ROS 2 — BOOTCAMP

Jour 1 — Introduction

ROS 2 · écosystème · concepts · CLI

Etienne Schmitz

Au programme

1. Ce qu'est ROS 2 et son **écosystème**
2. L'**architecture** : graphe de nœuds, RMW, DDS
3. Les **concepts** : nodes, topics, services, actions, paramètres
4. L'**organisation** d'un projet : packages, workspace, launch, CLI
5. La **mise en pratique** : découverte des tutoriels ROS 2 + préparation du projet

PARTIE 01 · PRÉSENTATION

Un écosystème open-source

Middleware, distributions et communication : le socle de ROS 2.

Qu’est-ce que ROS 2 ?

ROS = **R**obot **O**perating **S**ystem.
Mais malgré son nom, **ce n’est pas un système d’exploitation** : ROS 2 est un **middleware** + un **écosystème d’outils** pour construire des applications robotiques.

Qu’est-ce que ROS 2 ?

ROS = **R**obot **O**perating **S**ystem.
Mais malgré son nom, **ce n’est pas un système d’exploitation** : ROS 2 est un **middleware** + un **écosystème d’outils** pour construire des applications robotiques.

- 🧩 un robot = plusieurs programmes (**nœuds**) : perception, décision, contrôle

Qu’est-ce que ROS 2 ?

ROS = **R**obot **O**perating **S**ystem.
Mais malgré son nom, **ce n’est pas un système d’exploitation** : ROS 2 est un **middleware** + un **écosystème d’outils** pour construire des applications robotiques.

- 🧩 un robot = plusieurs programmes (**nœuds**) : perception, décision, contrôle
- 🔗 ROS 2 les fait **communiquer** entre eux (topics, services, actions)

Qu’est-ce que ROS 2 ?

ROS = **R**obot **O**perating **S**ystem.
Mais malgré son nom, **ce n’est pas un système d’exploitation** : ROS 2 est un **middleware** + un **écosystème d’outils** pour construire des applications robotiques.

- 🧩 un robot = plusieurs programmes (**nœuds**) : perception, décision, contrôle
- 🔗 ROS 2 les fait **communiquer** entre eux (topics, services, actions)
- ♻️ et fournit des **briques réutilisables** : on ne réécrit pas tout pour chaque robot

Qu’est-ce que ROS 2 ?

ROS = **R**obot **O**perating **S**ystem.
Mais malgré son nom, **ce n’est pas un système d’exploitation** : ROS 2 est un **middleware** + un **écosystème d’outils** pour construire des applications robotiques.

- 🧩 un robot = plusieurs programmes (**nœuds**) : perception, décision, contrôle
- 🔗 ROS 2 les fait **communiquer** entre eux (topics, services, actions)
- ♻️ et fournit des **briques réutilisables** : on ne réécrit pas tout pour chaque robot

> 💡 **En une phrase**
>
> ROS 2 = un **langage commun** + une **boîte à outils** pour passer du prototype au robot industriel.

Ce que ROS 2 fournit

Ce que ROS 2 fournit

🧰 Bibliothèques

API C++/Python (`rclcpp`, `rclpy`), `tf2`, calcul de trajectoires, `ros2_control`

Ce que ROS 2 fournit

🧰 Bibliothèques

API C++/Python (`rclcpp`, `rclpy`), `tf2`, calcul de trajectoires, `ros2_control`

🖥️ Outils

Simulation (Gazebo), visualisation (RViz, Foxglove), enregistrement (`rosbag2`), build (`colcon`)

Ce que ROS 2 fournit

🧰 Bibliothèques

API C++/Python (`rclcpp`, `rclpy`), `tf2`, calcul de trajectoires, `ros2_control`

🖥️ Outils

Simulation (Gazebo), visualisation (RViz, Foxglove), enregistrement (`rosbag2`), build (`colcon`)

📐 Conventions

Messages standard (`sensor_msgs`...), URDF/SDF, unités SI, nommage des topics

Ce que ROS 2 fournit

🧰 Bibliothèques

API C++/Python (`rclcpp`, `rclpy`), `tf2`, calcul de trajectoires, `ros2_control`

🖥️ Outils

Simulation (Gazebo), visualisation (RViz, Foxglove), enregistrement (`rosbag2`), build (`colcon`)

📐 Conventions

Messages standard (`sensor_msgs`...), URDF/SDF, unités SI, nommage des topics

🌍 Communauté

Des milliers de packages open-source, docs, ROS Discourse, drivers constructeurs

Où l’utilise-t-on ?

Où l’utilise-t-on ?

🧪 Recherche

Prototypage rapide et expérimentation en laboratoire

Où l’utilise-t-on ?

🧪 Recherche

Prototypage rapide et expérimentation en laboratoire

🏭 Industrie

Cobots, automatisation, lignes de production

Où l'utilise-t-on ?

🧪 Recherche

Prototypage rapide et expérimentation en laboratoire

🏭 Industrie

Cobots, automatisation, lignes de production

🚗 Mobilité

Véhicules autonomes, drones, AMR

Où l'utilise-t-on ?

🧪 Recherche

Prototypage rapide et expérimentation en laboratoire

🏭 Industrie

Cobots, automatisation, lignes de production

🚗 Mobilité

Véhicules autonomes, drones, AMR

🦾 Manipulation

Bras manipulateurs et robots de service

Où l'utilise-t-on ?

🧪 Recherche

Prototypage rapide et expérimentation en laboratoire

🏭 Industrie

Cobots, automatisation, lignes de production

🚗 Mobilité

Véhicules autonomes, drones, AMR

🦾 Manipulation

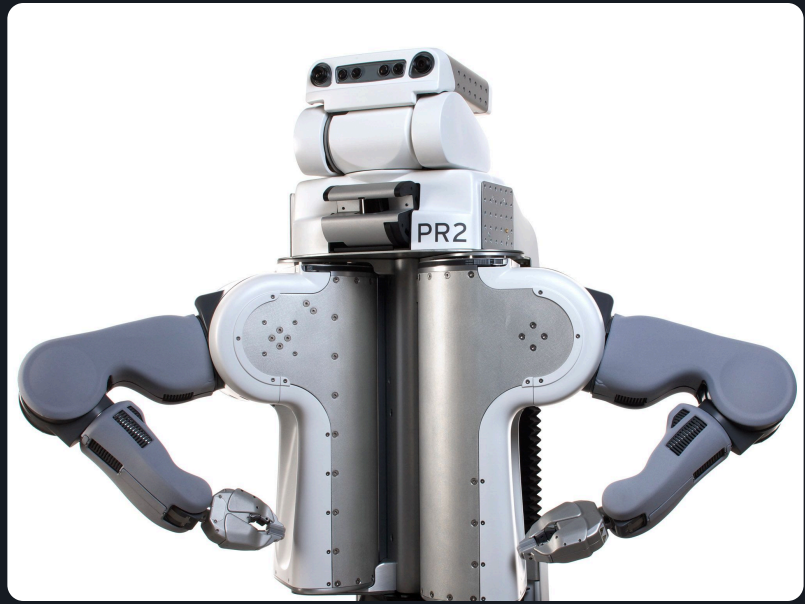
Bras manipulateurs et robots de service

🚀 **Des projets phares construits sur ROS 2**

Autoware pour la conduite autonome, **ROS-Industrial** pour l'usine (ABB, Fanuc, UR…) — on y revient en fin de partie.

ROS 2 — Bootcamp — Jour 1
Introduction

- **2010** ROS 1 (Willow Garage, robot PR2)
- **2012** ROS-Industrial + création OSRF → Open Robotics
- **2017** ROS 2 : **réécriture complète** (temps réel, sécurité, DDS)
- **2020** Noetic — **dernière** distribution ROS 1
- **2025** Noetic en **fin de vie** · Kilted Kaiju, distribution de cette session



De ROS 1 à ROS 2

Réécriture complète pour les besoins modernes de la robotique :

De ROS 1 à ROS 2

Réécriture complète pour les besoins modernes de la robotique :

- ⏱️ **temps réel** mieux géré (middleware **DDS**)

De ROS 1 à ROS 2

Réécriture complète pour les besoins modernes de la robotique :

- ⏱️ **temps réel** mieux géré (middleware **DDS**)
- 🔐 **sécurité** renforcée (chiffrement, authentification)

De ROS 1 à ROS 2

Réécriture complète pour les besoins modernes de la robotique :

- ⏱️ **temps réel** mieux géré (middleware **DDS**)
- 🔐 **sécurité** renforcée (chiffrement, authentification)
- 🧩 **modularité** accrue, architecture propre

De ROS 1 à ROS 2

Réécriture complète pour les besoins modernes de la robotique :

- ⏱️ **temps réel** mieux géré (middleware **DDS**)
- 🔐 **sécurité** renforcée (chiffrement, authentification)
- 🧩 **modularité** accrue, architecture propre
- ↔️ architecture **centralisée → distribuée** (plus de `roscore`)

De ROS 1 à ROS 2

Réécriture complète pour les besoins modernes de la robotique :

- ⏱️ **temps réel** mieux géré (middleware **DDS**)
- 🔐 **sécurité** renforcée (chiffrement, authentification)
- 🧩 **modularité** accrue, architecture propre
- ↔️ architecture **centralisée → distribuée** (plus de `roscore`)

> ⚠️ **Pas de rétrocompatibilité**
>
> Un package ROS 2 ne communique qu'avec ROS 2. Des *bridges* (`ros1_bridge`) permettent de relier ponctuellement les deux mondes — sans plus.

Le cycle des distributions

📅 Une par an

Une nouvelle version chaque **23 mai** (World Turtle Day)

🐢 Nom de tortue

Adjectif + tortue, dans l'ordre **alphabétique**

🛡️ LTS tous les 2 ans

Support **5 ans** (sinon ~1,5 an)

🐧 Calée sur Ubuntu

Chaque distro cible une version d'Ubuntu et **suit son support** (ex. Kilted ↔ Ubuntu 24.04)

Rolling Ridley — la branche de tête

Développement continu : toujours à jour, jamais figée, **sans fin de vie** — mais **non stable**. À réserver aux tests, pas à la production.

Les versions actives

	Distribution	Sortie	Fin de vie	Type
	Makoa Mata-mata	mai 2027	déc. 2028	`à venir`
	**Lyrical Luth**	mai 2026	mai 2031	`LTS`
	**Kilted Kaiju**	mai 2025	déc. 2026	`cette session`
	**Jazzy Jalisco**	mai 2024	mai 2029	`LTS`
	**Humble Hawksbill**	mai 2022	mai 2027	`LTS`

ROS 2 — Bootcamp — Jour 1
Introduction

ROS 2 — Bootcamp — Jour 1
Introduction

ROS 2 — Bootcamp — Jour 1
Introduction

ROS 2 — Bootcamp — Jour 1
Introduction

REP — ROS Enhancement Proposals

Des documents qui **standardisent** ROS (inspirés des **PEP** de Python) :

REP — ROS Enhancement Proposals

Des documents qui **standardisent** ROS (inspirés des **PEP** de Python) :

- 📦 organisation des distributions et des packages

REP — ROS Enhancement Proposals

Des documents qui **standardisent** ROS (inspirés des **PEP** de Python) :

- 📦 organisation des distributions et des packages
- 📨 formats de messages, fichiers, conventions de nommage

REP — ROS Enhancement Proposals

Des documents qui **standardisent** ROS (inspirés des **PEP** de Python) :

- 📦 organisation des distributions et des packages
- 📨 formats de messages, fichiers, conventions de nommage
- 🔧 évolutions du middleware (DDS, RMW…)

REP — ROS Enhancement Proposals

Des documents qui **standardisent** ROS (inspirés des **PEP** de Python) :

- 📦 organisation des distributions et des packages
- 📨 formats de messages, fichiers, conventions de nommage
- 🔧 évolutions du middleware (DDS, RMW…)

> 🧠 Les REP assurent une **gouvernance ouverte** et une **cohérence technique**. Ex : REP 2000 (publication), REP 2002 (rolling).

Avantages

Avantages

⏱️ Gain d’ingénierie

Des briques éprouvées (navigation, perception, contrôle) prêtes à l'emploi

Avantages

⏱️ Gain d’ingénierie

Des briques éprouvées (navigation, perception, contrôle) prêtes à l'emploi

🧩 Modularité

Architecture en nœuds : remplacer un composant sans tout casser

Avantages

⏱️ Gain d’ingénierie

Des briques éprouvées (navigation, perception, contrôle) prêtes à l'emploi

🧩 Modularité

Architecture en nœuds : remplacer un composant sans tout casser

🔓 Pas de lock-in

Standards ouverts, multi-fournisseurs, aucune dépendance constructeur

Avantages

⏱️ Gain d’ingénierie

Des briques éprouvées (navigation, perception, contrôle) prêtes à l'emploi

🧩 Modularité

Architecture en nœuds : remplacer un composant sans tout casser

🔓 Pas de lock-in

Standards ouverts, multi-fournisseurs, aucune dépendance constructeur

🌍 Écosystème riche

Des milliers de packages, simulateurs et outils de debug

Avantages

⏱️ Gain d'ingénierie

Des briques éprouvées (navigation, perception, contrôle) prêtes à l'emploi

🧩 Modularité

Architecture en nœuds : remplacer un composant sans tout casser

🔓 Pas de lock-in

Standards ouverts, multi-fournisseurs, aucune dépendance constructeur

🌍 Écosystème riche

Des milliers de packages, simulateurs et outils de debug

🤝 Communauté

Forums, GitHub, ROS Discourse + support professionnel (intégrateurs)

Avantages

⏱️ Gain d'ingénierie

Des briques éprouvées (navigation, perception, contrôle) prêtes à l'emploi

🧩 Modularité

Architecture en nœuds : remplacer un composant sans tout casser

🔓 Pas de lock-in

Standards ouverts, multi-fournisseurs, aucune dépendance constructeur

🌍 Écosystème riche

Des milliers de packages, simulateurs et outils de debug

🤝 Communauté

Forums, GitHub, ROS Discourse + support professionnel (intégrateurs)

🔁 Du simu au réel

Le même code du simulateur au robot, du prototype à l'industrie

Du simu au réel : un piège classique

Le **même code** passe du simulateur au robot — c'est un vrai atout. Mais ⚠️ **attention** :

> ⚠️ **Le sim-to-real gap**
>
> La simulation **ne reproduit pas tout**. Le réel ajoute **friction**, **bruit des capteurs**, **latences** et **calibration**. Un système qui marche en simu demande **toujours** une phase d'ajustement sur le vrai robot.

Limites

Limites

📘 Apprentissage

Concepts denses (DDS, QoS, tf2, colcon) à assimiler

Limites

📘 Apprentissage

Concepts denses (DDS, QoS, tf2, colcon) à assimiler

🐧 Surtout Linux

Support Windows/macOS partiel ; Docker souvent nécessaire

Limites

📘 Apprentissage

Concepts denses (DDS, QoS, tf2, colcon) à assimiler

🐧 Surtout Linux

Support Windows/macOS partiel ; Docker souvent nécessaire

🔄 Évolution rapide

APIs et distributions changent vite → veille nécessaire

Limites

📘 Apprentissage

Concepts denses (DDS, QoS, tf2, colcon) à assimiler

🐧 Surtout Linux

Support Windows/macOS partiel ; Docker souvent nécessaire

🔄 Évolution rapide

APIs et distributions changent vite → veille nécessaire

🧱 Standards rigides

Parfois inadaptés à des cas très spécifiques

Limites

📘 Apprentissage

Concepts denses (DDS, QoS, tf2, colcon) à assimiler

🐧 Surtout Linux

Support Windows/macOS partiel ; Docker souvent nécessaire

🔄 Évolution rapide

APIs et distributions changent vite → veille nécessaire

🧱 Standards rigides

Parfois inadaptés à des cas très spécifiques

🐢 Latence

La couche middleware ajoute un coût ; tuning QoS requis

Limites

📘 Apprentissage

Concepts denses (DDS, QoS, tf2, colcon) à assimiler

🐧 Surtout Linux

Support Windows/macOS partiel ; Docker souvent nécessaire

🔄 Évolution rapide

APIs et distributions changent vite → veille nécessaire

🧱 Standards rigides

Parfois inadaptés à des cas très spécifiques

🐢 Latence

La couche middleware ajoute un coût ; tuning QoS requis

🧩 Mise en place

Configurer un système complet reste long et technique

Langages supportés

Deux langages **officiels** :

- 🐍 **Python** (`rclpy`) — scripts, prototypage, démos pédagogiques
- ⚙️ **C++** (`rclcpp`) — drivers, nœuds critiques, performance

Autres via *bindings* : 🦀 Rust (`rclrs`), ☕ Java, Ada…

Langages supportés

Deux langages **officiels** :

- 🐍 **Python** (`rclpy`) — scripts, prototypage, démos pédagogiques
- ⚙️ **C++** (`rclcpp`) — drivers, nœuds critiques, performance

Autres via *bindings* : 🦀 Rust (`rclrs`), ☕ Java, Ada…

> 🔬 **micro-ROS**
>
> Une version allégée de ROS 2 pour les **microcontrôleurs** (ESP32, STM32…) : un capteur ou un actionneur rejoint directement le graphe ROS 2.

Robots compatibles

- 🚗 Robots à roues : AGV, AMR (ex. **LeKiwi**)
- 🦾 Cobots et bras manipulateurs (ex. **SO-101**)
- 🚁 Robots volants : drones, UAV
- 🦿 Robots à pattes et humanoïdes

Robots compatibles

- 🚗 Robots à roues : AGV, AMR (ex. **LeKiwi**)
- 🦾 Cobots et bras manipulateurs (ex. **SO-101**)
- 🚁 Robots volants : drones, UAV
- 🦿 Robots à pattes et humanoïdes

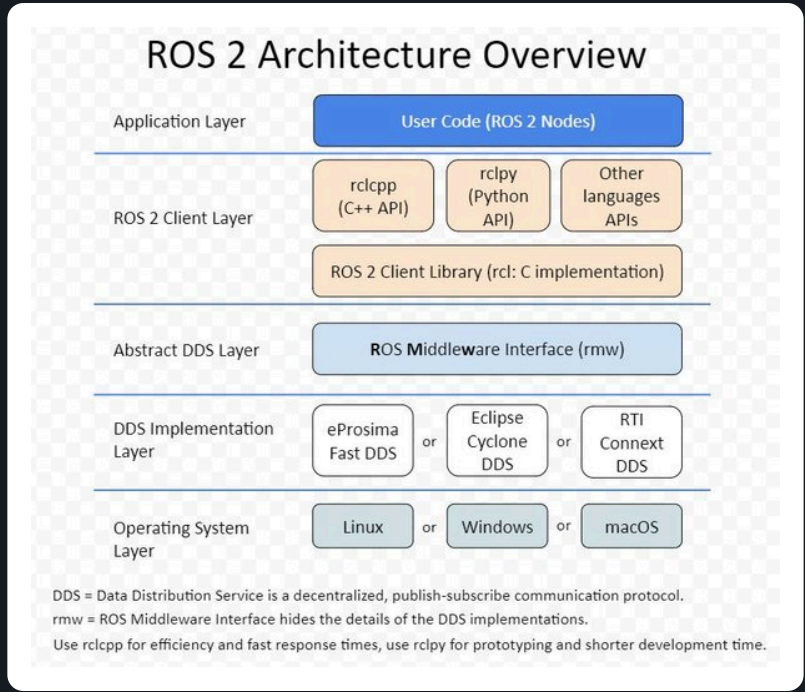
C'est quoi un driver ROS ?

Le **pont logiciel** entre le matériel et le graphe : il publie les données des capteurs en *topics* et exécute les commandes reçues. Fourni par le constructeur, un labo ou la communauté — catalogue sur robots.ros.org.

ROS 2 — Bootcamp — Jour 1 Introduction

De votre **code applicatif** jusqu'au **réseau**, ROS 2 s'organise en **5 couches** empilées : chacune masque la complexité de la suivante.

> On détaille chaque couche dans les slides suivantes.



ROS 2 — Bootcamp — Jour 1
Introduction

De votre code jusqu'au réseau, ROS 2 empile **5 couches**.
Couche 1 / 5 — 🧩 Vos nœuds
Votre **code applicatif**. Chaque nœud (C++ ou Python) réalise une tâche : perception, décision, contrôle. C'est la **seule couche que vous écrivez**.

ROS 2 — Bootcamp — Jour 1
Introduction

Couche 2 / 5 — 🐍 rclcpp / rclpy
L'**API ROS 2**. `rclcpp` (C++) et `rclpy` (Python) exposent nœuds, topics, services… Les deux s'appuient sur `rcl`, une base commune en C → un **comportement identique** entre les langages.

ROS 2 — Bootcamp — Jour 1
Introduction

Couche 3 / 5 — 🔌 RMW
ROS MiddleWare interface : un **adaptateur universel** entre ROS 2 et le DDS.
Chaque DDS a sa propre API → le RMW expose une **interface unique** par-dessus.
→ Changer de DDS = une variable (`RMW_IMPLEMENTATION`), **sans toucher au code**.

ROS 2 — Bootcamp — Jour 1
Introduction

Couche 4 / 5 — 🛰️ DDS
Le **transport réel** : découverte des nœuds, fiabilité et **QoS**. Standard ouvert publié par l'**OMG** (*Object Management Group*, qui maintient aussi UML) → plusieurs implémentations **interchangeables** : **Fast DDS** (défaut), **Cyclone DDS**…

ROS 2 — Bootcamp — Jour 1
Introduction

Couche 5 / 5 — 🐧 OS
Le **système d'exploitation**. ROS 2 tourne surtout sous **Linux (Ubuntu)** ; support Windows/macOS partiel. C'est par le réseau de l'OS que **DDS** fait transiter les messages.

DDS — le moteur réseau

Sous le RMW, ROS 2 utilise **DDS** (*Data Distribution Service*), middleware réseau standardisé par l'**OMG** (qui maintient aussi UML).

DDS — le moteur réseau

Sous le RMW, ROS 2 utilise **DDS** (*Data Distribution Service*), middleware réseau standardisé par l'**OMG** (qui maintient aussi UML).

- 🛠️ **QoS** (*Quality of Service*) : règle la **fiabilité**, le **débit**, l'**historique** et la durée de vie des échanges

DDS — le moteur réseau

Sous le RMW, ROS 2 utilise **DDS** (*Data Distribution Service*), middleware réseau standardisé par l'**OMG** (qui maintient aussi UML).

- 🛠️ **QoS** (*Quality of Service*) : règle la **fiabilité**, le **débit**, l'**historique** et la durée de vie des échanges
- 🔒 **sécurité** (`sros2`) : chiffrement, authentification, contrôle d'accès

DDS — le moteur réseau

Sous le RMW, ROS 2 utilise **DDS** (*Data Distribution Service*), middleware réseau standardisé par l'**OMG** (qui maintient aussi UML).

- 🛠️ **QoS** (*Quality of Service*) : règle la **fiabilité**, le **débit**, l'**historique** et la durée de vie des échanges
- 🔒 **sécurité** (`sros2`) : chiffrement, authentification, contrôle d'accès
- 🔄 **interopérabilité** : Fast DDS, Cyclone DDS, Connext…

DDS — le moteur réseau

Sous le RMW, ROS 2 utilise **DDS** (*Data Distribution Service*), middleware réseau standardisé par l'**OMG** (qui maintient aussi UML).

- 🛠️ **QoS** (*Quality of Service*) : règle la **fiabilité**, le **débit**, l'**historique** et la durée de vie des échanges
- 🔒 **sécurité** (`sros2`) : chiffrement, authentification, contrôle d'accès
- 🔄 **interopérabilité** : Fast DDS, Cyclone DDS, Connext…

> 🔁 **« Changer de DDS », ça veut dire quoi ?**
>
> Toutes ces implémentations suivent le **même standard OMG**. On bascule de l'une à l'autre via la variable `RMW_IMPLEMENTATION`, **sans toucher à votre code** — pour des raisons de licence, de performance, de temps réel ou d'embarqué.

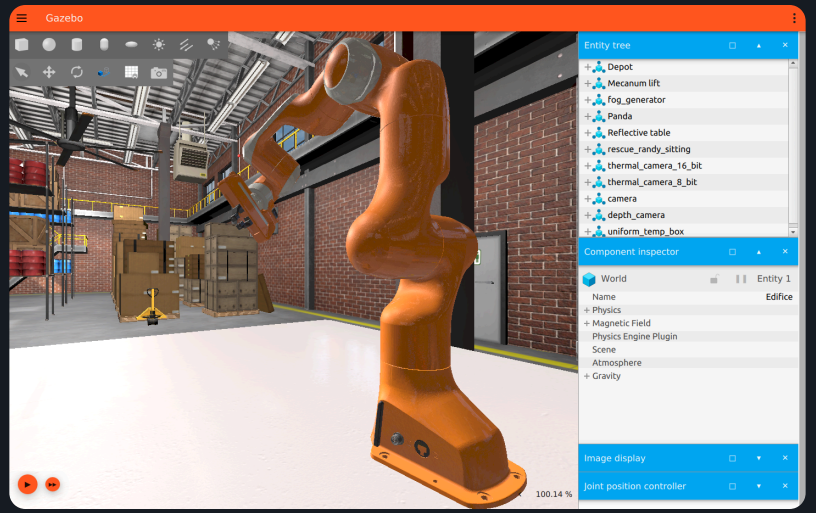
PARTIE 02 · BOÎTE À OUTILS

La boîte à outils

Simulation, visualisation et briques applicatives.

ROS 2 — Bootcamp — Jour 1
Introduction

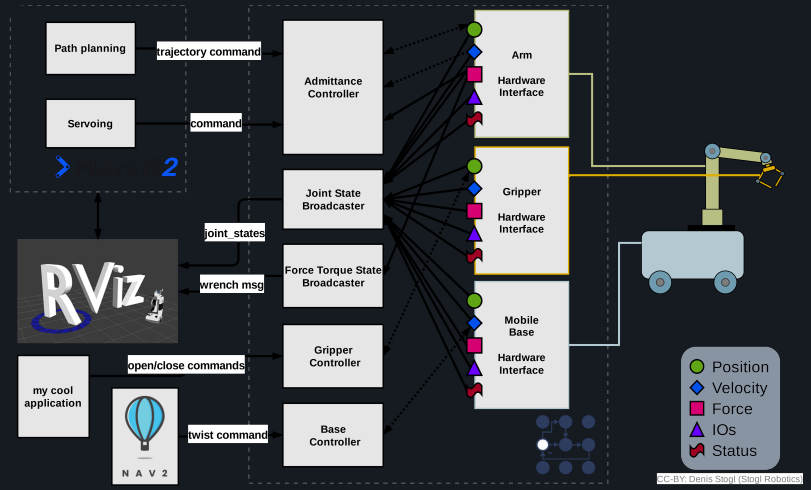
- 🌍 **Gazebo** — simulation physique 3D (capteurs, moteurs)
- 🧭 **RViz** — visualiseur 3D des données ROS
- 🧩 **rqt** — outils graphiques (`rqt_graph` , `rqt_console`)
- 🛰️ **Foxglove** — visualisation moderne (web)



ROS 2 — Bootcamp — Jour 1 Introduction

- 🚗 **Nav2** — Navigation autonome (Jour 2)
- 🦾 **MoveIt 2** — Manipulation (Jour 3)
- ⚙️ **ros2_control** — Contrôle bas-niveau temps réel
- 🌲 **Behavior Trees** — Décision (Nav2, Groot 2)

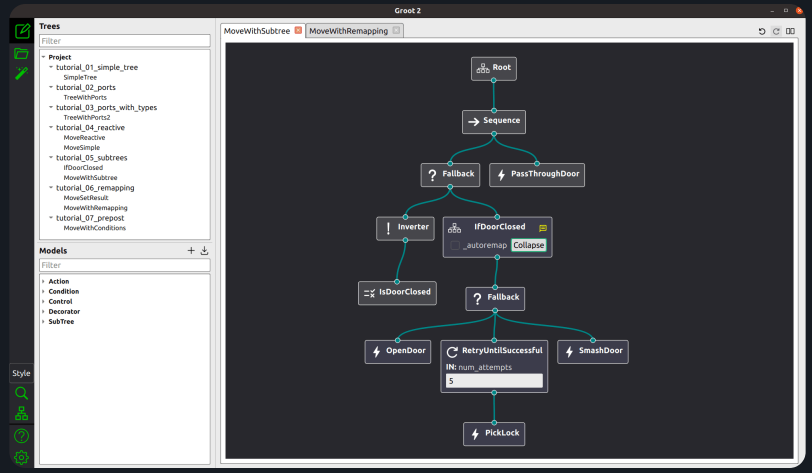
> Le logiciel reste **indépendant du hardware** du robot.



ROS 2 — Bootcamp — Jour 1
Introduction

- 🌲 modèle de décision en **arborescence d'actions**
- remplace les machines à états (FSM)
- utilisé par **Nav2**, MoveIt Task Constructor
- édition visuelle avec **Groot 2**

🔗 behaviortree.dev



Projets connexes

ROS 2 sert de **socle** à de nombreux projets spécialisés :

🚗 Autoware

Conduite autonome open-source (voitures, navettes)

🏭 ROS-Industrial

Besoins industriels et bras constructeurs (ABB, Fanuc, UR...)

🚦 Open-RMF

Coordination de **flottes** de robots de service (hôpitaux, bâtiments)

🛰️ Space ROS

ROS 2 durci pour le **spatial** et les systèmes critiques (NASA)

🔬 micro-ROS

ROS 2 sur **microcontrôleurs** (ESP32, STM32...)

🚁 PX4 / MAVROS

Pont vers les autopilotes de **drones** et UAV

> 🧩 Un écosystème en pleine expansion dans la robotique moderne.

ROS 2 — Bootcamp — Jour 1 Introduction

Tout le monde « parle le même langage » → **interopérabilité**.

- 📏 **Unités SI** : mètre, seconde, radian, newton
- 📨 **Messages standardisés** : `geometry_msgs`, `sensor_msgs`
- 🧩 **Nommage** : `/joint_states`, `/scan`, `/cmd_vel`
- 📂 **Formats** : URDF, SRDF, YAML

La boîte à outils associée

- 🧩 **URDF** — description du robot
- 🔄 **tf2** — transformations entre repères datées
- 🎥 **rosbag2** — enregistrement / rejeu
- 📈 **PlotJuggler** — courbes temps réel
- 📊 **rqt_graph** — vue des nœuds

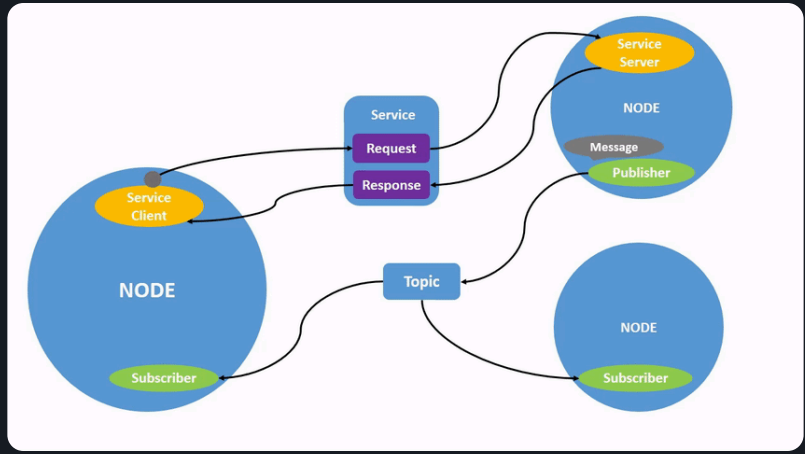
PARTIE 03 · CONCEPTS

Les briques de base

Nœuds, topics, services, actions et paramètres.

ROS 2 — Bootcamp — Jour 1
Introduction

- Un **nœud** = une unité de calcul (exécutable C++ ou Python)
- Chaque nœud fait **une** tâche précise
- Ils communiquent via **topics / services / actions**
- Exemple : **caméra → planif → moteurs**

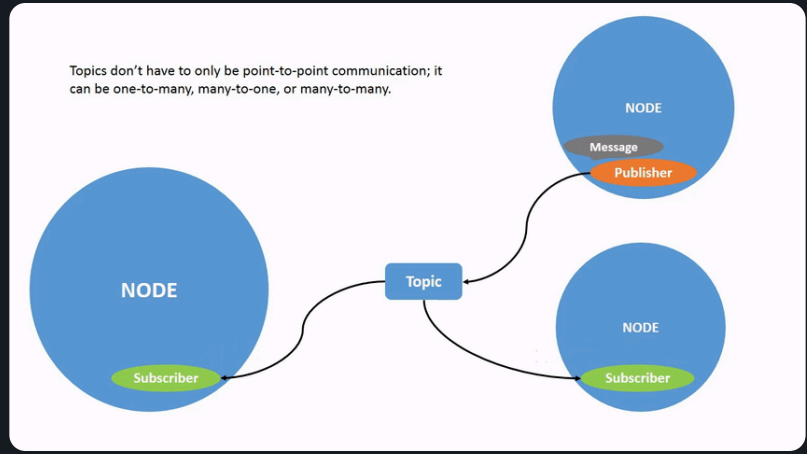


ROS 2 — Bootcamp — Jour 1
Introduction

Le bus **publish / subscribe** — les nœuds publient et s'abonnent **sans se connaître**.

- 📡 **asynchrone** : N publishers, N subscribers
- 🏷️ un **nom** (`/scan`) + un **type** de message
- 🔁 idéal pour les **flux continus** (capteurs, commandes)
- 🕵️ communication **anonyme** et **découplée**

Ex : `/camera/image_raw` → `sensor_msgs/msg/Image`

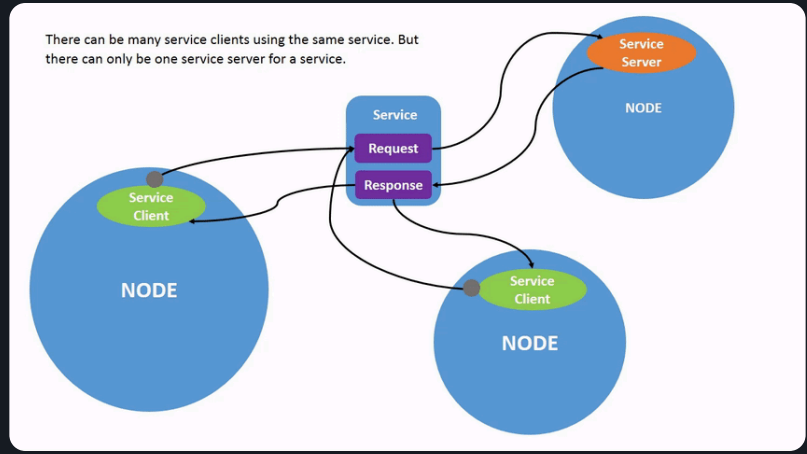


ROS 2 — Bootcamp — Jour 1
Introduction

Communication **synchrone** client → serveur, en **requête / réponse**.

- 🙋 le client **envoie une requête** puis **attend** la réponse
- 👷 **un seul** serveur, plusieurs clients possibles
- ⏱️ pour une **tâche courte** avec un résultat immédiat
- ⚠️ bloquant → **à éviter** pour les tâches longues (→ action)

Ex : « remets le robot à l'origine », « prends une photo »

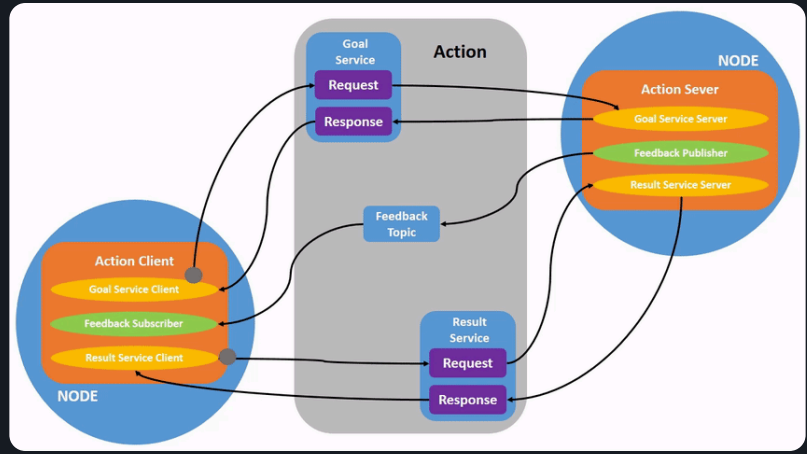


ROS 2 — Bootcamp — Jour 1
Introduction

Pour les **tâches longues**, suivies dans le temps :

- 🎯 le client envoie un **objectif** (goal)
- 📡 le serveur renvoie du **feedback** continu (progression)
- ✅ puis un **résultat** final
- 🛑 **annulable** à tout moment
- 🧱 bâtie sur **topics + services** sous le capot

Ex : « va à cette pose » (Nav2) — on suit la progression, on peut annuler



Paramètres

Configurent un nœud **sans recompiler** — chaque nœud expose ses propres paramètres **typés**.

Paramètres

Configurent un nœud **sans recompiler** — chaque nœud expose ses propres paramètres **typés**.

- 🎛️ seuils, fréquences, vitesses, repères, couleurs…

Paramètres

Configurent un nœud **sans recompiler** — chaque nœud expose ses propres paramètres **typés**.

- 🎛️ seuils, fréquences, vitesses, repères, couleurs…
- 💻 lus / écrits par le **code** ou la **CLI** (`ros2 param list/get/set`)

Paramètres

Configurent un nœud **sans recompiler** — chaque nœud expose ses propres paramètres **typés**.

- 🎛️ seuils, fréquences, vitesses, repères, couleurs…
- 💻 lus / écrits par le **code** ou la **CLI** (`ros2 param list/get/set`)
- 📄 chargés depuis un **fichier YAML** ou surchargés au **lancement**

Paramètres

Configurent un nœud **sans recompiler** — chaque nœud expose ses propres paramètres **typés**.

- 🎛️ seuils, fréquences, vitesses, repères, couleurs…
- 💻 lus / écrits par le **code** ou la **CLI** (`ros2 param list/get/set`)
- 📄 chargés depuis un **fichier YAML** ou surchargés au **lancement**
- 🔔 un nœud peut **réagir** à un changement (callback)

Paramètres

Configurent un nœud **sans recompiler** — chaque nœud expose ses propres paramètres **typés**.

- 🎛️ seuils, fréquences, vitesses, repères, couleurs…
- 💻 lus / écrits par le **code** ou la **CLI** (`ros2 param list/get/set`)
- 📄 chargés depuis un **fichier YAML** ou surchargés au **lancement**
- 🔔 un nœud peut **réagir** à un changement (callback)

Exporter la configuration complète d'un nœud en YAML :

```
ros2 param dump /turtlesim > turtlesim.yaml
```

PARTIE 04 · PROJET

Organiser son code

Workspace, packages, launch et CLI.

ROS 2 — Bootcamp — Jour 1
Introduction

L'**unité de base** de ROS 2 : on **build**, **partage** et **réutilise** par package.

- 📄 `package.xml` — nom, version, **dépendances**
- 🔧 `CMakeLists.txt` (C++) ou `setup.py` (Python)
- 🧩 le code : nœuds, `launch/`, `config/`, `msg/`
- ▶️ créé avec `ros2 pkg create <nom>`

```
mission/
├── package.xml
├── setup.py
├── mission/
│   └── coordinator.py
├── launch/
└── config/
```

ROS 2 — Bootcamp — Jour 1
Introduction

Un dossier qui **regroupe vos packages** dans `src/` et les compile ensemble.

```
ros2_bootcamp_ws/
├── src/
│   ├── mission_interfaces/
│   └── mission/
├── build/  install/  log/
```

- 🏗️ `colcon build` → `build/`, `install/`, `log/`
- 🔌 `source install/setup.bash` rend les packages **disponibles**
- 🧅 **overlay** : votre workspace se superpose à ROS 2 (*underlay*)
- ♻️ `--packages-select <pkg>` pour ne recompiler qu'un package

ROS 2 — Bootcamp — Jour 1
Introduction

Démarrer **tout un système** d'une seule commande, plutôt que chaque nœud à la main.

- 🚀 lance **plusieurs nœuds** d'un coup
- ⚙️ leur passe leurs **paramètres** (YAML)
- 🔀 compose d'autres launch files (réutilisable)
- 🐍 écrit en **Python** (`.launch.py`)

```
ros2 launch mission mission.launch.py
```

```
# mission.launch.py
def generate_launch_description():
    return LaunchDescription([
        Node(package="mission", executable="coordinato
        Node(package="mission", executable="detector")
    ])
```

`ROS_DOMAIN_ID` — isolation réseau

DDS fonctionne par **multidiffusion** sur le réseau local. Pour éviter que les robots se perturbent, on isole les communications par un identifiant (`0` – `232`).

`ROS_DOMAIN_ID` — isolation réseau

DDS fonctionne par **multidiffusion** sur le réseau local. Pour éviter que les robots se perturbent, on isole les communications par un identifiant (`0` – `232`).

- même `ROS_DOMAIN_ID` sur **le robot et le PC**

`ROS_DOMAIN_ID` — isolation réseau

DDS fonctionne par **multidiffusion** sur le réseau local. Pour éviter que les robots se perturbent, on isole les communications par un identifiant (`0` – `232`).

- même `ROS_DOMAIN_ID` sur **le robot et le PC**
- un numéro **unique par groupe** dans la salle

`ROS_DOMAIN_ID` — isolation réseau

DDS fonctionne par **multidiffusion** sur le réseau local. Pour éviter que les robots se perturbent, on isole les communications par un identifiant (`0` – `232`).

- même `ROS_DOMAIN_ID` sur **le robot et le PC**
- un numéro **unique par groupe** dans la salle

  à définir dans `~/.bashrc`

```
export ROS_DOMAIN_ID=12
```

PARTIE 05 · PRATIQUE

À vous de jouer

La première brique du projet final.

Exercices

1. Installation ROS 2 Kilted (natif ou Docker)
2. Exercice 1 — premiers pas avec ROS 2
3. Commencer le projet du fil rouge (**Jours 5-6**) — la première brique

En résumé

🧩 ROS 2

Un middleware + un écosystème : des nœuds qui communiquent

🛰️ Architecture

5 couches : vos nœuds → rcl → RMW → DDS → OS

📨 Communication

Topics (flux), services (requête/réponse), actions (tâches longues)

⚙️ Paramètres

Configurer un nœud sans recompiler

📦 Organisation

Package, workspace (`colcon`), launch file

🛠️ Outils

`ros2` CLI, `rqt_graph`, RViz, Gazebo

> Place à la pratique : montez la **première brique** du projet.

ROS 2 — BOOTCAMP

Questions ?

ROS 2 — Bootcamp · Perpignan 2026

ros2.etienne-schmitz.com