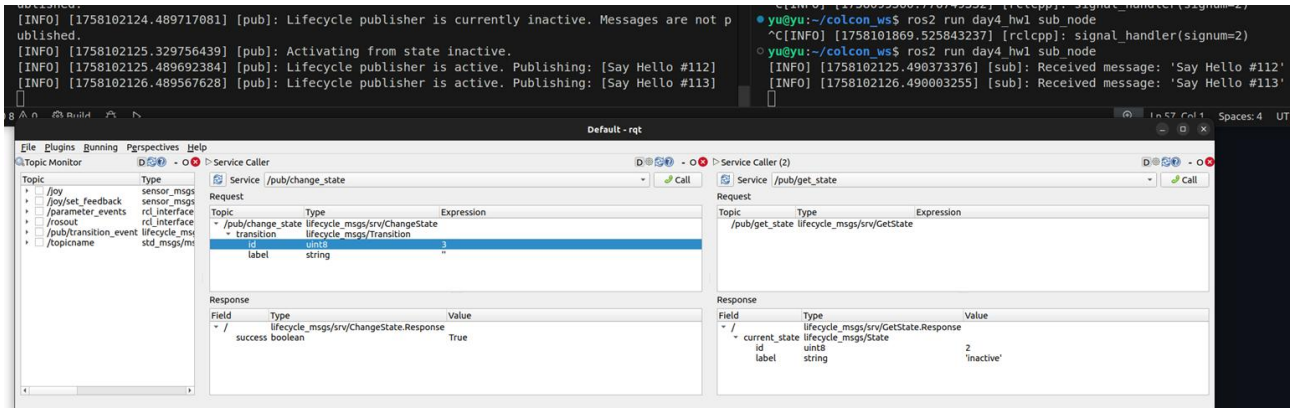
여기서 1을 입력한 뒤 다시 call 하면 true(전환 가능)을 표시하고

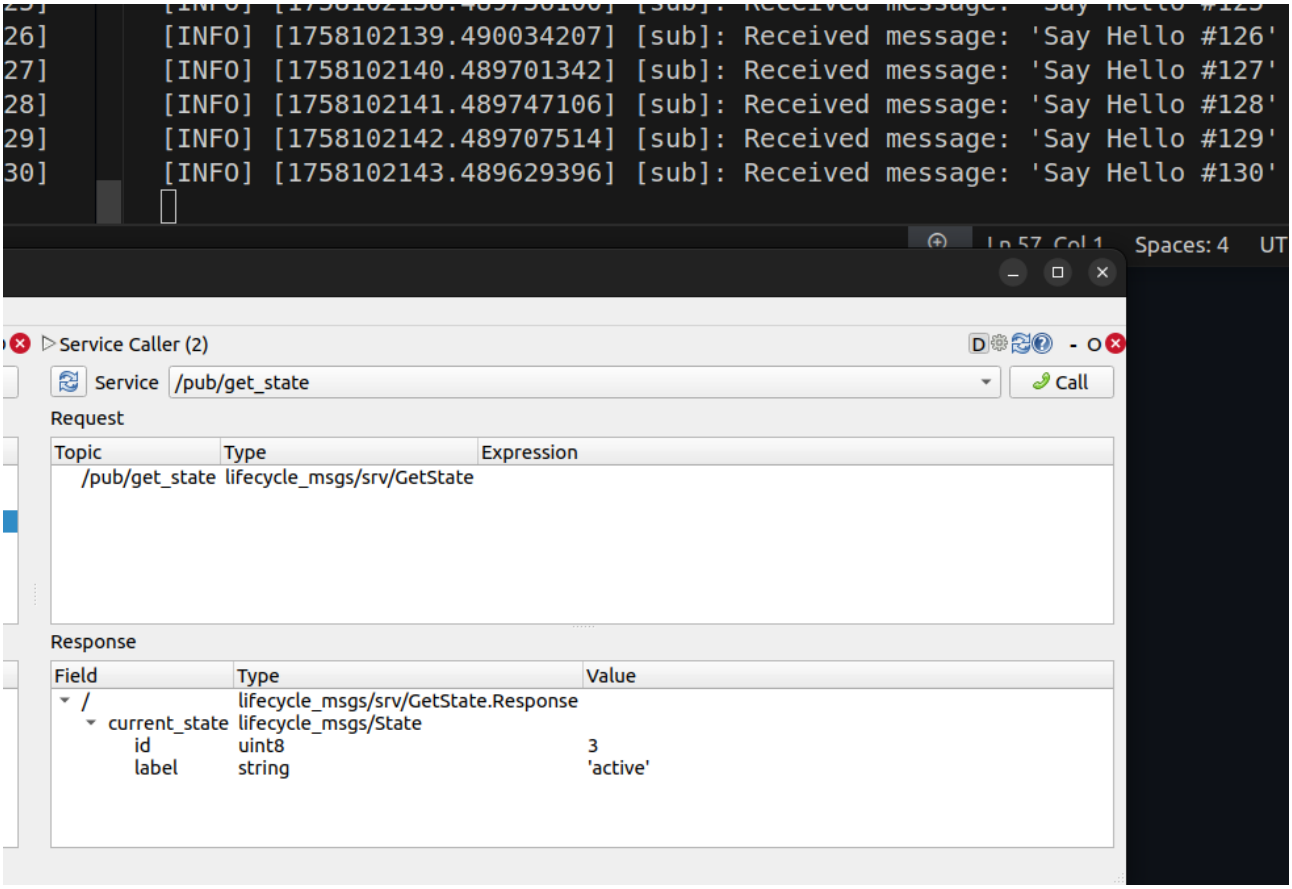


inactive 상태로 전환합니다.

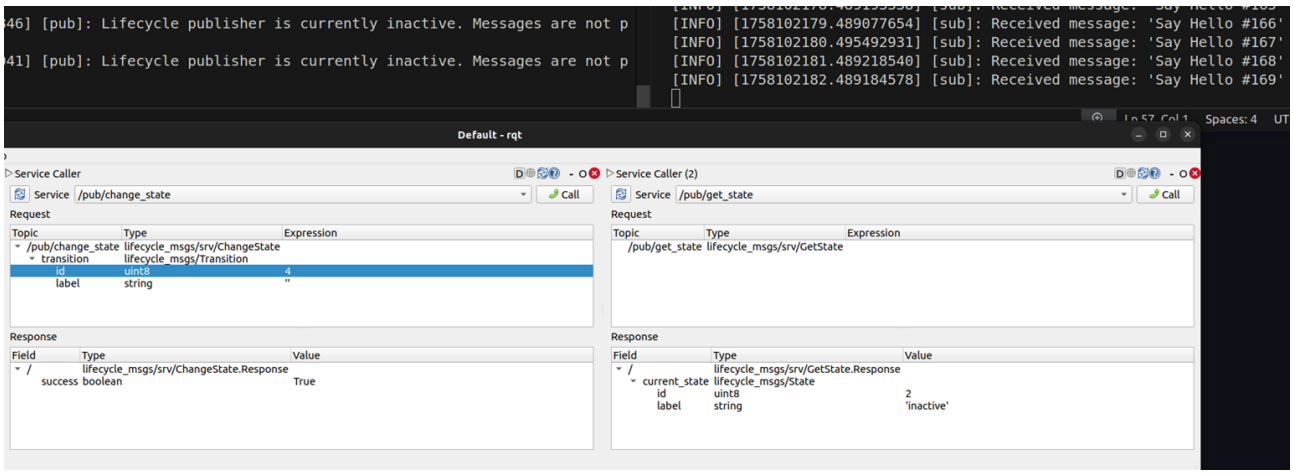


다음으로, 3을 호출하면 메시지를 발행하고, active 로 전환합니다.

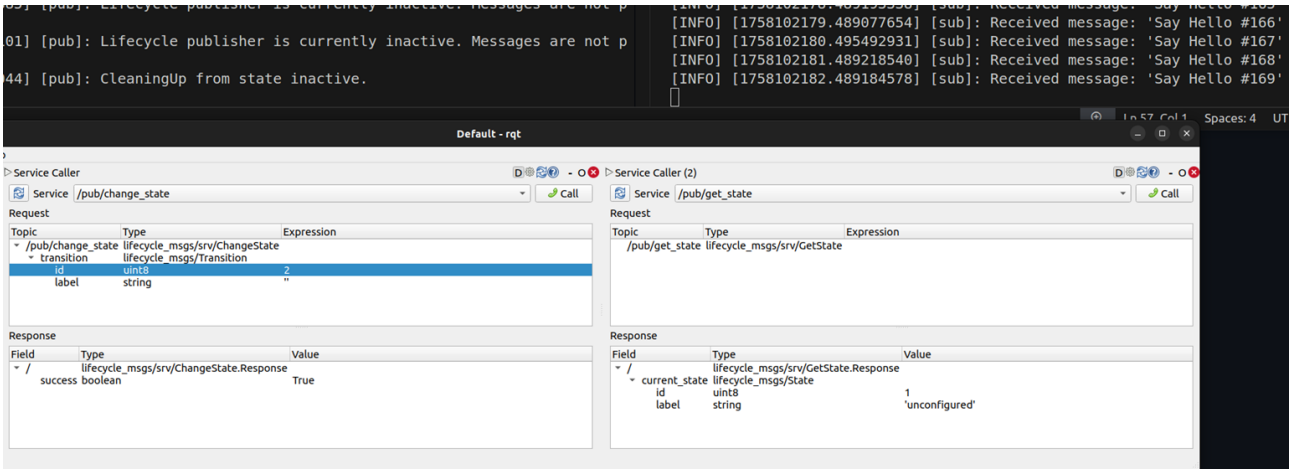




다시 4를 호출하면 inactive 로 돌아갑니다.



2를 호출하면 unfigured 로 돌아가고,



5의 경우에는 finalized 로 전환합니다.