Understanding ROS 2 Services: The Calculator Example

In ROS 2, while topics handle continuous data streams, services provide synchronous request-response communication - perfect for commands that need immediate results. Think of services as remote function calls between nodes: one node asks a question, another provides the answer.

The Calculator Analogy:

Imagine you're in a restaurant 🍽️:

- **You (Client): "What's 5 + 3?" → *Service Request***
- Waiter (Service): Takes your question to the kitchen
- Chef (Server): Calculates the answer
- Waiter: "It's 8!" → ***Service Response***

What We're Building:

In this exercise, we'll create a Calculator Service with two nodes:

- `calculator_server`**: The "math expert" that waits for calculation requests**
- `calculator_client`**: The "student" that asks math questions and gets answers**

Key Service Characteristics:

- **🔄 Synchronous: Client waits for the response**
- **🎯 One-to-One: One request gets one response**
- **⚡ Immediate: Perfect for commands and calculations**
- **✅ Guaranteed: Request either succeeds or fails clearly**

This pattern is essential for robot commands (start/stop), configuration changes, and any operation where you need to wait for a result before proceeding.

Now let's build our calculator service step by step! 🚀

Step 1: Create Package

```bash
# Navigate to workspace src
cd ~/ros2_ws/src

# Create package with correct dependencies
ros2 pkg create simple_service --build-type ament_python --dependencies
rclpy example_interfaces
```

Step 2: Create Server Code

File: `~/ros2_ws/src/simple_service/simple_service/server.py`

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts  # 🎯 Built-in service

class SimpleServer(Node):
    def __init__(self):
        super().__init__('simple_server')
        # 🎯 Create service
        self.srv = self.create_service(AddTwoInts, 'add_two_ints',
self.add_callback)
        self.get_logger().info('Server ready!')

    def add_callback(self, request, response):
        self.get_logger().info(f'Received: {request.a} + {request.b}')
        response.sum = request.a + request.b
        return response

def main():
    rclpy.init()
    node = SimpleServer()
    rclpy.spin(node)
    rclpy.shutdown()
```

```
if __name__ == '__main__':
    main()
```

Step 3: Create Client Code

File: `~/ros2_ws/src/simple_service/simple_service/client.py`

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from example_interfaces.srv import AddTwoInts  # 🎯 Built-in service

class SimpleClient(Node):
    def __init__(self):
        super().__init__('simple_client')
        # 🎯 Create client
        self.cli = self.create_client(AddTwoInts, 'add_two_ints')

        # Wait for service
        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Waiting for server...')

    def call_service(self, a, b):
        # 🎯 Create request
        request = AddTwoInts.Request()
        request.a = a
        request.b = b

        # 🎯 Send request async (returns future)
        future = self.cli.call_async(request)

        return future

def main():
    rclpy.init()
    client = SimpleClient()

    # Send request
    future = client.call_service(5, 3)

    # 🎯 Wait for response
    rclpy.spin_until_future_complete(client, future)

    if future.result() is not None:
        response = future.result()
        client.get_logger().info(f'Result: {response.sum}')
    else:
        client.get_logger().error('Service call failed')

    client.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Step 4: Verify Package Configuration

File: `~/ros2_ws/src/simple_service/package.xml`

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypelocation="http://download.ros.org/schema/package_format3.xsd
package_format3.xsd"?>
<package format="3">
  <name>simple_service</name>
  <version>0.0.0</version>
  <description>Simple service example</description>
  <maintainer email="you@example.com">Your Name</maintainer>
  <license>Apache License 2.0</license>

  <depend>rclpy</depend>
  <depend>example_interfaces</depend>  <!-- 🎯 CRITICAL -->

  <buildtool_depend>ament_python</buildtool_depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
```

```
</package>
```

File: `~/ros2_ws/src/simple_service/setup.py`

```
python
```

```
from setuptools import setup

package_name = 'simple_service'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='your_name',
    maintainer_email='your_email@example.com',
    description='Simple service example',
    license='Apache License 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'server = simple_service.server:main',
            'client = simple_service.client:main',
        ],
    },
)
```

Step 5: Build Package

```
bash
```

```
# Navigate to workspace root
```

```
cd ~/ros2_ws

# Build the package
colcon build --packages-select simple_service

# Source the workspace
```

```
source install/setup.bash
```

Step 6: Run the Example

Terminal 1 - Start Server:

```
bash
ros2 run simple_service server
```

Output:

```
text
[INFO] [1700000000.000000000] [simple_server]: Server ready!
```

Terminal 2 - Run Client:

```
bash
ros2 run simple_service client
```

Output:

```
text
[INFO] [1700000000.100000000] [simple_client]: Waiting for server...
[INFO] [1700000000.200000000] [simple_client]: Result: 8
```

Terminal 3 - Test with CLI:

```
bash
```

```
# List services
ros2 service list

# Call service directly
ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 10,
b: 20}"
```

```
# Show service info
ros2 service type /add_two_ints
```

Exercise: Create a Custom Calculator Service Interface

Objective

In this exercise, you will learn how to create and use custom service interfaces in ROS 2. Instead of using the built-in `example_interfaces.srv.AddTwoInts`, you will create your **own service definition** called `AddTwoInts.srv` that extends the functionality to support multiple mathematical operations.