Comparatif entre ROS et ROS2

Il existe des différences entre ROS et ROS2, pas seulement au niveau architectural, mais aussi au niveau des langages, des systèmes d'exploitation supportés, des commandes ..

Différences Architecturales :

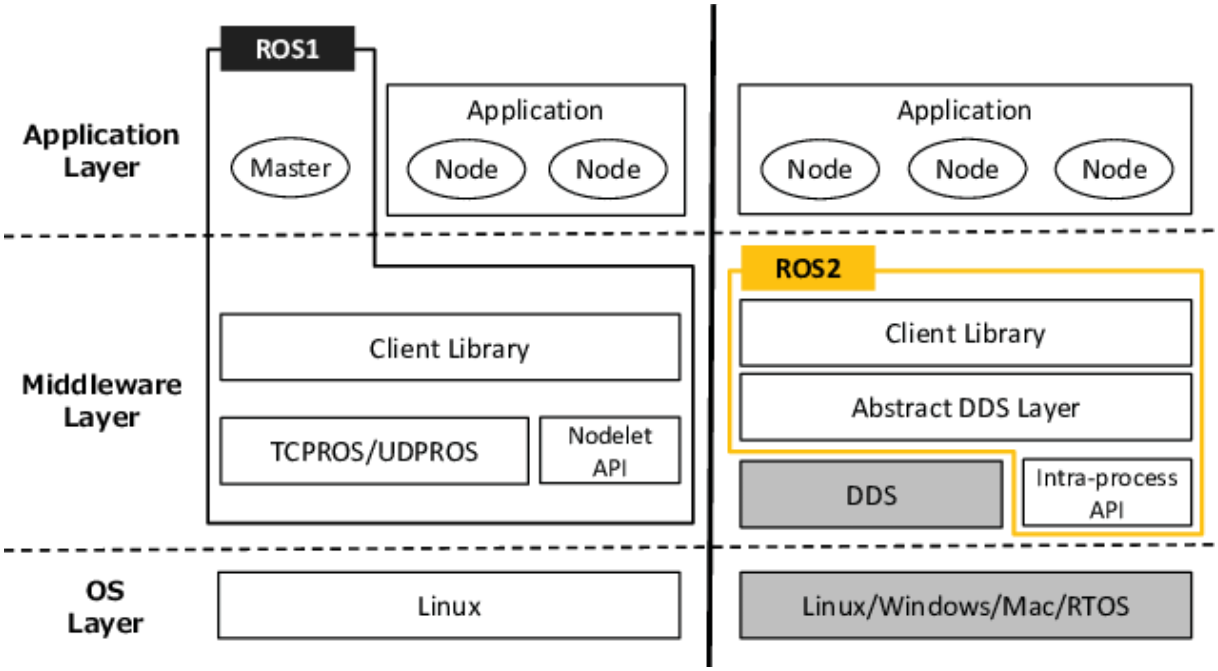

Fig-1-Architecture ROS vs ROS2

Sur le plan architectural, la différence réside notamment au niveau du Middleware Layer (ou intergiciel en Français). ROS2 utilise la norme DDS (Data Centric Middleware) au lieu des protocoles TCPROS/UDPROS utilisés par ROS. La norme DDS est centrée sur les données contrairement à la norme TCP (centrée sur les messages et les services) ce qui offre un découplage spatio-temporel, dans la mesure où les nœuds n'ont pas besoin d'être simultanément actifs au moment de la production et de la consommation des données. De ce fait, la communication devient fondamentalement asynchrone. Ceci explique le fait que ROS2 n'utilise pas le concept de « master ».

Communication :

ROS est un système centralisé qui repose sur un master qui joue le rôle d'intermédiaire dans la communication entre les nœuds. Ce master, doit être exécuté (via la commande roscore) avant le lancement de tout programme ROS.

En ROS2, le « master » n'est plus indispensable, les nœuds peuvent communiquer entre eux directement sans intermédiaire.

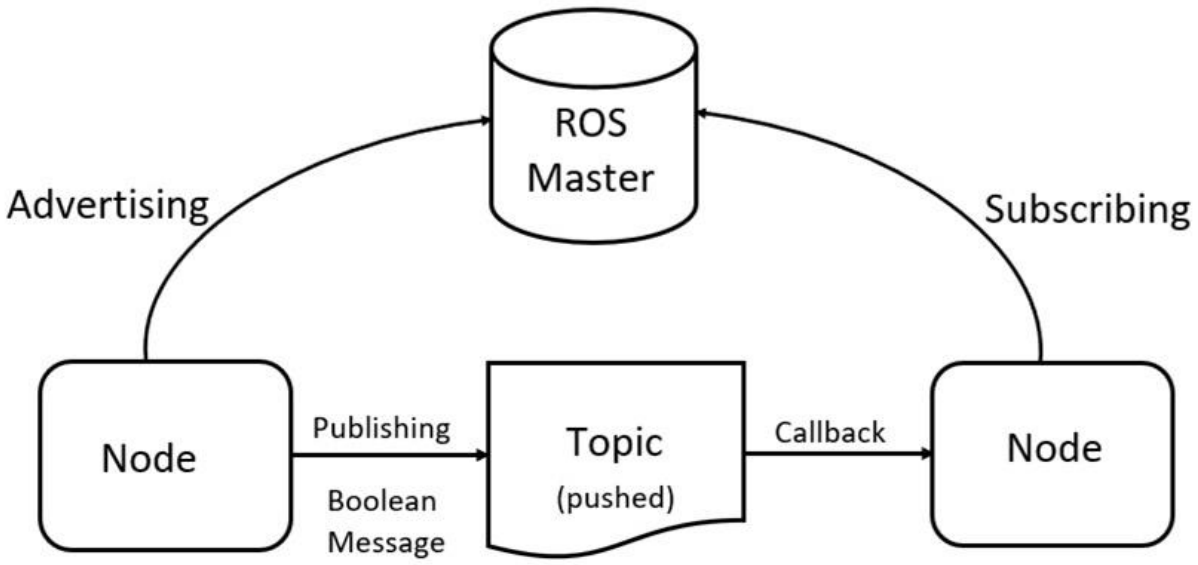

Fig-2- Communication ROS

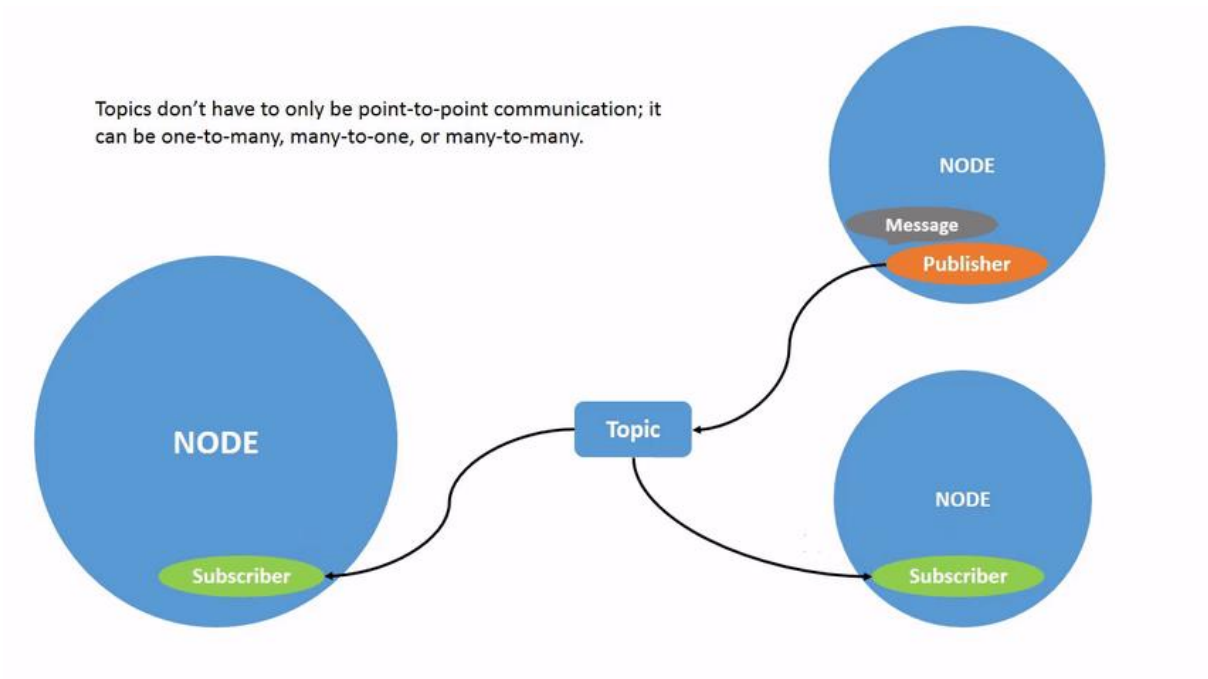

Fig-3- Communication ROS2

Sur ROS, un nœud ne peut pas publier sur plusieurs nœuds à la fois. En contrepartie, un nœud peut écouter plusieurs nœuds à la fois.

Sur ROS2, il est possible à un nœud « publisher » de publier sur un ou plusieurs nœuds à la fois. Il peut également écouter un ou plusieurs Topics.

Langages :

ROS et ROS2 utilisent les deux plusieurs langages de programmation, notamment les langages C++ et python.

Cependant, ROS ne supporte que les anciennes versions de langages telles que C++03 et Python2. Contrairement à ROS2 qui ne tolère pas les versions antérieures à C++11 et Python 3.5.

ROS2 utilise également quelques extensions de C++14, et vise d'être compatible avec C++17 dans le futur.

En ROS, pour le langage C++, on doit utiliser la librairie roscpp et pour Python, on doit utiliser la librairie rospy. Ce qui veut dire que les deux librairies sont complètement indépendantes.

En ROS2 il n'y a qu'une seule librairie de base, nommée RCL et codée en langage C, ce qui facilite le passage d'un langage à un autre.

Contrairement à ROS, ROS2 impose une convention pour la création des nœuds et offre l'opportunité d'avoir un code plus structuré et modulé par la création d'une structure par nœud qui regroupe toutes ses fonctionnalités.

Commandes :

En ROS, il est impossible de créer plus d'un nœud par commande.

En ROS2, il est possible de créer plusieurs nœuds en une seule commande.

En ROS, les fichiers du roslaunch sont écrits en langage XML avec des fonctionnalités très limitées.

En ROS2, les fichiers du launch sont écrits en Python, ce qui rajoute beaucoup de fonctionnalités possibles en utilisant la logique.

Paramètres :

En ROS, les paramètres sont tous gérés par le Master. Or, en ROS2, le master n'existe plus ce qui signifie qu'il n'y aura plus la notion de « paramètre global ». Chaque paramètre est attribué à un nœud spécifique. Chaque nœud gère ses paramètres indépendamment.

Services :

En ROS, les services sont synchrones, contrairement à ROS2, où les services sont asynchrones.

Paquets et environnement de travail « workspace » :

Pour créer et exécuter des paquets, sur ROS, on utilise les commandes « catkin_make » et « catkin_build ». En ROS2, le compilateur « Ament » a remplacé « Catkin ».

En ROS, on crée le paquet et ensuite on spécifie sa nature : un paquet de type python ou bien C++ .. Contrairement à ROS2, dans lequel on doit spécifier à l'avance la nature du paquet avec les commandes « ament_cmake » ou « ament_python ». L'architecture du paquet ne sera pas la même.

Systèmes d'exploitation :

ROS n'est compatible qu'avec le système d'exploitation Linux.

La nouveauté avec ROS2, qu'il est fonctionnel sur les systèmes d'exploitation tels que Linux, OS and Windows.