# Install ROS2 Foxy on Raspberry Pi 4

In this tutorial you will learn how to install ROS2 Foxy Fitzroy on Raspberry Pi 4.

ROS2 is a great framework/middleware for robotics applications. With a Raspberry Pi board you can go much further, and embed a complete ROS2 application inside a small robot.

For example, if you want to build a mobile base with wheels and a camera, well a Raspberry Pi 4 will be perfect for that. You can then create a network of robots all connected together.

## Which Raspberry Pi 4 version for ROS2 (RAM specs)

As you may know, the Raspberry Pi 4 board comes with different possible hardware configurations. You can choose between 2, 4 or even 8GB of RAM (previously also 1GB but discontinued).

If you currently have the 1GB version: 1 GB for Ubuntu Server + ROS2 is enough, but you’ll probably be limited in the future if you try to start more than a few nodes and launch files. If you want to avoid doing some optimization because of the RAM, I suggest you go with the 2GB (or more) version of the board.

After 2GB, then it’s quite simple: more RAM means more expensive. For starting and prototyping with ROS2, 2GB should be enough.

And one thing to keep in mind when using ROS2 on a Raspberry Pi: the Pi is certainly not as powerful as your computer/laptop, so you won’t be able to do 100% of what you do in your laptop, and some heavy features might run slower.

## Prerequisites: install Ubuntu Server 20.04

First you will need to [install Ubuntu Server 20.04 on your Raspberry Pi 4](https://roboticsbackend.com/install-ubuntu-on-raspberry-pi-without-monitor/).

We’ll use Ubuntu Server here because it’s probably the best suited OS for ROS2 on Raspberry Pi. ROS2 is really easy to install and works well on Ubuntu. For this tutorial you won’t need a desktop with Ubuntu on your Pi.

Also, as we’ll install ROS2 Foxy, make sure you have the version **20.04** for Ubuntu, since each version of ROS2 is only compatible with one Ubuntu version.

Once everything is setup and you have an Internet connection + ssh access to your Pi, you can start installing ROS2.

## Install ROS2 on Ubuntu on Raspberry Pi 4

Make sure to execute the commands in the order.

### **Setup locale**

sudo apt update && sudo apt install locales

sudo locale-gen en\_US en\_US.UTF-8

sudo update-locale LC\_ALL=en\_US.UTF-8 LANG=en\_US.UTF-8

export LANG=en\_US.UTF-8

### **Setup sources**

sudo apt update && sudo apt install curl gnupg2 lsb-release

sudo curl -sSL <https://raw.githubusercontent.com/ros/rosdistro/master/ros.key> -o /usr/share/keyrings/ros-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] <http://packages.ros.org/ros2/ubuntu> $(source /etc/os-release && echo $UBUNTU\_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null

### **Install ROS2 core packages**

Now that you have added the ROS2 repository to your sources list, update.

sudo apt update

And now you can install ROS2.

**Important: here we won’t install ros-foxy-desktop, we’ll install ros-foxy-ros-base**, which contains no GUI tools, just the bare minimum you need to write and execute your ROS2 programs.

sudo apt install ros-foxy-ros-base

The ros-foxy-ros-base package is in fact a collection of many other packages, so the list of stuff to install will be pretty big. This step may take a few minutes to complete.

### **Install colcon (build tool)**

After you’ll write some ROS2 code into your own packages, you’ll need to build those packages.

ROS2 uses colcon as a build tool (and ament as the build system). When you only install the ROS2 core packages, colcon is not here, so install it manually.

sudo apt install python3-colcon-common-extensions

### **Auto-completion for ROS2 command line tools**

ROS2 comes with a lot of useful command line tools, and if you want to be able to use auto-completion for those tools, you’ll need to install The argcomplete Python module. First make sure you have pip3 installed.

sudo apt install python3-pip

pip3 install argcomplete

## Setup your environment for ROS2

Great, ROS2 Foxy is now installed on your Raspberry Pi 4 with Ubuntu Server 20.04!

Now, in order to actually use ROS2, you’ll need to setup your environment every time you open a new session (terminal). To source your ROS2 installation in your environment, run

source /opt/ros/foxy/setup.bash

.

And because it won’t be that practical to do this for every new session you open, add this line to your bashrc.

echo "source /opt/ros/foxy/setup.bash" >> ~/.bashrc

So, from now on, every time you open a terminal, your bashrc will be executed and your environment will be ready for ROS2.

## A few tips on how to best use ROS2 on Raspberry Pi

As you saw during the installation, we didn’t install any GUI tool, just the minimum to get things running.

If you need to use some visualization tools, such as Rviz or Gazebo, or if you need to do some heavy processing, then you might use a multi-machine setup with your Pi (or multiple Pis) and a more powerful computer/laptop. It’s very easy to make multiple machines communicate between each other with ROS2. You can run the programs that control hardware on your Raspberry Pi, and then exchange data – through ROS2 communication features – with your laptop, which will handle any GUI or heavy processing tasks.

Also, as you’ve only installed the core base, you might sometimes have to install additional packages.

To install a new ROS2 package, it’s quite easy: the name of the package will be “ros-distribution-package-name”. So if you want to install, on ROS2 Foxy, the example-interfaces package (which contains message and service definitions you can use when you get started with ROS2), you will run

sudo apt install ros-foxy-example-interfaces

.

# ROS2 Multiple Machines Tutorial (including Raspberry Pi)

In this tutorial you will learn how to run ROS2 on multiple machines, including a Raspberry Pi 4 board. As you’ll see, things will be quite easy and there is almost no configuration to do.

If you’re using a Raspberry Pi with ROS2 as one of the machines, make sure you have [correctly installed ROS2 on your Pi](https://roboticsbackend.com/install-ros2-on-raspberry-pi/). Then, as you’ll see, no extra setup is required.

For this tutorial I will personally use a laptop running Ubuntu on a virtual machine (with a bridged adapter), and a Raspberry Pi with ROS2 on Ubuntu Server.

## Network configuration for ROS2 multi-machines

Before starting, obviously make sure you have installed ROS2 on each machine, preferably the same distribution.

Then, make sure you don’t have a firewall blocking your communications on the network. If you have a firewall, allow UDP multicasting, or disable the firewall at least during your first tests.

Connect all your machines in the same network. This is very important, otherwise they simply won’t be able to find each other.

Now, you can easily check if the machines can reach out to each other.

First, get the IP address of each machine inside the network by running

hostname -I

.

Example:

* **Machine 1:**
* hostname -I
* returns 192.168.43.138 172.17.0.1
* **Machine 2:**
* hostname -I
* returns 192.168.43.56

You may have multiple IP addresses on each machine, depending on what you’ve previously configured (ex: on Machine 1 I have Docker, this is why you see 172.17.0.1). Just find the IP addresses that are on the same network, here those who start with 192.168.43.xx.

If you can ping the machines from each other then the network configuration is done.

**Machine 1:**

$ ping 192.168.43.56

PING 192.168.43.56 (192.168.43.56) 56(84) bytes **of** data.

64 bytes from 192.168.43.56: icmp\_seq=1 ttl=64 time=128 ms

64 bytes from 192.168.43.56: icmp\_seq=2 ttl=64 time=136 ms

^C

**Machine 2:**

$ ping 192.168.43.138

PING 192.168.43.138 (192.168.43.138) 56(84) bytes **of** data.

64 bytes from 192.168.43.138: icmp\_seq=1 ttl=64 time=8.75 ms

64 bytes from 192.168.43.138: icmp\_seq=2 ttl=64 time=132 ms

^C

## Run ROS2 on 2 machines

Now it’s very simple. All you have to do is to start some nodes in Machine 1, some other nodes in Machine 2, and they will all be able to communicate through topics, services and actions. Just like they were all in the same machine.

**Machine 1:**

$ source /opt/ros/your\_ros2\_distribution/setup.bash # You can put that line into your ~/.bashrc

$ ros2 run demo\_nodes\_cpp talker

**Machine 2:**

$ source /opt/ros/your\_ros2\_distribution/setup.bash # You can put that line into your ~/.bashrc

$ ros2 run demo\_nodes\_cpp listener

And you should see logs on both machines!

If you want to communicate with a third/fourth/… machine, simply follow the network configuration steps again, and you’ll be all set.

## Use ROS\_DOMAIN\_ID to run multiple (separate) ROS2 applications on the same network

So, after you’ve configured the machines to be in the same network, they are all part of the same ROS2 application. This can be a problem: what if you want to run 2 different ROS2 applications on the same network and on multiple machines? Here you might want to completely separate the applications from each other.

Well, that’s possible, you just need to set one environment variable before you start your nodes.

Before you start any node in one session (= one terminal), you need to export a new environment variable, named ROS\_DOMAIN\_ID, using a number for the value (preferably a low number, between 1 and 232). Then, only the nodes started in sessions with the same ROS\_DOMAIN\_ID will be able to communicate with each other.

Example:

**Machine 1 – session (terminal) A:**

$ export ROS\_DOMAIN\_ID=5

$ source /opt/ros/your\_ros2\_distribution/setup.bash

$ ros2 run demo\_nodes\_cpp talker

**Machine 2 – session (terminal) B:**

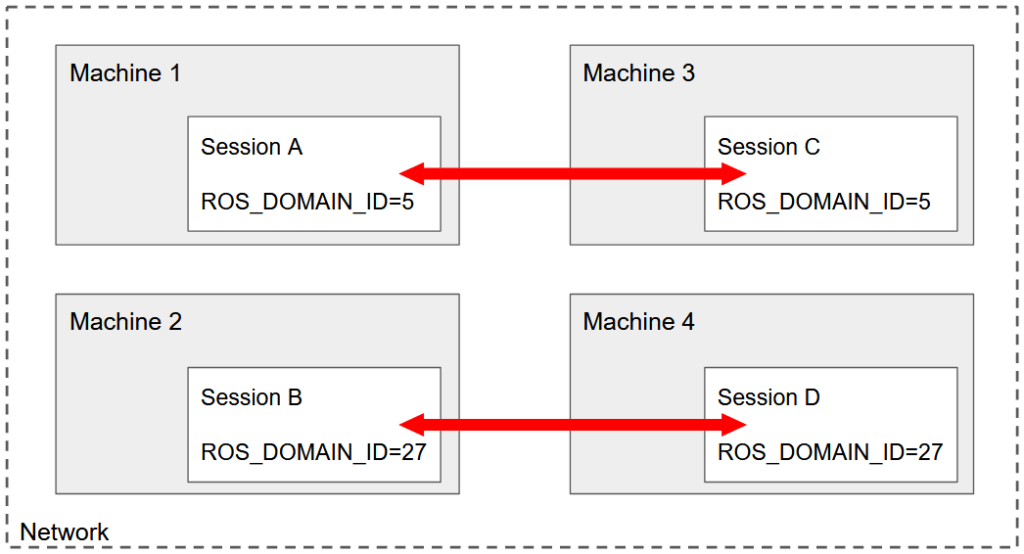
$ export ROS\_DOMAIN\_ID=5

$ source /opt/ros/your\_ros2\_distribution/setup.bash

$ ros2 run demo\_nodes\_cpp listener

Try to set a different ROS\_DOMAIN\_ID (or don’t set one at all) on Machine 2 – session B, and you’ll see that the communication won’t work.

Now, you can have any number of configuration:

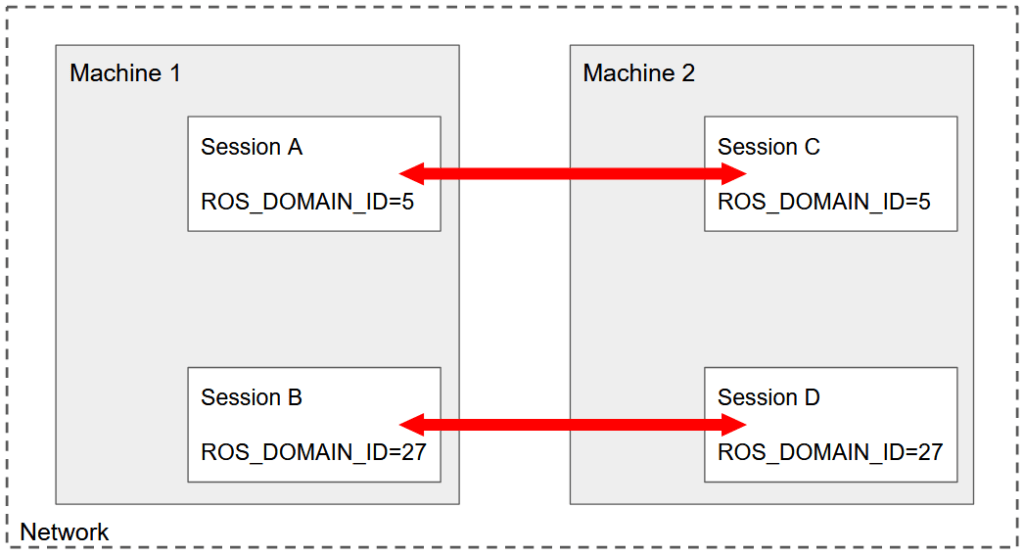


Here you have 4 different machines, each starting one session. This is a classic example of what you can setup in real life.

In the future, make sure you don’t forget the ROS\_DOMAIN\_ID. If you know that all your sessions inside one machine will be used for the same application, you can add

export ROS\_DOMAIN\_ID=xx

to your ~/.bashrc.



But… You can also run multiple ROS2 applications on just 2 machines! What’s important is to understand that the ROS\_DOMAIN\_ID configuration is working at the session level, not the machine level – because it’s an environment variable.

## ROS2 multi-machine with Raspberry Pi

Running ROS2 across multiple machines is especially useful when you have a robot powered by a Raspberry Pi board, or any kind of embedded computer suitable for ROS2 (ex: Jetson Nano). Setting up your Pi for multi-machines communication is the same as setting up any other computer/laptop.

So, here is just an additional tip/best practice to keep in mind whenever you need to work with a robot powered by a Raspberry Pi.

If you want to make some experiments, or work with the robot simulation on the side, or do some heavy computing, etc., here is what you can do:

On your Raspberry Pi, just install the core packages, and run only the core nodes of your applications, which are responsible for talking to the hardware. Then, on your other (remote) computer, start any simulation tool such as RViz, Gazebo. Start your heavy nodes such as motion planning, etc.

This is a great way to minimize the CPU and RAM use on your Pi, and also to speed up your development time.

All in all, multiple machine communication is a really powerful ROS2 functionality. With ROS2 you can build a complete distributed system, not only on the software side, but also on the hardware side.