Instalacao do ROS 2

auto node = std::make\_shared<RoboNewsStationNode>(); *// Cria uma instância compartilhada do nó.*

DESTINATION lib/${PROJECT\_NAME})

robot\_news\_station

install(TARGETS

ament\_target\_dependencies(robot\_news\_station rclcpp example\_interfaces)

add\_executable(robot\_news\_station src/robot\_news\_station.cpp)

<depend>example\_interfaces</depend>

}

return 0; *// Retorna 0 indicando sucesso.*

rclcpp::shutdown(); *// Encerra o ambiente ROS 2.*

rclcpp::spin(node); *// Mantém o nó rodando para processar eventos (como o timer).*

rclcpp::init(argc, argv); *// Inicializa o ambiente ROS 2.*

{

int main(int argc, char \*\*argv) *// Função principal que inicializa e executa o nó.*

};

rclcpp::TimerBase::SharedPtr timer\_; *// Ponteiro compartilhado para o timer. // Correção para SharedPtr.*

rclcpp::Publisher<example\_interfaces::msg::String>::SharedPtr publisher\_; *// Ponteiro compartilhado para o publicador. // Correção para SharedPtr.*

std::string robot\_name\_; *// Variável que armazena o nome do robô (ex.: "R2D2").*

}

publisher\_->publish(msg); *// Publica a mensagem no tópico "robot\_news".*

msg.data = "Hello, aqui é " + robot\_name\_ + " from the robot news station"; *// Define o conteúdo da mensagem com o nome do robô. // Correção da string original.*

auto msg = example\_interfaces::msg::String(); *// Cria uma nova mensagem do tipo String.*

{

void publishNews() *// Método privado que publica as notícias.*

private:

}

RCLCPP\_INFO(this->get\_logger(), "Robot News Station has been Started"); *// Registra uma mensagem informando que o nó foi iniciado.*

timer\_ = this->create\_wall\_timer(0.5s, std::bind(&RoboNewsStationNode::publishNews, this)); *// Cria um timer que chama publishNews a cada 0.5 segundos.*

publisher\_ = this->create\_publisher<example\_interfaces::msg::String>("robot\_news", 10); *// Cria um publicador para o tópico "robot\_news" com mensagens do tipo String e fila de 10.*

{

RoboNewsStationNode() : Node("robot\_news\_station"), robot\_name\_("R2D2") *// Construtor do nó, inicializa com o nome "robot\_news\_station" e define robot\_name\_ como "R2D2".*

public:

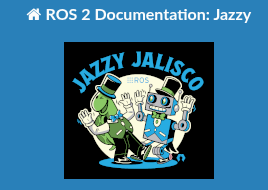
{

class RoboNewsStationNode : public rclcpp::Node *// Define uma classe que herda de rclcpp::Node, criando um nó ROS 2 chamado robot\_news\_station.*

using namespace std::chrono\_literals; *// Permite o uso de literais como 0.5s para definir durações, compatível com C++14 ou superior.*

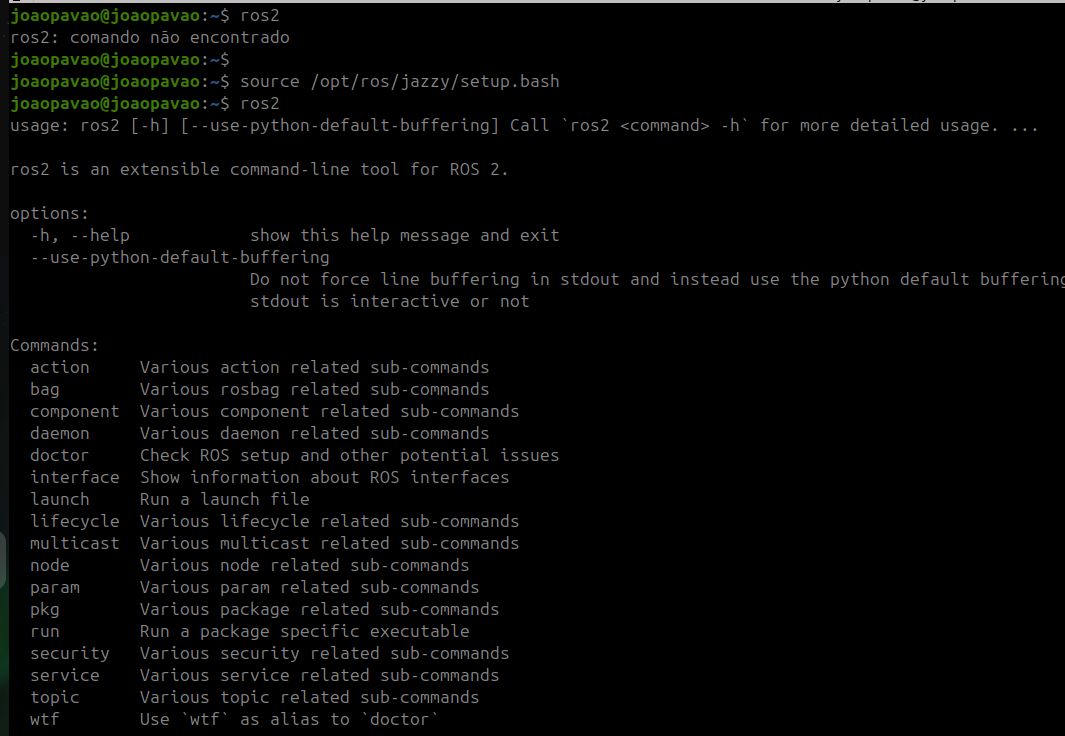
#include "example\_interfaces/msg/string.hpp" *// Inclui a mensagem String do pacote example\_interfaces, usada para enviar texto.*

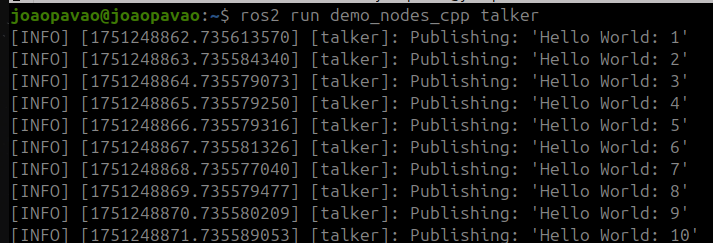
#include "rclcpp/rclcpp.hpp" *// Inclui a biblioteca principal do ROS 2 em C++ para funcionalidades como nós, publicadores e timers.*

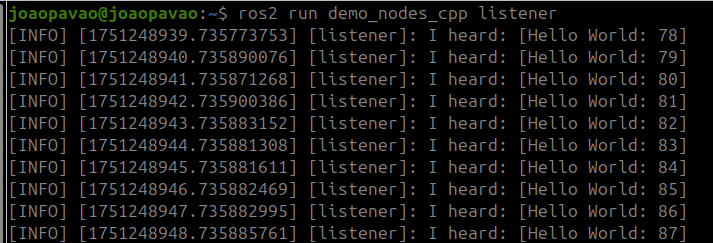


<https://docs.ros.org/en/jazzy/Installation/Ubuntu-Install-Debs.html>

Apos instalar, algumas configuracoes

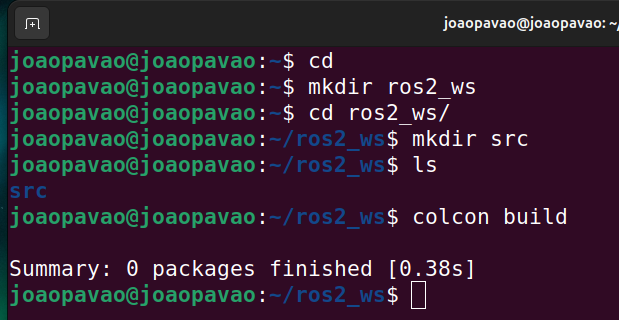
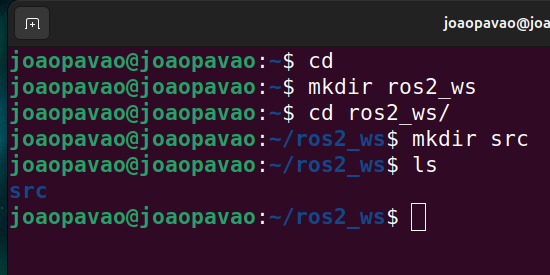
Primeiro programa no ROS 2

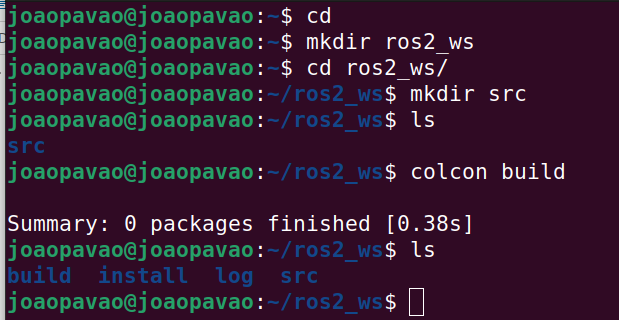




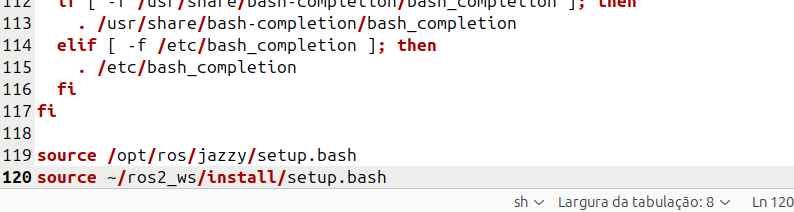
Registo do dia 31 de junho

tudo o que precisamos para a criacao de um workspace

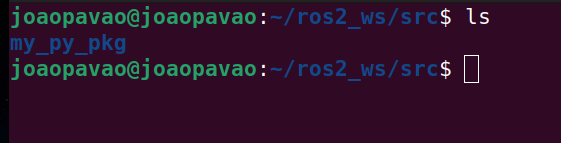






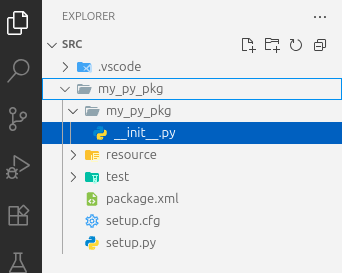


Criar pacotes

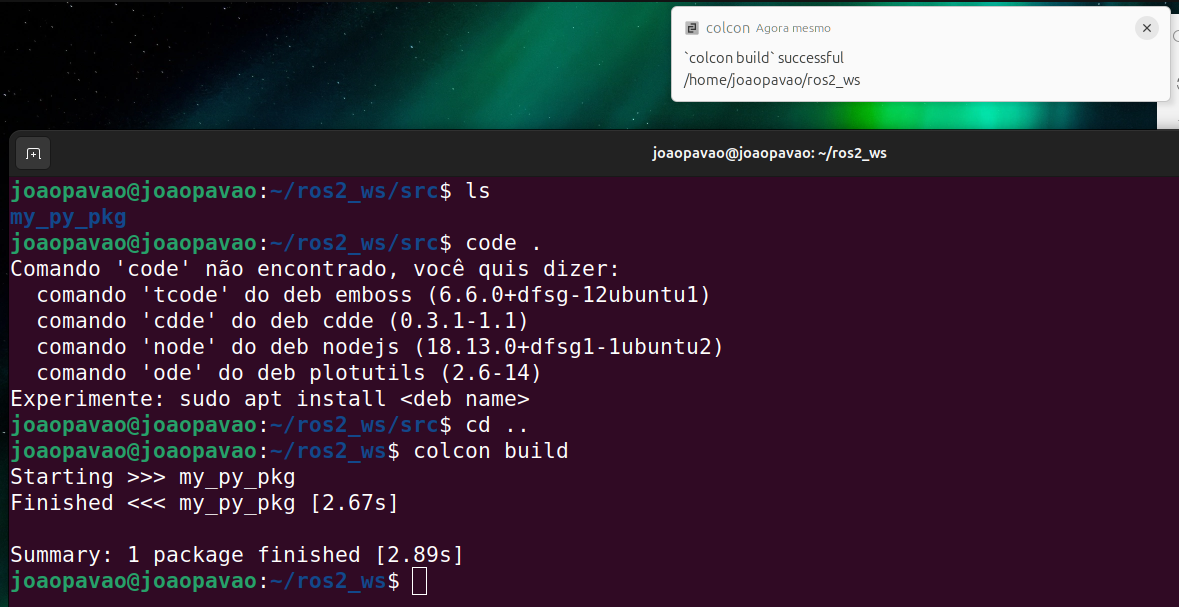
pacote novo criado

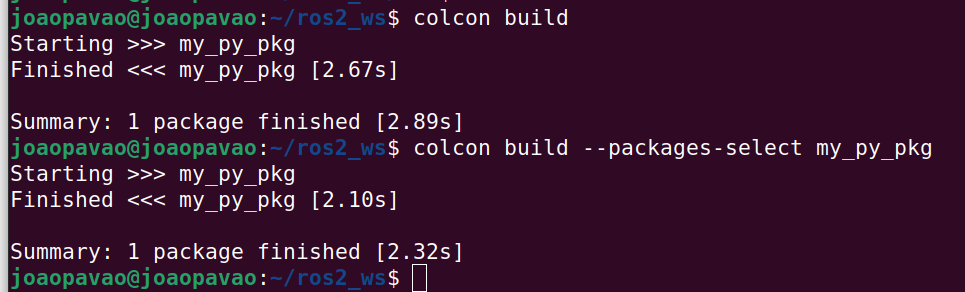
depois

abrir o pacote com o VS CODE

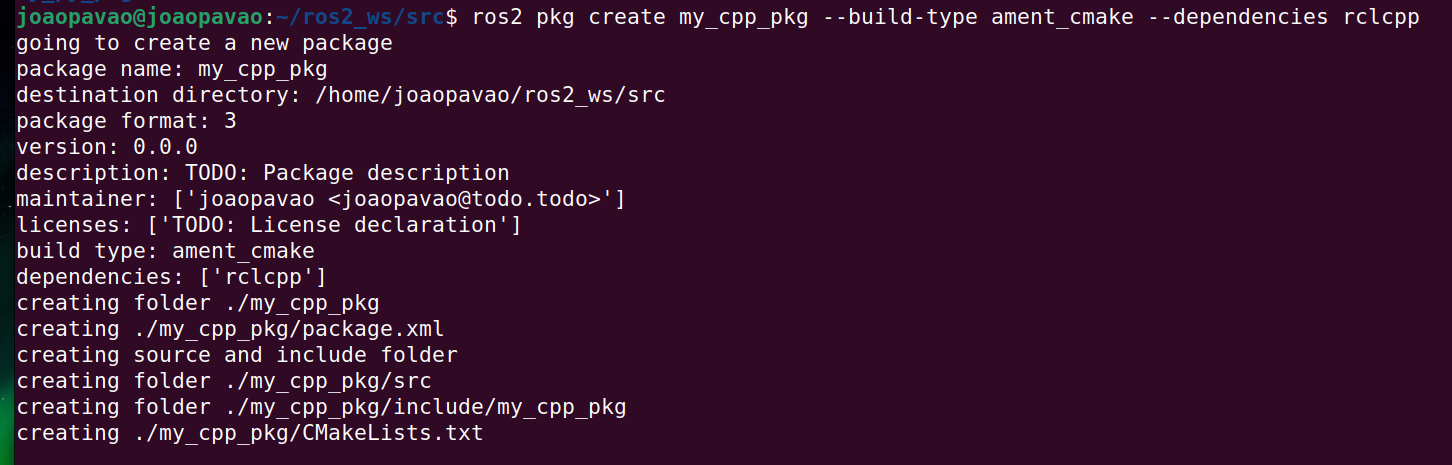


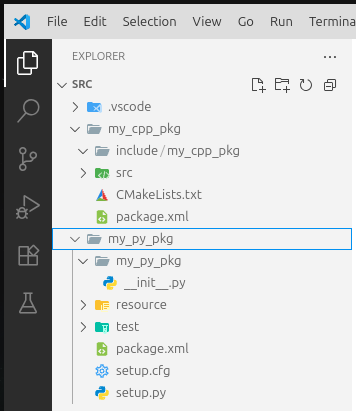
Criação de um pacote

em python

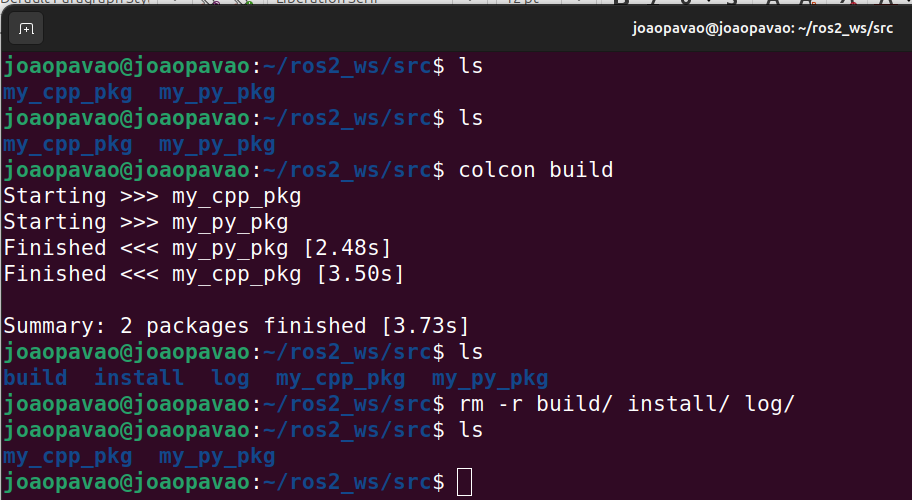


De seguida, criar um pacote em C++

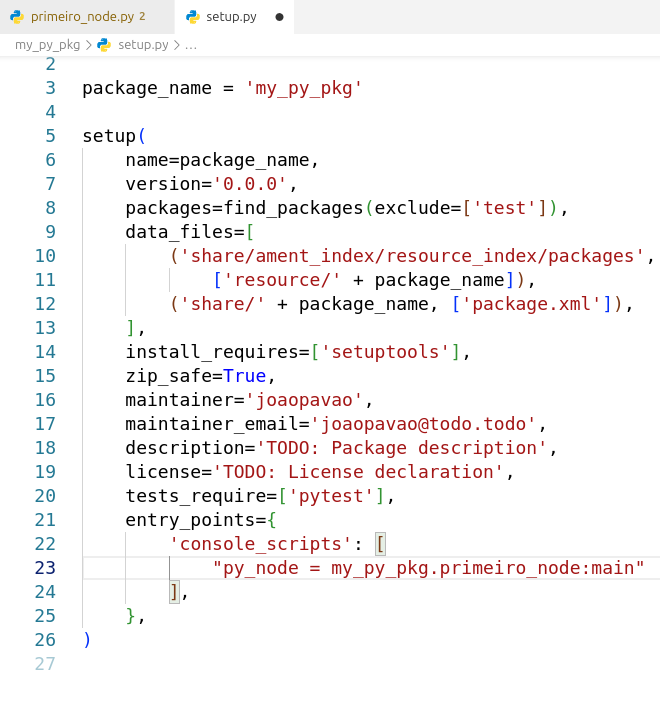




no caso de criar os pacotes no sitio errado, tem o comando para anular a ação



Criando nodes

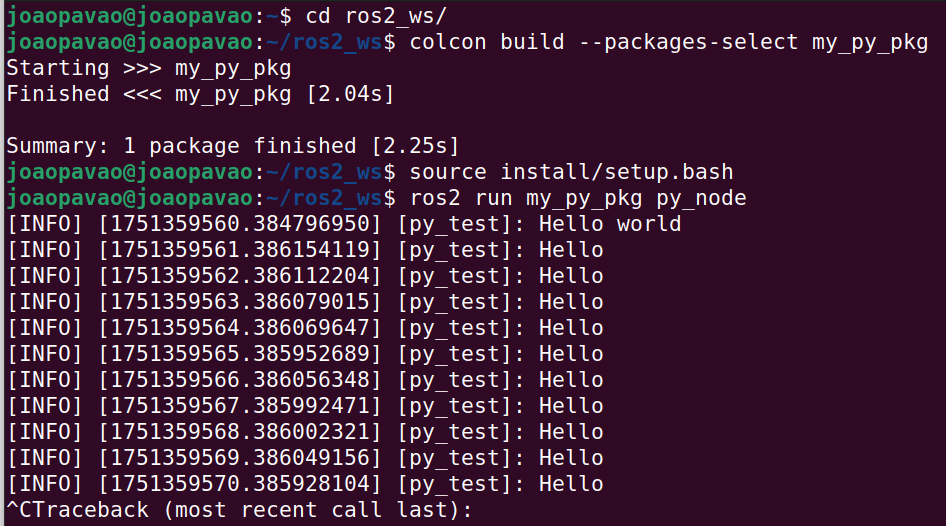
criando executaveis dentro do setup.py

esta 2ª versao faz o mesmo

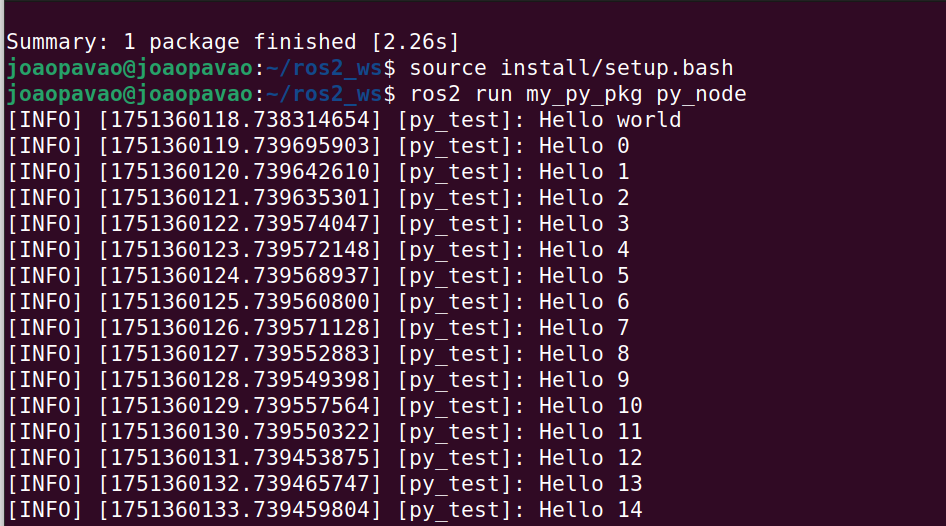


este é um modelo (template) para escrever qualquer nó (node). São poucas linhas de codigo em python

bastará alterar poucas coisas como o nome da classe (MyNode) e o nome do nó (py\_test)

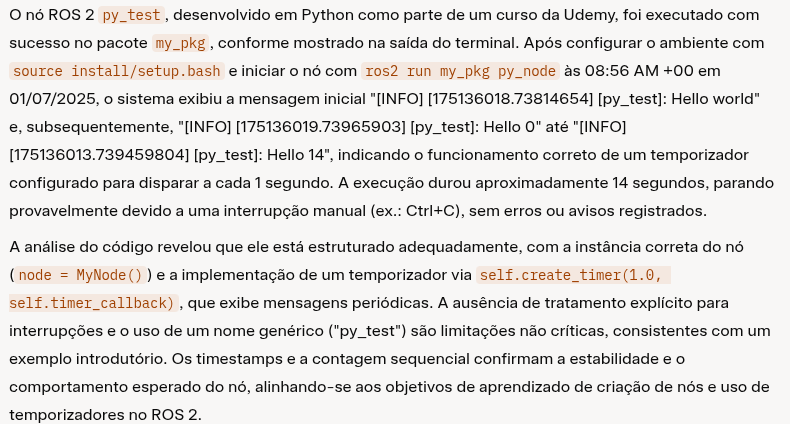


neste exemplo, temos novamente a construção do codigo fonte e a sua execução. Agora com o timer a cada segundo mostrando a mensagem “hello”, mas podia ser a temperatura de um sensor, por ex



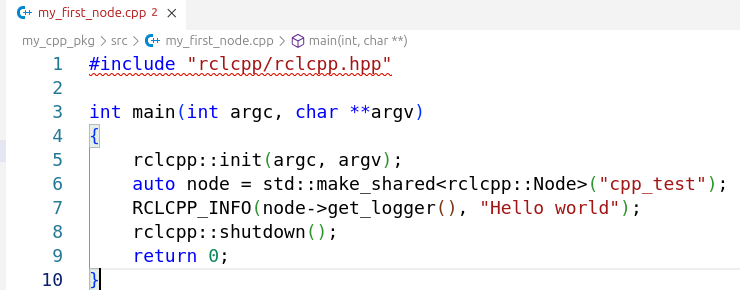
O nó ROS 2 py\_test, desenvolvido em Python como parte de um curso da Udemy, foi executado com sucesso no pacote my\_pkg, conforme mostrado na saída do terminal. Após configurar o ambiente com source install/setup.bash e iniciar o nó com ros2 run my\_pkg py\_node às 08:56 AM +00 em 01/07/2025, o sistema exibiu a mensagem inicial "[INFO] [175136018.73814654] [py\_test]: Hello world" e, subsequentemente, "[INFO] [175136019.73965903] [py\_test]: Hello 0" até "[INFO] [175136013.739459804] [py\_test]: Hello 14", indicando o funcionamento correto de um temporizador configurado para disparar a cada 1 segundo. A execução durou aproximadamente 14 segundos, parando provavelmente devido a uma interrupção manual (ex.: Ctrl+C), sem erros ou avisos registrados.

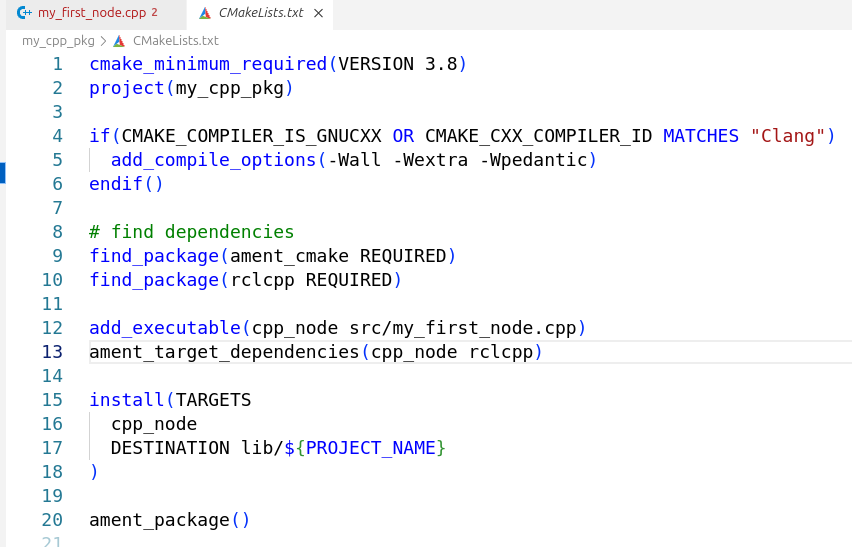
A análise do código revelou que ele está estruturado adequadamente, com a instância correta do nó (node = MyNode()) e a implementação de um temporizador via self.create\_timer(1.0, self.timer\_callback), que exibe mensagens periódicas. A ausência de tratamento explícito para interrupções e o uso de um nome genérico ("py\_test") são limitações não críticas, consistentes com um exemplo introdutório. Os timestamps e a contagem sequencial confirmam a estabilidade e o comportamento esperado do nó, alinhando-se aos objetivos de aprendizado de criação de nós e uso de temporizadores no ROS 2.



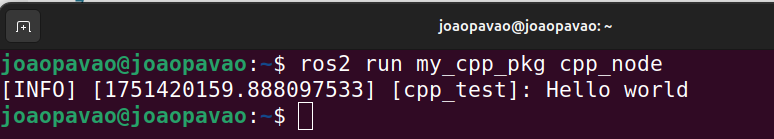
Criamos um nó com uma classe em python

De seguida, será o mesmo mas na linguagem c++

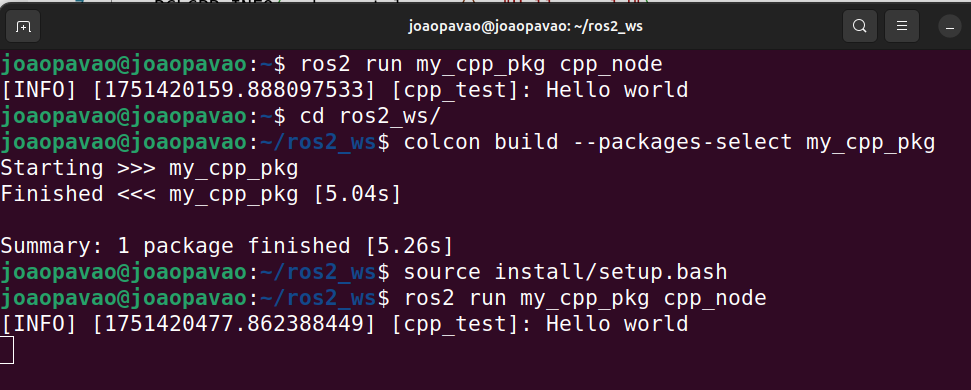




Resultado ao executar o cpp\_node

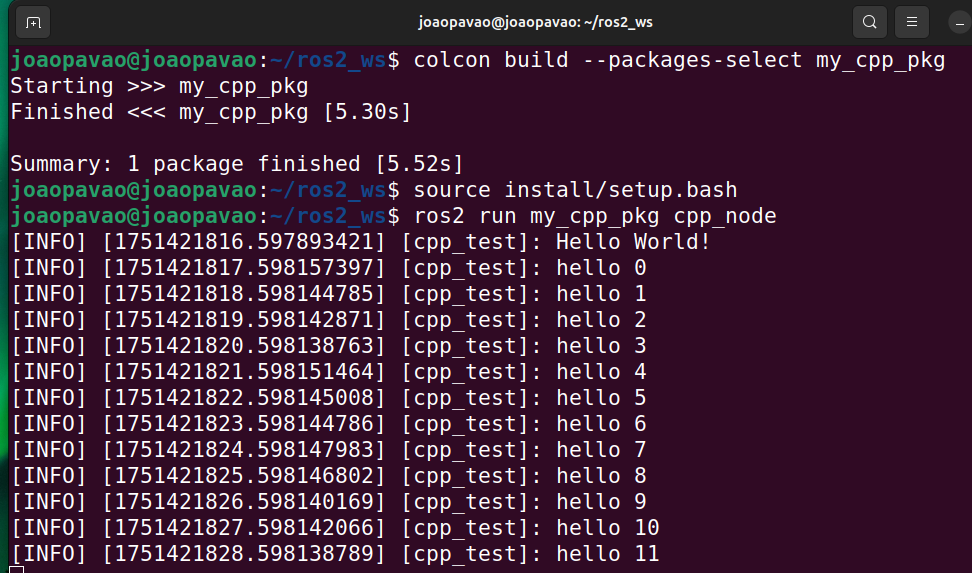


o nó só termina quando se clica em ctrl+c



Vamos agora melhorar este nó, com POO (programação arientanda a objetos)

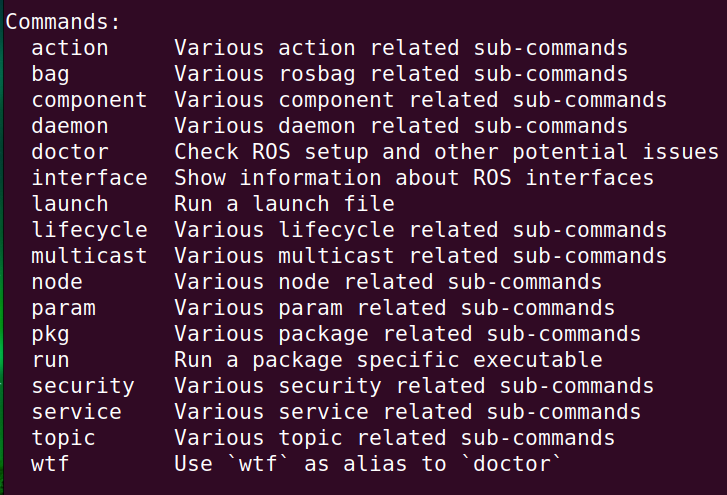




Foi criado um nó em C++, com POO, e também foi criado um temporizador com retorno de chamada

Introdução ao ROS2 Tools

As diferentes funcionalidades do ROS2

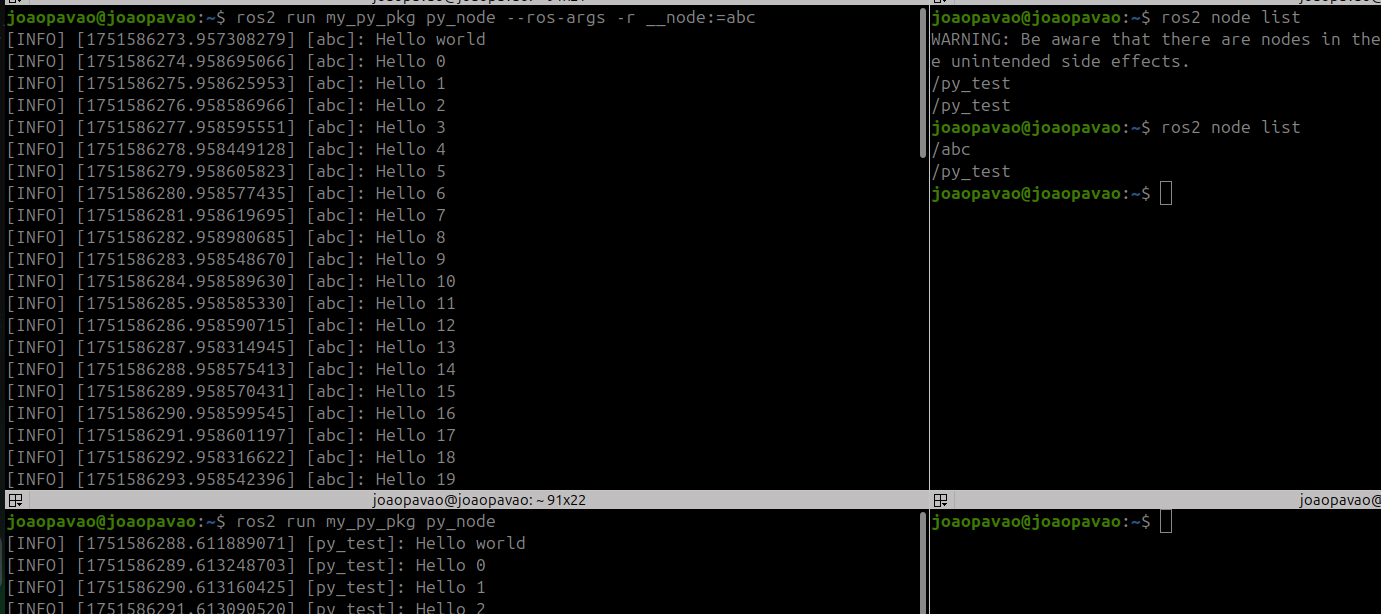


### Comandos:

* action: Vários sub-comandos relacionados a ações.
* bag: Vários sub-comandos relacionados a rosbag.
* component: Vários sub-comandos relacionados a componentes.
* daemon: Verifica configuração ROS e outros problemas potenciais.
* interface: Mostra informações sobre interfaces ROS.
* launch: Executa um arquivo de lançamento.
* lifecycle: Vários sub-comandos relacionados a ciclo de vida.
* multicast: Vários sub-comandos relacionados a multicast.
* node: Vários sub-comandos relacionados a nós.
* param: Vários sub-comandos relacionados a parâmetros.
* pkg: Vários sub-comandos relacionados a pacotes.
* run: Executa um executável específico de um pacote.
* security: Vários sub-comandos relacionados a segurança.
* service: Vários sub-comandos relacionados a serviços.
* topic: Vários sub-comandos relacionados a tópicos.
* wtf: Use 'wtf' como alias para 'doctor'.

Dia 03 de julho

Verificar vários nós ao mesmo tempo. Se em 2 terminais tivermos o mesmo nó a correr, quando se vê na lista aparecem como dois. Deve-se renomear um deles, para não haver problemas



Não precisamos alterar o nome do nó (py\_node) , pois quando o corremos, podemos mudar o seu nome.

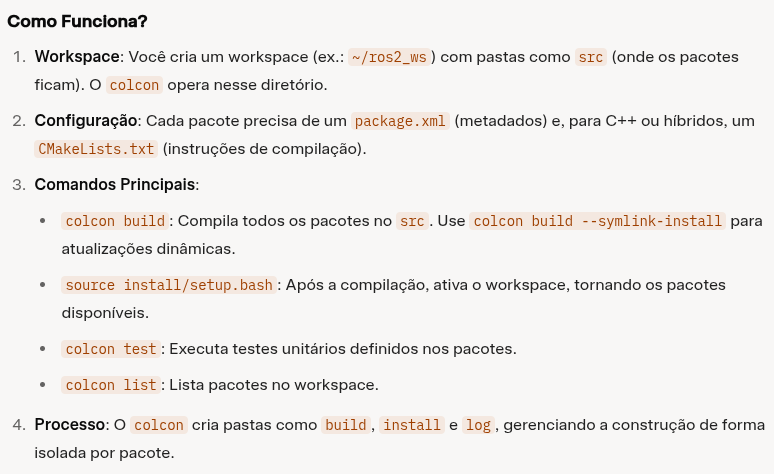
E do mesmo modo, podemos renomear outras coisas como as comunicações.

ros2 run my\_py\_pkg py\_node –ros-arg -r \_\_node:=abc

colcon no ROS2

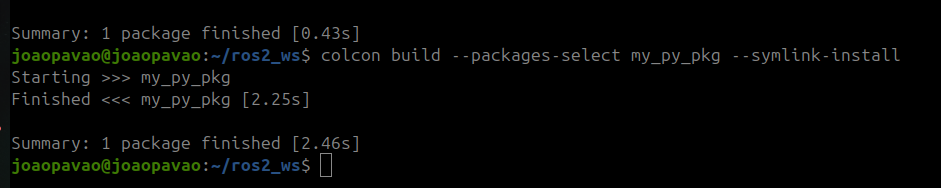
### O que é colcon?

O colcon (abreviação de collection constructor) é a ferramenta oficial de construção de projetos no ROS 2. Ele substitui o catkin usado no ROS 1 e é responsável por compilar, testar e instalar pacotes ROS 2 em um workspace. Pense nele como um "gerenciador de construção" que organiza e processa múltiplos pacotes C++ e Python, garantindo que dependências sejam resolvidas e que o código seja gerado corretamente para uso com o ROS 2.

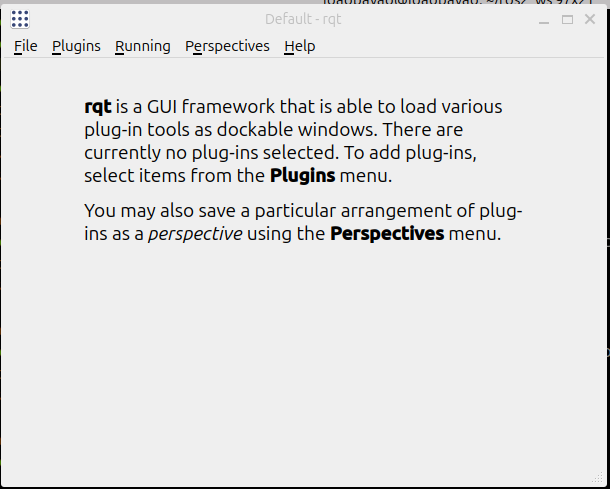


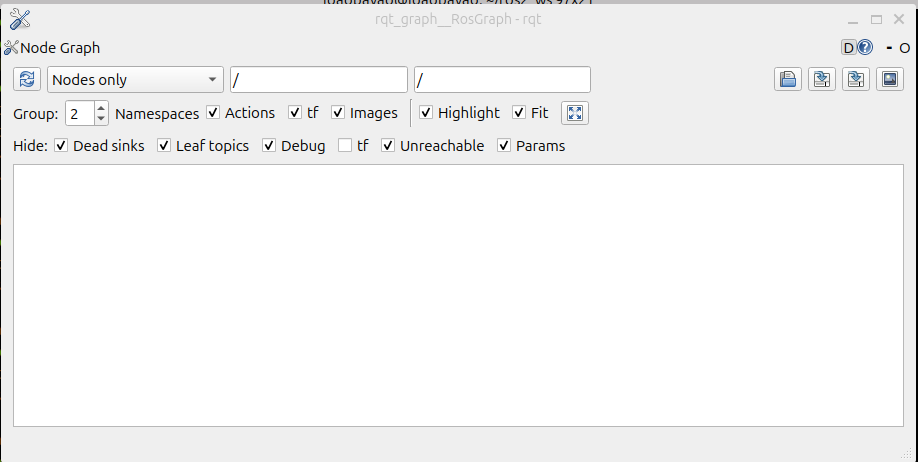


Devemos sempre compilar os pacotes dentro da pasta ros2\_ws e não em outros sítios. Como se vê na imagem, foram compilados 2 pacotes.

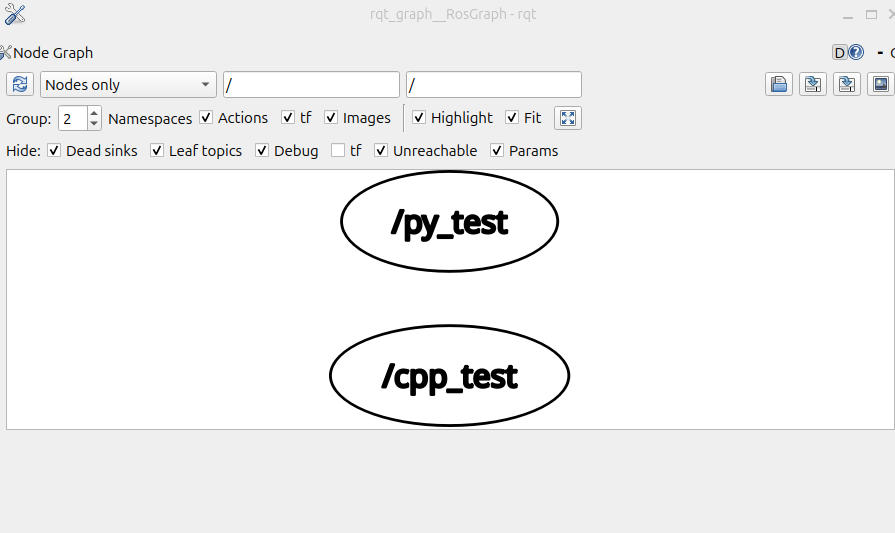


NEXT: Rqt e rqt\_graph

janela do rqt

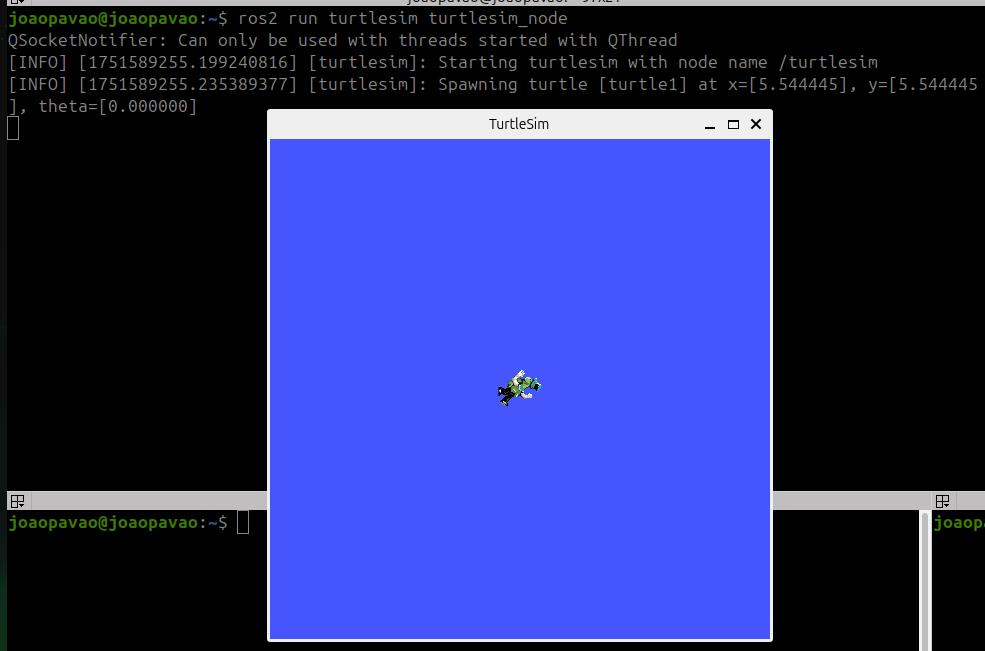


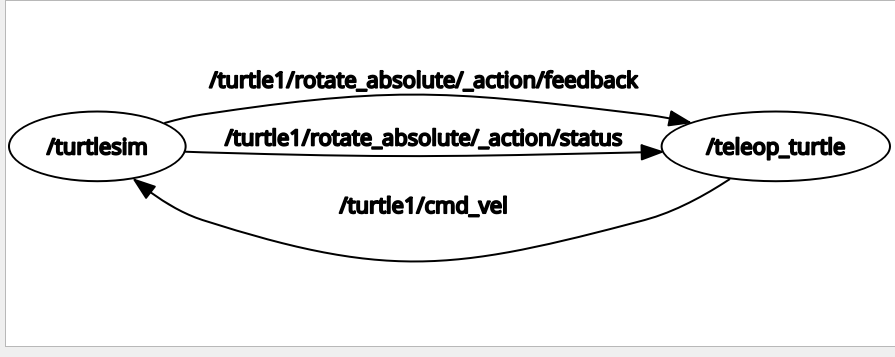
janela do rqt\_graph



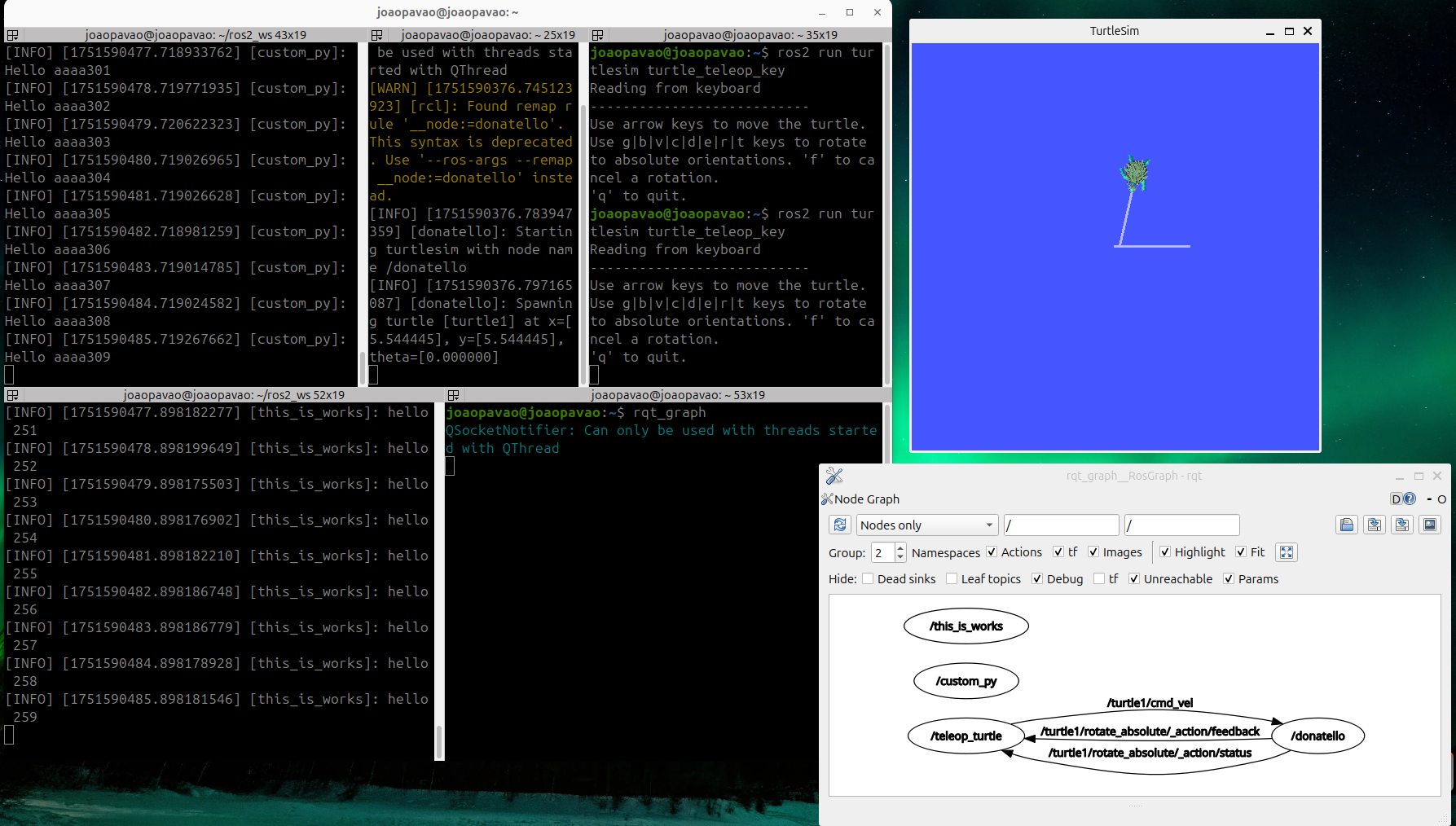
janela rqt\_graph com a visualização dos 2 nós activos.

Turtlesim





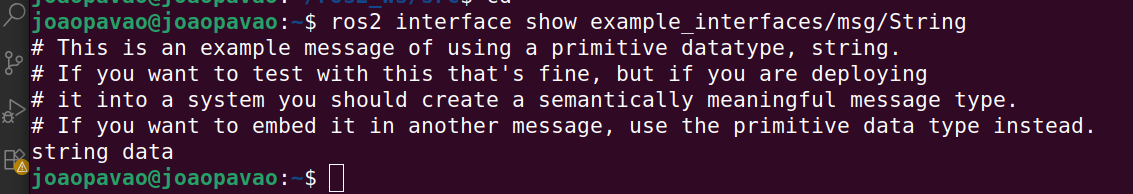
Atividade 1

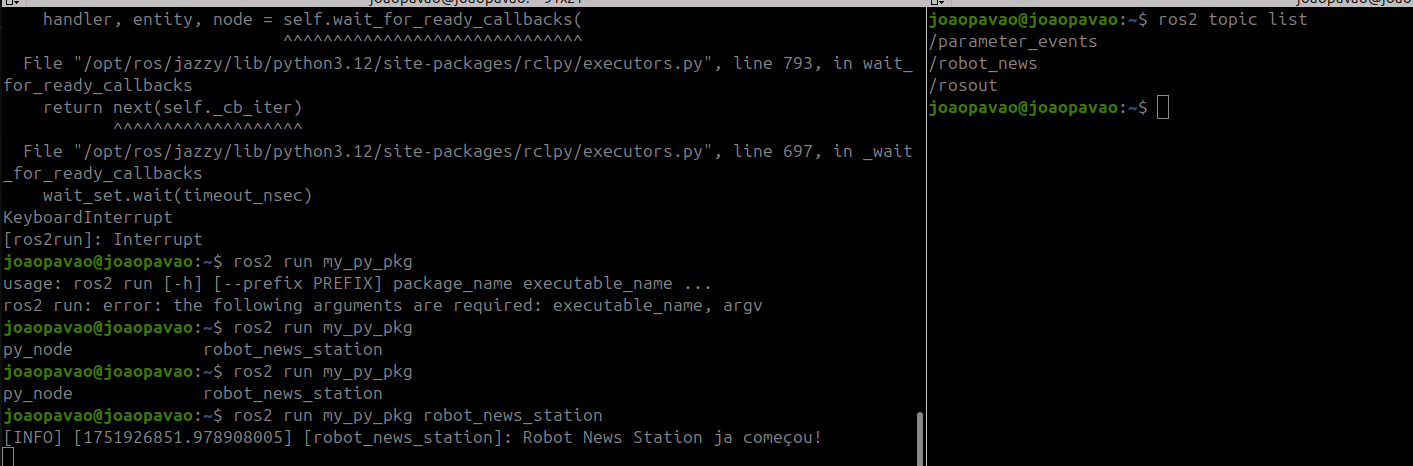


Dia 07 de julho

Assunto: Tópicos

Criar um tópico em python





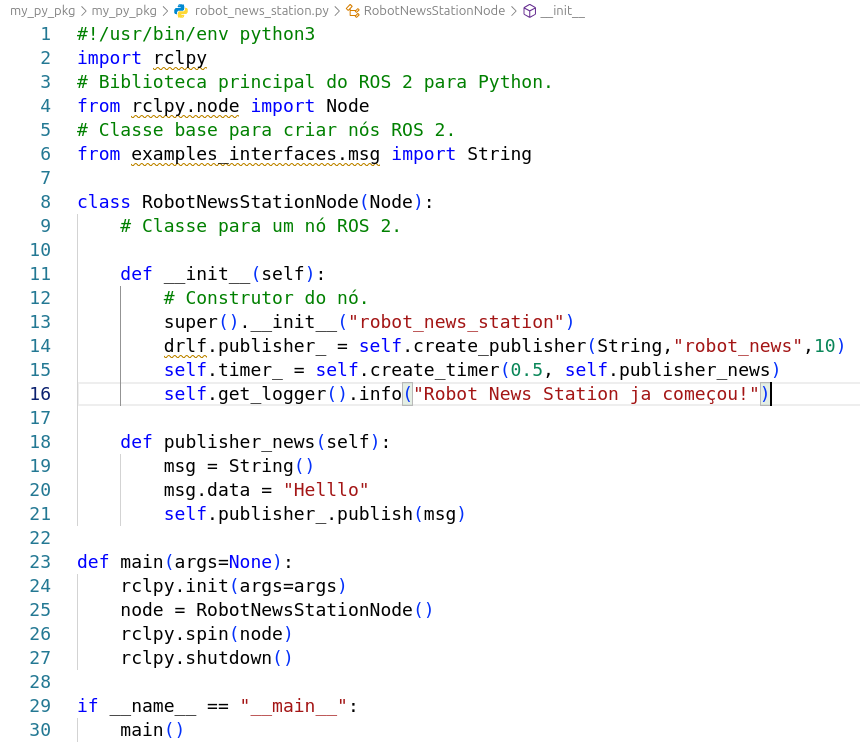
ros2 topic list

Criação do primeiro publisher em python no ROS2

O código que você forneceu é para criar um publisher em ROS 2 (Robot Operating System 2), escrito em Python. Um publisher é um componente que envia mensagens para um tópico, que pode ser "ouvido" por outros nós no sistema ROS 2. Vou explicar de forma mais detalhada e pedagógica, como se estivéssemos em uma aula de ROS 2, para que você entenda cada parte e seu propósito.

### O que é um Publisher no ROS 2?

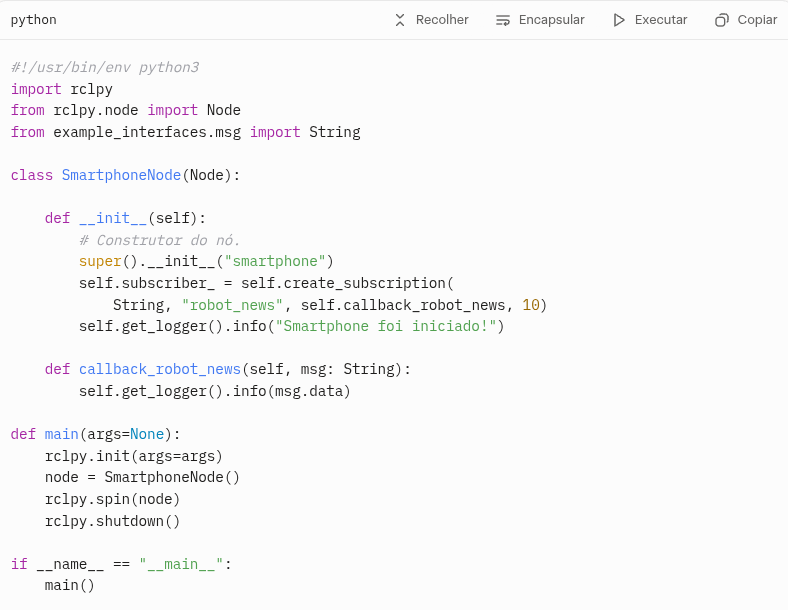
Imagine o ROS 2 como uma rede de comunicação entre diferentes "robôs" ou processos (nós). Um publisher é como uma estação de rádio que transmite informações (mensagens) através de um canal chamado tópico. Outros nós podem se conectar a esse tópico como "ouvintes" (assinantes) para receber essas mensagens. No seu caso, o código cria um nó que publica mensagens de texto no tópico robot\_news.



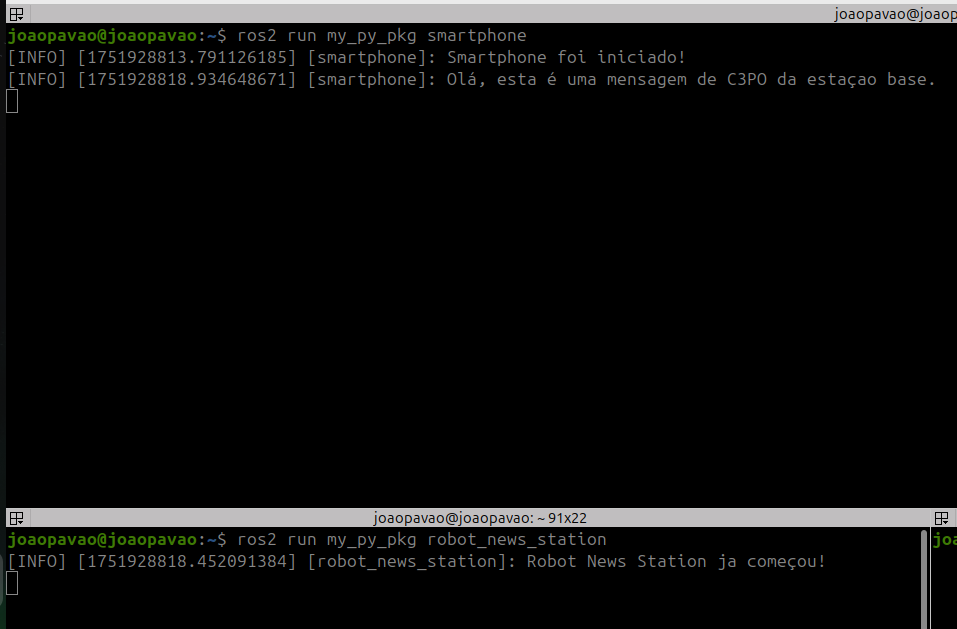
NEXT, criaçao de um Subscribe

### O que é um Assinante no ROS 2?

Um assinante é um nó que "escuta" um tópico e recebe mensagens publicadas por um publisher. No nosso caso, vamos criar um assinante para receber as mensagens "Olá" do tópico robot\_news.

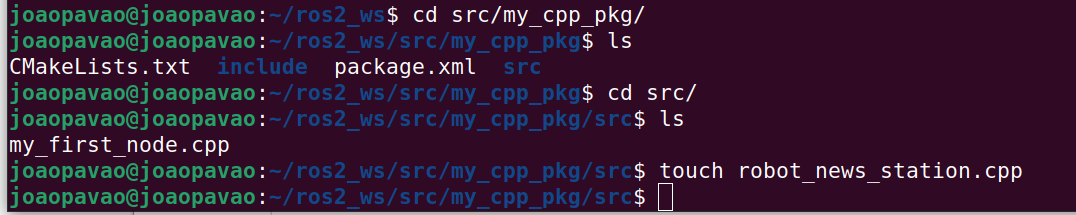


Exemplo de um assinante (codigo em python)



Terminal de cima tem o assinante (o ouvinte) e o terminal de baixo tem o subscribe (o emissor)

VAMOS agora ao C++





Vamos analisar e comentar o código fornecido para o nó RoboNewsStationNode, um publisher ROS 2 em C++, com base na imagem que você compartilhou. Vou registrar cada parte, explicar seu propósito e adicionar comentários detalhados em português, como um professor de ROS 2.

### Código Fornecido

cpp

RecolherEncapsularExecutar

Copiar

### Comentários e Registro

#### 1. Inclusões (Headers)

* #include "rclcpp/rclcpp.hpp": Biblioteca essencial do ROS 2 em C++, que fornece as ferramentas para criar nós, publicadores, assinantes, etc.
* #include "example\_interfaces/msg/string.hpp": Inclui a definição da mensagem String, que tem um campo data do tipo string, usada aqui para as notícias.

#### 2. Namespace chrono\_literals

* using namespace std::chrono\_literals;: Habilita o uso de literais como 0.5s para definir durações. Isso requer que o compilador esteja configurado para C++14 ou superior (ex.: -std=c++14).

#### 3. Classe RoboNewsStationNode

* Construtor RoboNewsStationNode():
  + Node("robot\_news\_station"): Inicializa o nó com o nome robot\_news\_station, que o identifica no sistema ROS 2.
  + robot\_name\_("R2D2"): Define o nome do robô como "R2D2", que será incluído nas mensagens.
  + publisher\_ = ...: Cria um publicador para o tópico robot\_news com mensagens String e uma fila de 10 (para lidar com atrasos de assinantes).
  + timer\_ = ...: Configura um timer que executa publishNews a cada 0.5 segundos, usando std::bind para associar o método à instância atual.
  + RCLCPP\_INFO(...): Registra uma mensagem no log para indicar que o nó foi iniciado, útil para depuração.
* Método publishNews():
  + auto msg = ...: Cria uma mensagem String.
  + msg.data = ...: Define o conteúdo da mensagem como "Hello, aqui é " + robot\_name\_ + " from the robot news station". A correção removeu o parêntese extra e corrigiu "startion" para "station".
  + publisher\_->publish(msg): Envia a mensagem para o tópico robot\_news.
* Membros Privados:
  + std::string robot\_name\_: Armazena o nome do robô.
  + rclcpp::Publisher<...>::SharedPtr publisher\_: Ponteiro compartilhado para gerenciar o publicador, corrigido de SharePtr para SharedPtr.
  + rclcpp::TimerBase::SharedPtr timer\_: Ponteiro compartilhado para gerenciar o timer, também corrigido.

#### 4. Função main

* rclcpp::init(argc, argv): Configura o ambiente ROS 2 com os argumentos da linha de comando.
* auto node = ...: Cria uma instância compartilhada do nó usando std::make\_shared.
* rclcpp::spin(node): Mantém o nó ativo, processando callbacks (como o timer).
* rclcpp::shutdown(): Finaliza o ambiente ROS 2.
* return 0: Indica que o programa terminou com sucesso.

### Observações e Correções Aplicadas

* A imagem mostra anotações em verde indicando correções:
  + A string foi corrigida para "Hello, aqui é " + robot\_name\_ + " from the robot news station", removendo o parêntese extra e corrigindo "startion".
  + Os tipos foram ajustados para SharedPtr em vez de SharePtr.
* O uso de chrono\_literals com 0.5s está correto, desde que o compilador suporte C++14 ou superior.

### Como Testar?

1. Configuração:
   * Salve o código em robot\_news\_station.cpp dentro de um pacote ROS 2.
   * Adicione example\_interfaces como dependência no package.xml:

xml

RecolherEncapsular

Copiar

* + Atualize o CMakeLists.txt:

cmake

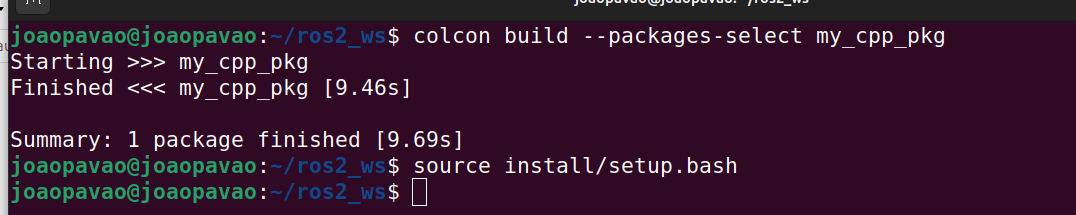
RecolherEncapsular

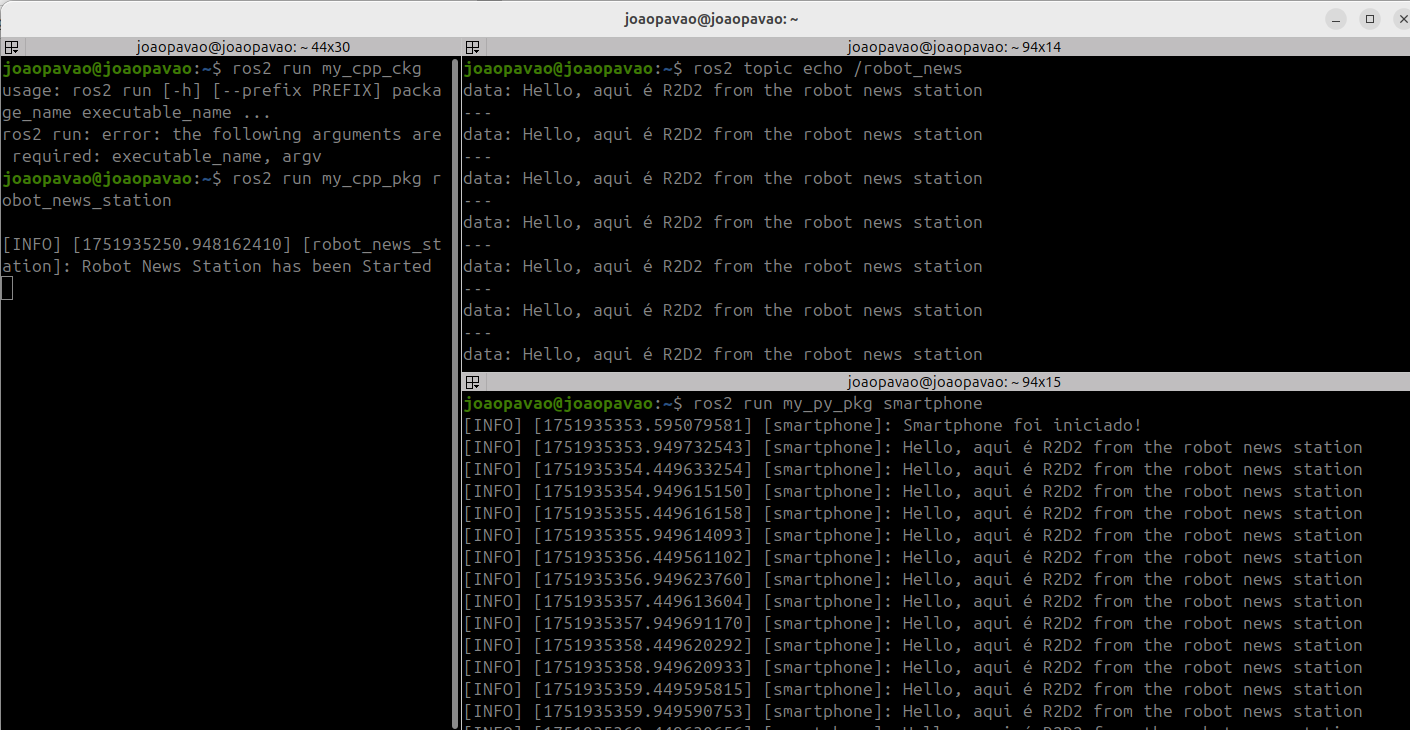
Copiar

* + Compile com colcon build e fonte o ambiente com source install/setup.bash.

1. Execução:
   * Rode o nó: ros2 run seu\_pacote robot\_news\_station.
   * Veja as mensagens: ros2 topic echo /robot\_news.
2. Saída Esperada:
   * "Robot News Station has been Started" no log ao iniciar.
   * Mensagens como "Hello, aqui é R2D2 from the robot news station" a cada 0.5 segundos.

Correr em c++ e executar

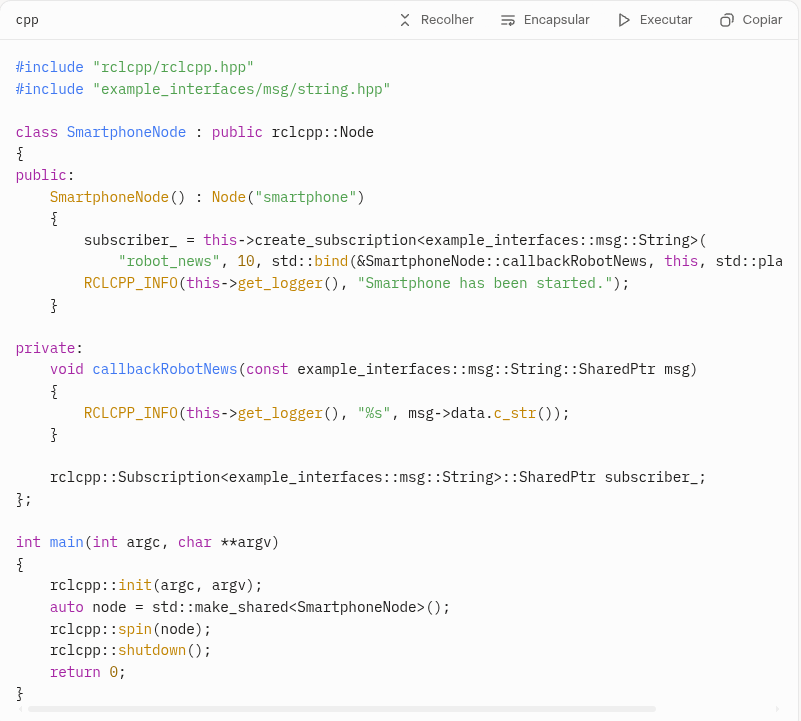




Este exemplo mostra que a criação de nos é independente da linguagem em ROS2

Os nos comunicam usando tópicos

Agora vamos à criação de um c++ subscrible

Olá! Vamos analisar o código que você forneceu para o nó `SmartphoneNode`, um assinante ROS 2 escrito em C++. O código já parece estar bastante corrigido com base nas sugestões anteriores, então vou revisá-lo, confirmar sua funcionalidade e adicionar explicações detalhadas como um professor de ROS 2.

