Ferramentas a usar no Ubuntu 24.04

Terminator terminal

sudo apt install terminator

[igniteflow/terminator-keyboard-shortcuts: Terminator keyboard shortcuts](https://github.com/igniteflow/terminator-keyboard-shortcuts)

VS Code

Apos a instalação, ir para as extensões e procurar por ros

INSTALAÇAO do ROS2 no UBUNTU

Seguir o tutorial oficial

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

(27 de maio de 2025)

2ª tentativa de instalar o ROS 2

Correu bem ate agora

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Os conteúdos gerados por IA podem estar incorretos.

Conclusão da Secção

Nesta secção instalaste o ROS 2 no Ubuntu e configuraste o teu ambiente.

Foi um processo com bastantes etapas, mas conseguiste — parabéns!

A instalação pode não ser a parte mais divertida, mas é uma etapa necessária.

Recapitulação rápida sobre que versão do ROS 2 deves usar:

Utiliza sempre a última versão LTS do ROS 2. Neste momento, essa versão é o ROS 2 Jazzy (lançado em 2024).

Nota: se já começaste com o ROS 2 Humble, não há problema — também é uma versão LTS e terá suporte até meados de 2027.

Por vezes, pode ser necessário esperar um pouco mais (6 meses a 1 ano) antes de usares a versão LTS mais recente, porque alguns pacotes ainda não foram adaptados.

Neste caso, continua a usar a penúltima LTS disponível.

Sobre as ferramentas utilizadas (não específicas do ROS):

Editor de texto/IDE: VS Code e gedit.

Sente-te à vontade para usar qualquer outro editor com o qual te sintas mais confortável — o mais importante é manter a consistência.

Terminais múltiplos no ecrã: recomenda-se o Terminator (no Ubuntu).

Se não quiseres usar o Terminator, podes simplesmente abrir vários terminais e usar separadores.

Outro programa conhecido e semelhante ao Terminator é o tmux.

Registo do dia 31 de janeiro

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.

At the end of this section you’ll be able to write and run your own ROS 2 programs (nodes).

I will explain what is a ROS 2 node, and then you’ll directly create one. To setup a new node you will:

Create a ROS 2 workspace in your home directory.

Create a Package inside this workspace.

Write a node inside the package, set it up, compile it and run it.

As you can see lectures are duplicated for Python and Cpp. I start with Python, then do the same thing with Cpp, so you can learn with both languages, and see the similarities and differences between a Python ROS 2 program and a Cpp ROS 2 program.

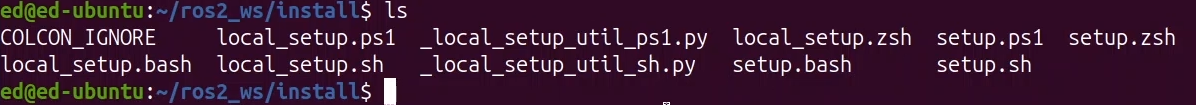
At the end of the section you’ll be able to download the complete workspace with all the code for both languages.

Even if you want to learn ROS 2 with only one language, I encourage you to at least watch the lectures for the other language. The repetition - in a different way - of explanations for a new concept is a great way to retain more information, and get a better understanding.

And now, go to the next lecture to create your ROS 2 workspace!

**Criação de um workspace**

(espaço de trabalho)



**📁 Ficheiros encontrados**

**🟩 setup.bash, setup.sh, setup.zsh, setup.ps1**

* **Função:** inicializam o ambiente ROS 2 para o workspace (ros2\_ws), incluindo variáveis de ambiente como ROS\_PACKAGE\_PATH, AMENT\_PREFIX\_PATH, etc.
* **Diferença entre eles:**
  + setup.bash → para quem usa **bash** (Ubuntu padrão).
  + setup.sh → genérico, usado por scripts ou shells não especificadas.
  + setup.zsh → para quem usa **zsh**.
  + setup.ps1 → para **PowerShell** (no Windows).

**🟦 local\_setup.bash, local\_setup.sh, local\_setup.zsh, local\_setup.ps1**

* **Função:** similar aos anteriores, mas afetam **apenas o workspace local** (ros2\_ws) — não os workspaces parent (ex: /opt/ros/jazzy/).
* São úteis em **workspaces sobrepostos** (overlay workspaces).
* Exemplo:

bash

CopiarEditar

source install/local\_setup.bash

**🐍 \_local\_setup\_util\_sh.py, \_local\_setup\_util\_ps1.py**

* **Scripts auxiliares em Python** que definem e preparam variáveis dinamicamente com base no ambiente do sistema e no shell utilizado.
* São chamados internamente pelos setup.\* e local\_setup.\*.

**🧱 COLCON\_IGNORE**

* Este ficheiro serve para impedir o colcon de interpretar a pasta como válida para compilar pacotes.
* **Se estiver presente**, o conteúdo da pasta pode ser **ignorado durante builds ou sourcing** — normalmente é usado para debugging ou desativar momentaneamente parte do workspace.

ROS 2 – PACKAGES

Os pacotes permitem estruturar o código em componentes reutilizáveis, facilitando a organização e a manutenção.

Para criar pacotes,

ros2 pkg create



Depois de criar o pacote, ir para a pasta e abrir com o VS Code

Uma imagem com Tipo de letra, Gráficos, logótipo, texto

Os conteúdos gerados por IA podem estar incorretos.

❓ **O que é um Nó (Node) no ROS 2?**

Um **nó** em ROS 2 é a unidade fundamental de execução. Podes pensar nele como um pequeno programa que executa uma tarefa específica dentro de um sistema robótico. Em termos simples:

**🔧 Definição:**

Um **nó ROS 2** é um processo que realiza uma função específica, como ler sensores, controlar motores, processar imagens ou tomar decisões com base em dados.

**🧠 Exemplo prático:**

Imagina um robô com:

* um nó que lê dados do sensor ultrassónico (sensor\_node);
* outro que comanda os motores (motor\_controller\_node);
* outro que decide o que fazer com base na distância medida (logic\_node).

Cada um destes é um **nó separado**, mas todos comunicam entre si usando **tópicos**, **serviços** ou **ações**.

**🛠️ Características principais:**

* São criados dentro de **pacotes** (ros2 pkg create).
* Podem ser escritos em **Python** (rclpy) ou **C++** (rclcpp).
* Comunicam usando **publishers/subscribers** (tópicos) ou **clients/servers** (serviços).
* São geridos pelo **sistema de execução do ROS 2**.

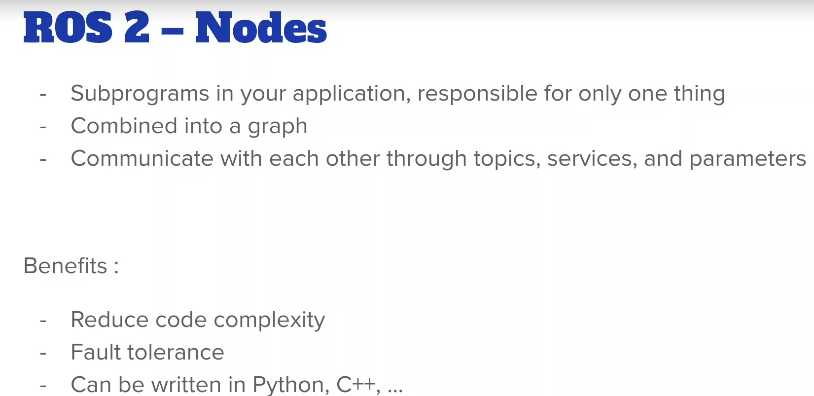
**💡 Analogia simples:**

Pensa num nó como uma **aplicação individual** dentro de um sistema operativo distribuído (o ROS 2). Todos cooperam para fazer o robô funcionar como um todo.

Uma imagem com texto, captura de ecrã, Tipo de letra, Retângulo

Os conteúdos gerados por IA podem estar incorretos. Uma imagem com texto, captura de ecrã, diagrama, file

Os conteúdos gerados por IA podem estar incorretos.



**🧠 ROS 2 – Nodes (Nós)**

**O que são?**

* **Subprogramas** dentro da tua aplicação robótica.
* Cada nó é responsável por **apenas uma tarefa específica** (ex: ler sensores, processar imagem, enviar comandos).
* Todos os nós são **ligados entre si formando um grafo** de comunicação.
* Comunicam através de:
  + **Tópicos** (dados em fluxo contínuo),
  + **Serviços** (pedidos e respostas),
  + **Parâmetros** (configurações dinâmicas).

**✅ Vantagens dos nós em ROS 2:**

* ✅ **Reduzem a complexidade do código**: cada tarefa é isolada num nó independente.
* ✅ **Tolerância a falhas**: se um nó falhar, os outros podem continuar.
* ✅ **Flexibilidade de linguagem**: podes programar em **Python, C++**, entre outras.

**📌 Em resumo, para os alunos:**

Um nó em ROS 2 é um pequeno programa especializado, que comunica com outros nós para que o robô funcione como um sistema inteligente, modular e robusto.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.

Uma imagem com texto, captura de ecrã, documento, número

Os conteúdos gerados por IA podem estar incorretos.

**✅ Observações úteis:**

* 📡 Este nó **não executa continuamente** (como seria comum com rclpy.spin(node)), pois está pensado apenas para mostrar uma mensagem de "Hello world" e encerrar.
* Uma imagem com texto, captura de ecrã, software, Software de multimédia

  Os conteúdos gerados por IA podem estar incorretos.🧱 Serve como **modelo base** para testes, especialmente útil para verificar se o ROS 2 está corretamente instalado e a funcionar.

Este é o **exemplo clássico de um nó ROS 2 com spin()**, e agora sim o nó fica ativo até o utilizador o interromper manualmente (por exemplo, com Ctrl+C no terminal).

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.

**🧠 Explicação extra: rclpy.spin(node)**

* Funciona como um **loop de eventos**, muito parecido com o while True: nos programas clássicos.
* É essencial para manter o nó **vivo e reativo** — sem isto, o nó terminaria imediatamente após imprimir a mensagem.
* Com spin(), o ROS 2 pode responder a **subscripções, publicações, timers ou serviços** que este nó implemente.
* **Ctrl+C** no terminal envia um **sinal de interrupção (SIGINT)** que é captado pelo rclpy.spin(), terminando o nó de forma segura.

De seguida vamos instalar o no (como executável)

Aqui tem um print de um node executável.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.

Vamos adicionar mais uma funcionalidade, por exemplo, imprimir algo a cada um segundo

Para isso vamos adicionar um timer

Sessao 19 – construir um no em C++

Resumo

**Aqui está um modelo que podes usar para todos os teus nós. Certifica-te de que compreendes o que cada linha faz, revendo as aulas anteriores se necessário.**

A partir de agora, vou começar os nós a partir deste modelo para não perdermos tempo a escrever sempre o mesmo código no início de cada vídeo.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Os conteúdos gerados por IA podem estar incorretos.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.

Capitulo 22



RCL – é a biblioteca cliente ROS2 mais baixa que se pode usar e é a ponte com o middleware ROS2

RCLCPP – é uma biblioteca cliente C++ que fornece as funcionalidades de RCL

RCLpy – é a mesma coisa, mas para python

**✅ Conclusão da Secção**

Nesta secção aprendeste como criar os teus primeiros nós ROS 2, tanto em Python como em C++. Esta é já uma base muito sólida. Se estás a ler isto, então já fizeste **muito progresso**!

**🔁 Recapitulando rapidamente:**

* Os **nós** são subprogramas dentro da tua aplicação.
* Cada nó é responsável **apenas por uma tarefa específica**.
* Os nós comunicam entre si através de **tópicos**, **serviços** e **parâmetros**.

Se olhares para as próximas secções, vais ver que é exatamente sobre isso que vamos trabalhar a seguir.

Aqui está um exemplo de uma arquitetura típica para uma aplicação ROS 2, onde podes ver pacotes e nós:

Uma imagem com texto, diagrama, file, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.

**🛠️ Antes de criares o teu primeiro nó, deves:**

1. Criar uma **área de trabalho ROS 2** (workspace) e fazer o **source** dessa área.
2. Criar um **pacote** (em Python ou C++).
3. Depois, escreves o teu **nó** utilizando a biblioteca cliente ROS 2 apropriada:
   * rclpy para **Python**
   * rclcpp para **C++**

Ambas as bibliotecas fornecem as **mesmas funcionalidades principais**, apenas adaptadas à linguagem escolhida.

Uma imagem com texto, diagrama, file, captura de ecrã

Os conteúdos gerados por IA podem estar incorretos.

**🧪 Após escreveres o nó:**

* **Compilas** o nó (apenas no caso de C++) e,
* **Fazes novamente source** do ambiente para poderes utilizá-lo.

Os nós são:

* **Compilados** (C++),
* e **instalados** (tanto em Python como em C++),

dentro da pasta install/ da tua área de trabalho ROS 2.

Podes executá-los diretamente a partir dessa pasta, **ou** usando o comando:

bash

CopiarEditar

ros2 run <nome\_do\_pacote> <nome\_do\_executável>

**🤔 Não compreendes tudo já? Não faz mal.**

É completamente normal.  
Os conceitos do ROS podem ser difíceis de perceber à primeira.  
Continua a avançar no curso e volta a esta secção mais tarde — tudo fará sentido com o tempo, e terás uma visão global sólida.

**🔧 Na próxima secção:**

Vais praticar mais com nós e descobrir algumas **ferramentas úteis do ROS 2**.  
Ficar confortável com essas ferramentas vai ajudar-te bastante quando chegares aos tópicos como **Topics**, **Services**, etc.

**📦 Código desta secção:**

Podes **descarregar o código completo desta secção** (está disponível na aula "Section Conclusion", com o nome code\_end\_section\_3.zip).

Esse ficheiro inclui **tudo o que está dentro da pasta ~/ros2\_ws/src/** da tua área de trabalho ROS 2.

DIA 03 DE julho

**🔧 O que é o colcon?**

**colcon** significa *COmplex Launcher for CONstruction*, e é usado para:

* **Compilar** pacotes ROS 2,
* **Gerir dependências entre pacotes**,
* **Criar workspaces organizados** com diretórios como build/, install/, log/.

**🧱 Estrutura típica do ROS 2 workspace com colcon:**

Uma imagem com texto, recibo, Tipo de letra, branco

Os conteúdos gerados por IA podem estar incorretos.

**Dito por si:**

Activity 01 - ROS 2 Nodes and Tools [Note] in this course I will often give you some “activities” for you to practice the subjects you’ve learned. This will start with an explanation of the challenge (this lesson) and then the video solution is in the next lesson. I encourage you to try to do the activities on your own and give it some time - that’s the best way you can make progress. If needed, come back to the previous lessons for a quick refresher. Of course, no worries if you can’t manage to solve the challenge the first time. If stuck for too long, watch the solution, and give it another try in a few days. [/Note] Here’s a quick challenge for you. I have run some nodes with different names, and here is what I see with rqt\_graph. (right click > "view image" to increase the size of the image in your browser)

**Dito pelo ChatGPT:**

Claro! Aqui está a tradução para português europeu:

**🧩 Atividade 01 – Nós ROS 2 e Ferramentas**

💡 **Nota:** Neste curso, vou frequentemente propor-te algumas *atividades* para praticares os conteúdos que aprendeste.  
A atividade começa com a explicação do desafio (nesta lição), e a solução em vídeo está na próxima lição.  
Recomendo vivamente que tentes resolver as atividades por ti próprio e dediques algum tempo a isso – essa é a melhor forma de evoluíres.  
Se necessário, volta às lições anteriores para rever rapidamente os conceitos.  
E claro, não te preocupes se não conseguires resolver à primeira.  
Se ficares bloqueado muito tempo, vê a solução e volta a tentar passados uns dias.

**🔍 Desafio rápido:**

Corri alguns nós com nomes diferentes e aqui está o que vejo no rqt\_graph.

*(Clica com o botão direito > “ver imagem” para aumentares o tamanho no teu navegador)*

Uma imagem com texto, captura de ecrã, file

Os conteúdos gerados por IA podem estar incorretos.

**✅ Conclusão da Secção**

Nesta secção, descobriste **duas ferramentas importantes** que vais usar em qualquer aplicação ROS 2 que venhas a desenvolver:

* A **CLI do ROS 2** (*Command Line Interface*),
* E o **rqt** com todos os seus plugins.

Ao longo do curso, continuarás a praticar com estas ferramentas e a descobrir outras novas.

Também exploraste o pacote **Turtlesim**, que contém um robô 2D simplificado — será muito útil para aplicares o que aprenderes ao longo do curso.

**📌 Com isto, juntando a secção anterior sobre nós e as atividades desta, já tens uma boa base para:**

* Criar um **nó personalizado** dentro do teu próprio pacote (em Python e C++);
* **Compilar e executar** os teus nós;
* **Lançar e depurar** os nós com diferentes ferramentas;
* **Executar nós existentes** de outros pacotes.

**⏭️ Próximo passo lógico:**

Ver **como os nós comunicam entre si**.  
E… é precisamente isso que vais aprender na próxima secção: **ROS 2 Topics**.

**COMUNICAÇAO entre NÓS**

**📘 Introdução**

Chegou agora o momento de explorares as funcionalidades de **comunicação no ROS**.

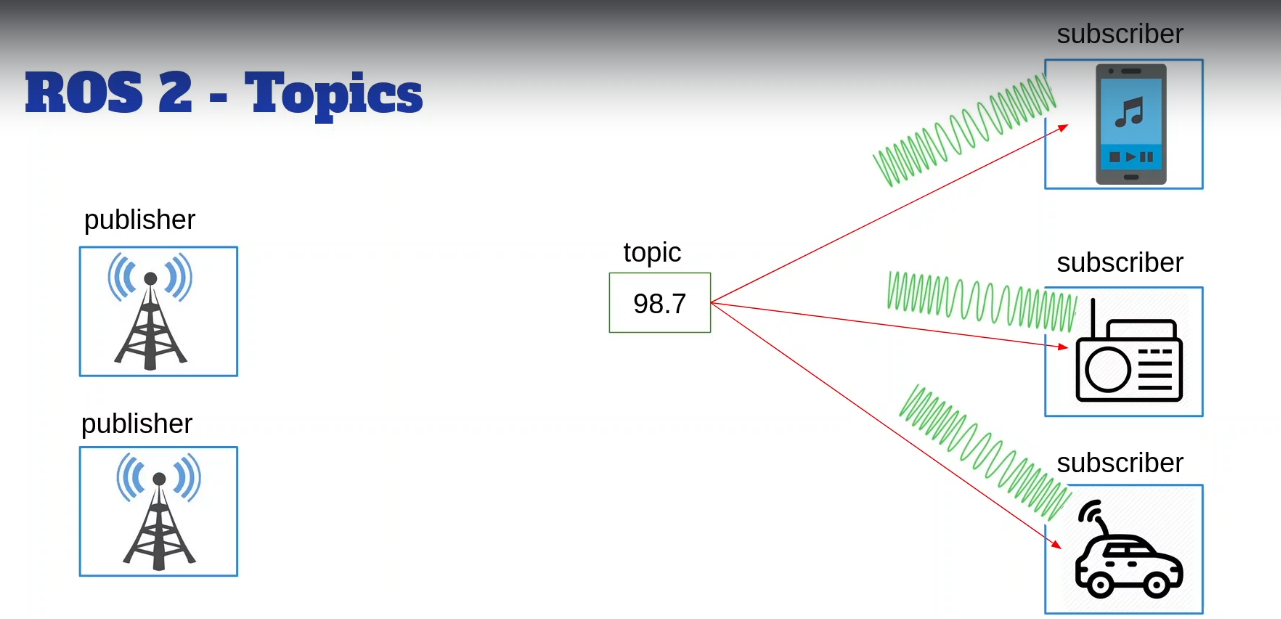
No final desta secção, serás capaz de **fazer os teus nós comunicarem entre si**, utilizando **Tópicos do ROS 2**.

**📋 O que vais fazer nesta secção:**

* Compreender **o que é um tópico ROS 2**, através de uma analogia com a vida real;
* Escrever o teu próprio tópico (**publisher/subscriber**) em **Python** e **C++**;
* **Depurar tópicos** usando as ferramentas do ROS 2;
* Fazer experiências com o **Turtlesim**;
* E, por fim, realizar uma **atividade prática** para treinar tudo o que aprendeste até agora.

**❓ E agora… o que é afinal um Tópico no ROS 2?**

DIA 07 DE julho



**🔊 Analogia com Rádio FM – ROS 2 Topics**

* **Publisher (emissor de rádio)**: Cada torre representa um **nó que publica mensagens** num tópico (neste caso, a “frequência” 98.7).
* **Topic (frequência 98.7)**: É o **canal de comunicação**. Publicadores enviam dados para este canal, e todos os interessados podem escutar.
* **Subscribers (receptores)**: Os dispositivos (telemóvel, rádio, carro) são **nós que subscrevem** esse tópico. Eles **recebem os dados publicados**, desde que estejam “sintonizados” na mesma frequência (ou seja, subscritos ao mesmo tópico).

**✅ Pontos-chave (resumo de professor):**

* Um **tópico** em ROS 2 é um canal de comunicação assíncrona entre nós.
* Um ou mais nós **publicam** dados num tópico.
* Um ou mais nós **subscrevem-se** a esse tópico para **receber os dados**.
* Não há ligação direta entre publisher e subscriber – apenas usam o mesmo nome de tópico.