

Installing Autoware ROS2 Stack

Approximate Time Investment: 3 hours

Autoware is one of the foundations of getting our car to drive autonomously. It is a ROS2 stack that acts as a middleware software between ROS2 and the vehicle's ADAS sensors. It is a large stack that can be containerized in Docker or installed from source. We will install it from source and run some basic simulations to check that it has installed correctly. As previously mentioned, open source software is updated frequently so it is best to read through the references below.

References:

- [Official Autoware Source Installation Guide](#)

Installing the Autoware ROS2 Stack

1. Follow the instructions for setting up a development environment on the Autoware source installation guide site.

How to set up a development environment

1. Clone `autowarefoundation/autoware` and move to the directory.

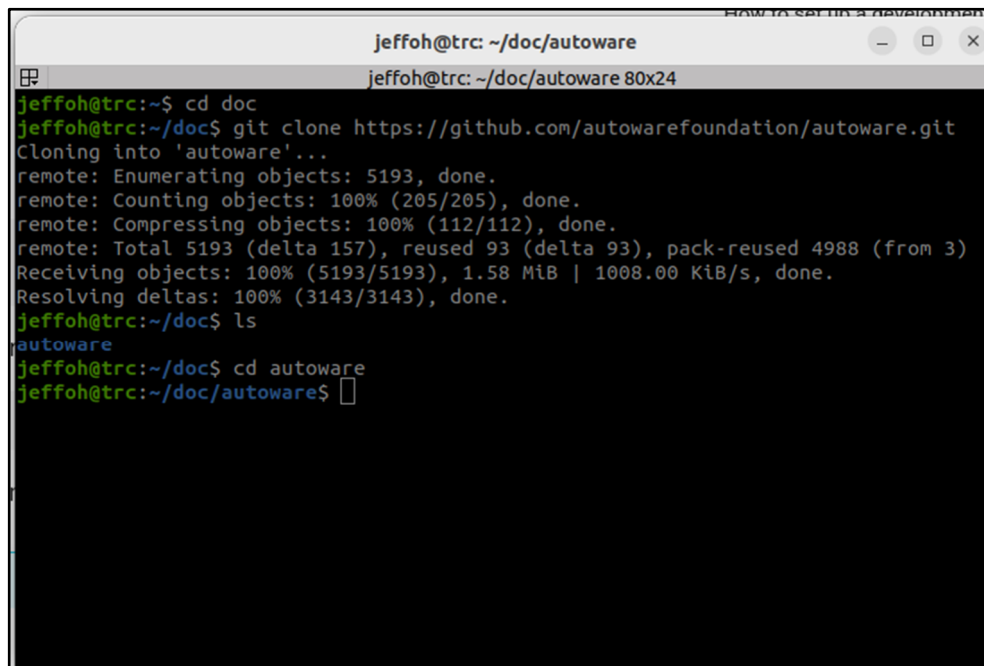
```
git clone https://github.com/autowarefoundation/autoware.git
cd autoware
```

2. If you are installing Autoware for the first time, you can automatically install the dependencies by using the provided Ansible script.

```
./setup-dev-env.sh
```

If you encounter any build issues, please consult the [Troubleshooting](#) section for assistance.

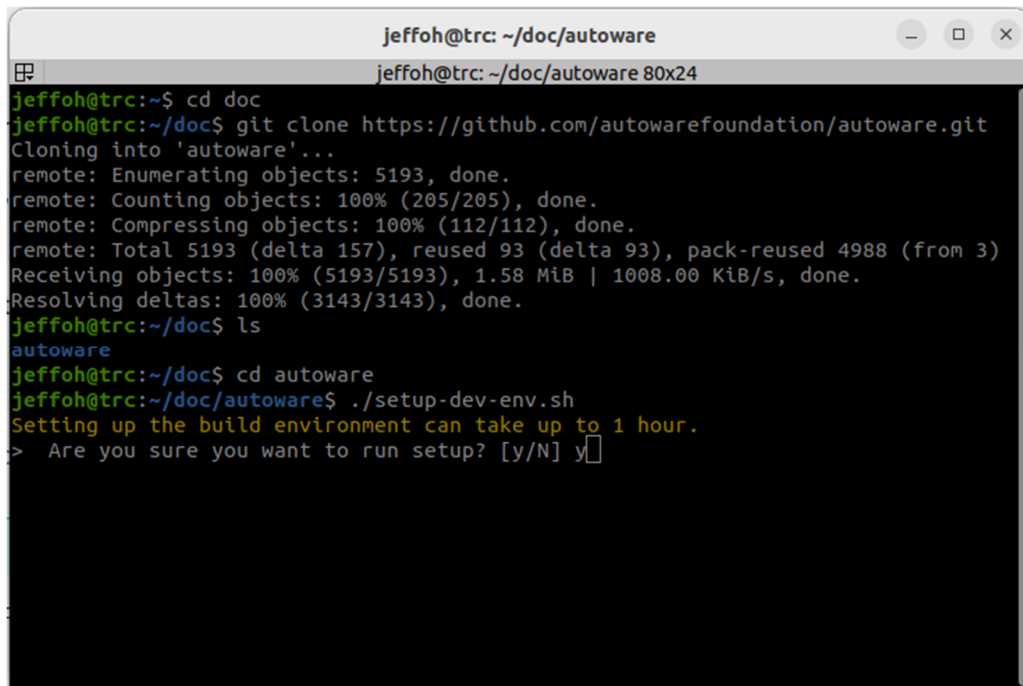
Figure 1: Setting up development environment per Autoware's guide



```
jeffoh@trc: ~/doc/autoware
jeffoh@trc: ~/doc/autoware 80x24
jeffoh@trc:~$ cd doc
jeffoh@trc:~/doc$ git clone https://github.com/autowarefoundation/autoware.git
Cloning into 'autoware'...
remote: Enumerating objects: 5193, done.
remote: Counting objects: 100% (205/205), done.
remote: Compressing objects: 100% (112/112), done.
remote: Total 5193 (delta 157), reused 93 (delta 93), pack-reused 4988 (from 3)
Receiving objects: 100% (5193/5193), 1.58 MiB | 1008.00 KiB/s, done.
Resolving deltas: 100% (3143/3143), done.
jeffoh@trc:~/doc$ ls
autoware
jeffoh@trc:~/doc$ cd autoware
jeffoh@trc:~/doc/autoware$
```

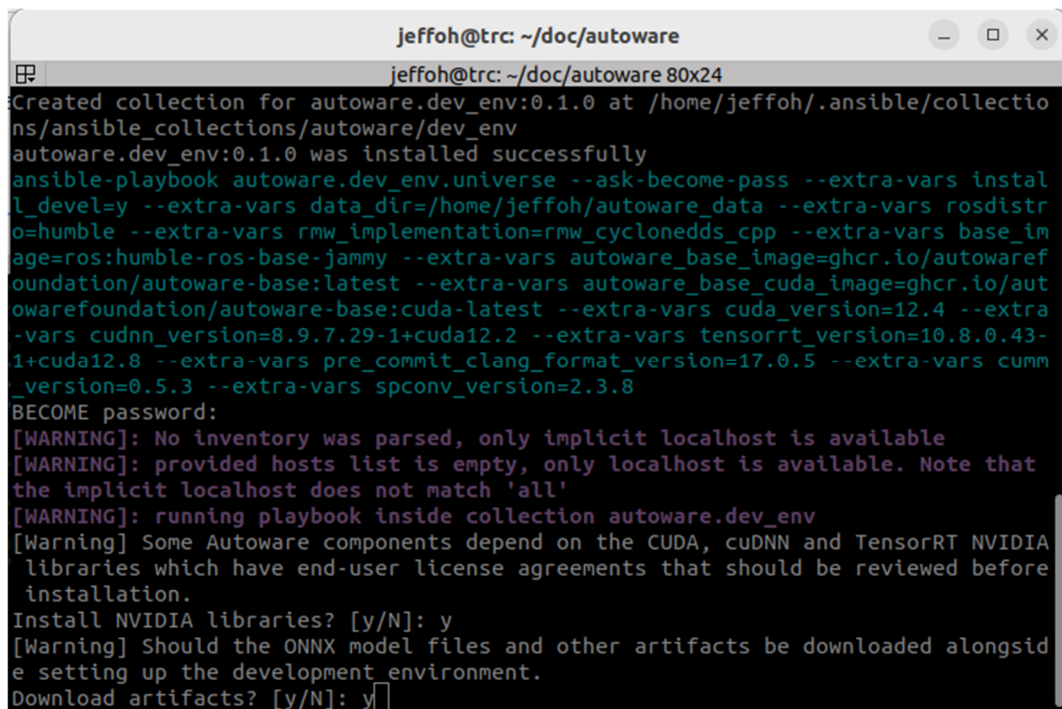
Figure 2: Output of following step 1 in Figure 1

Because I already have an autoware environment in my HOME directory, I have decided to clone the autoware git into a folder called git, see figure 2. You can clone the Autoware repository to any directory best convenient for you.



```
jeffoh@trc: ~/doc/autoware
jeffoh@trc: ~/doc/autoware 80x24
jeffoh@trc:~$ cd doc
jeffoh@trc:~/doc$ git clone https://github.com/autowarefoundation/autoware.git
Cloning into 'autoware'...
remote: Enumerating objects: 5193, done.
remote: Counting objects: 100% (205/205), done.
remote: Compressing objects: 100% (112/112), done.
remote: Total 5193 (delta 157), reused 93 (delta 93), pack-reused 4988 (from 3)
Receiving objects: 100% (5193/5193), 1.58 MiB | 1008.00 KiB/s, done.
Resolving deltas: 100% (3143/3143), done.
jeffoh@trc:~/doc$ ls
autoware
jeffoh@trc:~/doc$ cd autoware
jeffoh@trc:~/doc/autoware$ ./setup-dev-env.sh
Setting up the build environment can take up to 1 hour.
> Are you sure you want to run setup? [y/N] y
```

Figure 3: Running step two of setting up development environment (installing dependencies)



```
jeffoh@trc: ~/doc/autoware
jeffoh@trc: ~/doc/autoware 80x24
Created collection for autoware.dev_env:0.1.0 at /home/jeffoh/.ansible/collections/ansible_collections/autoware/dev_env
autoware.dev_env:0.1.0 was installed successfully
ansible-playbook autoware.dev_env.universe --ask-become-pass --extra-vars install_devel=y --extra-vars data_dir=/home/jeffoh/autoware_data --extra-vars rosdistro=humble --extra-vars rmw_implementation=rmw_cyclonedds_cpp --extra-vars base_image=ros:humble-ros-base-jammy --extra-vars autoware_base_image=ghcr.io/autowarefoundation/autoware-base:latest --extra-vars autoware_base_cuda_image=ghcr.io/autowarefoundation/autoware-base:cuda-latest --extra-vars cuda_version=12.4 --extra-vars cudnn_version=8.9.7.29-1+cuda12.2 --extra-vars tensorrt_version=10.8.0.43-1+cuda12.8 --extra-vars pre_commit_clang_format_version=17.0.5 --extra-vars cumm_version=0.5.3 --extra-vars spconv_version=2.3.8
BECOME password:
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'
[WARNING]: running playbook inside collection autoware.dev_env
[Warning] Some Autoware components depend on the CUDA, cuDNN and TensorRT NVIDIA libraries which have end-user license agreements that should be reviewed before installation.
Install NVIDIA libraries? [y/N]: y
[Warning] Should the ONNX model files and other artifacts be downloaded alongside setting up the development environment.
Download artifacts? [y/N]: y
```

Figure 4: Ansible script for installing dependencies

After running the ansible script in step two, it will ask if you would like to install Nvidia libraries and artifacts. Respond yes to those by typing “y” and pressing enter. The Nvidia libraries will install Cuda and TensorRT drivers which will enable GPU acceleration when using Autoware. The artifacts add-on will install the perception drivers for Autoware.

This dependency install procedure will take anywhere from 30 minutes to 1.5 hours, so be patient and let it run. Once finished, move onto setting up the workspace.

How to set up a workspace

 **Using Autoware Build GUI**

If you prefer a graphical user interface (GUI) over the command line for launching and managing your simulations, refer to the [Using Autoware Build GUI](#) section at the end of this document for a step-by-step guide.

1. Create the `src` directory and clone repositories into it.

Autoware uses `vcstool` to construct workspaces.

```
cd autoware
mkdir src
vcs import src < autoware.repos
```

If you are an active developer, you may also want to pull the nightly repositories, which contain the latest updates:

```
vcs import src < autoware-nightly.repos
```

 **Note:** The nightly repositories are unstable and may contain bugs. Use them with caution.

Optionally, you may also download the extra repositories that contain drivers for specific hardware, but they are not necessary for building and running Autoware:

```
vcs import src < extra-packages.repos
```

Figure 5: Official Autoware instructions for creating `src` folder and cloning critical repositories

2. We will follow Autoware's instructions on setting up the workspace. We should now have the workspace folder "autoware" after completing step 1.
 - a. Run the following the commands to create the "src" folder and to import Autoware packages.

```
cd autoware
mkdir src
vcs import src < autoware.repos
vcs import src < extra-packages.repos
```

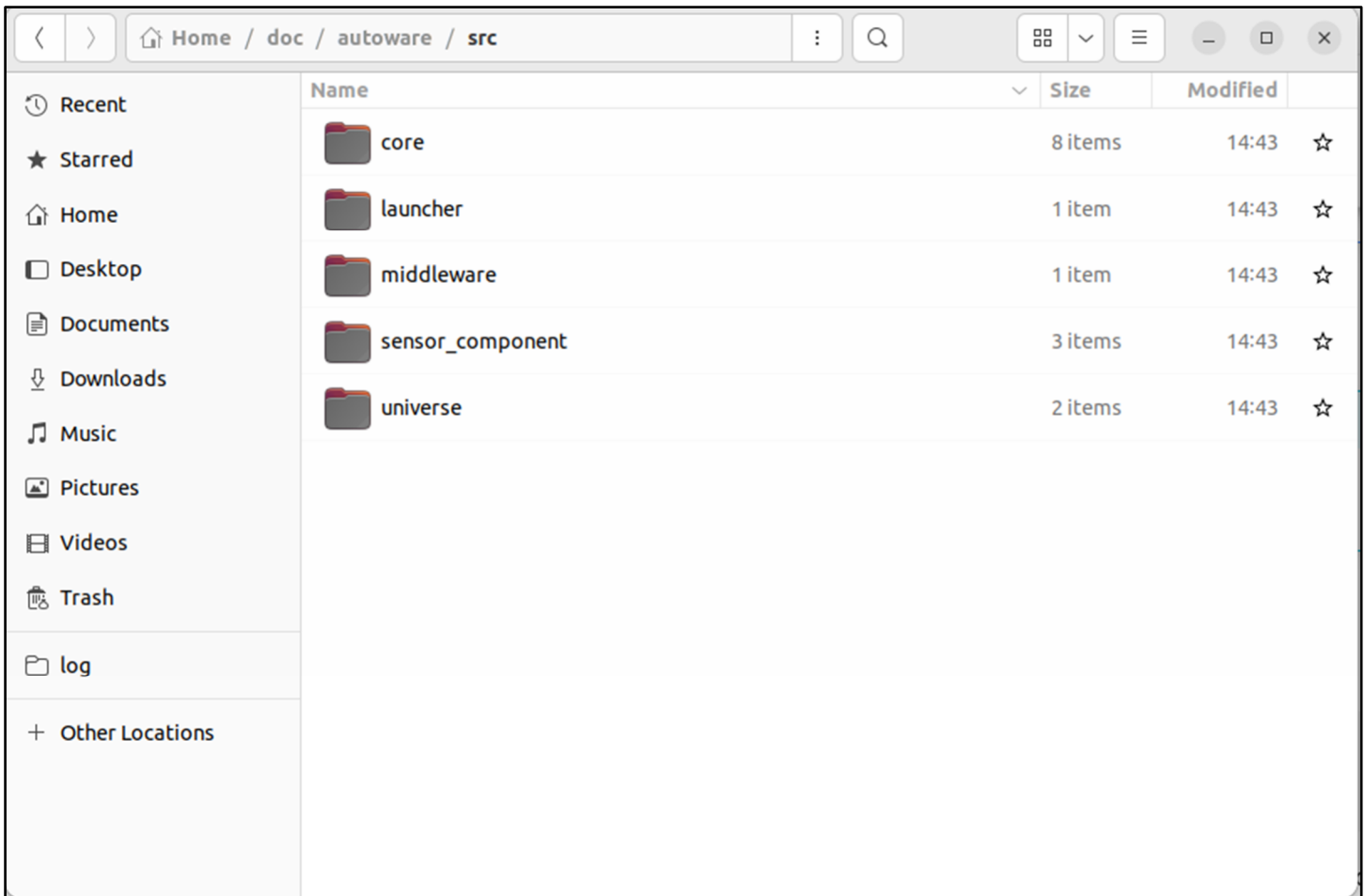


Figure 6: After importing the repositories, our src folder should be populated with Autoware elements

2. Install dependent ROS packages.

Autoware requires some ROS 2 packages in addition to the core components. The tool `rosdep` allows an automatic search and installation of such dependencies. You might need to run `rosdep update` before `rosdep install`.

```
source /opt/ros/humble/setup.bash
# Make sure all previously installed ros-$ROS_DISTRO-* packages are upgraded to their latest version
sudo apt update && sudo apt upgrade
rosdep update
rosdep install -y --from-paths src --ignore-src --rosdistro $ROS_DISTRO
```

3. Install and set up ccache to speed up consecutive builds. (optional but highly recommended)

4. Build the workspace.

Autoware uses `colcon` to build workspaces. For more advanced options, refer to the [documentation](#).

```
colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release
```

If there is any build issue, refer to [Troubleshooting](#).

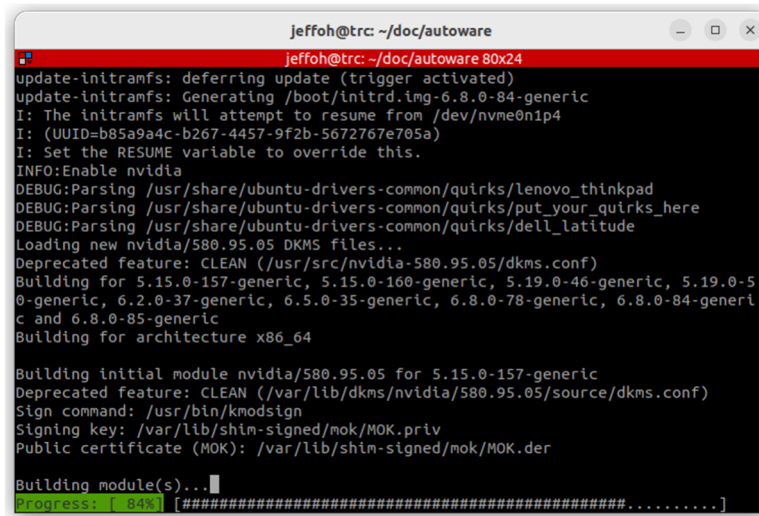
5. Follow the steps in [Network Configuration](#) before running Autoware.

6. Apply the settings recommended in [Console settings for ROS 2](#) for a better development experience. (optional)

Figure 7: Autoware's official instructions continued

- Now we will install dependent ROS packages and then build the workspace. Run the following commands to do so.

```
sudo apt update && sudo apt upgrade
rosdep update
rosdep install -y --from-paths-src --ignore-src --rosdistro
```

A terminal window titled 'jeffoh@trc: ~/doc/autoware' showing the output of 'sudo apt update && sudo apt upgrade'. The output includes messages about updating initramfs, parsing quirks for various hardware (lenovo_thinkpad, dell_latitude), loading NVIDIA DKMS files, and building the initial module for NVIDIA 580.95.05. The progress bar at the bottom shows 84% completion.

```
jeffoh@trc: ~/doc/autoware
jeffoh@trc: ~/doc/autoware 80x24
update-initramfs: deferring update (trigger activated)
update-initramfs: Generating /boot/initrd.img-6.8.0-84-generic
I: The initramfs will attempt to resume from /dev/nvme0n1p4
I: (UUID=b85a9a4c-b267-4457-9f2b-5672767e705a)
I: Set the RESUME variable to override this.
INFO:Enable nvidia
DEBUG:Parsing /usr/share/ubuntu-drivers-common/quirks/lenovo_thinkpad
DEBUG:Parsing /usr/share/ubuntu-drivers-common/quirks/put_your_quirks_here
DEBUG:Parsing /usr/share/ubuntu-drivers-common/quirks/dell_latitude
Loading new nvidia/580.95.05 DKMS files...
Deprecated feature: CLEAN (/usr/src/nvidia-580.95.05/dkms.conf)
Building for 5.15.0-157-generic, 5.15.0-160-generic, 5.19.0-46-generic, 5.19.0-5
0-generic, 6.2.0-37-generic, 6.5.0-35-generic, 6.8.0-78-generic, 6.8.0-84-generi
c and 6.8.0-85-generic
Building for architecture x86_64

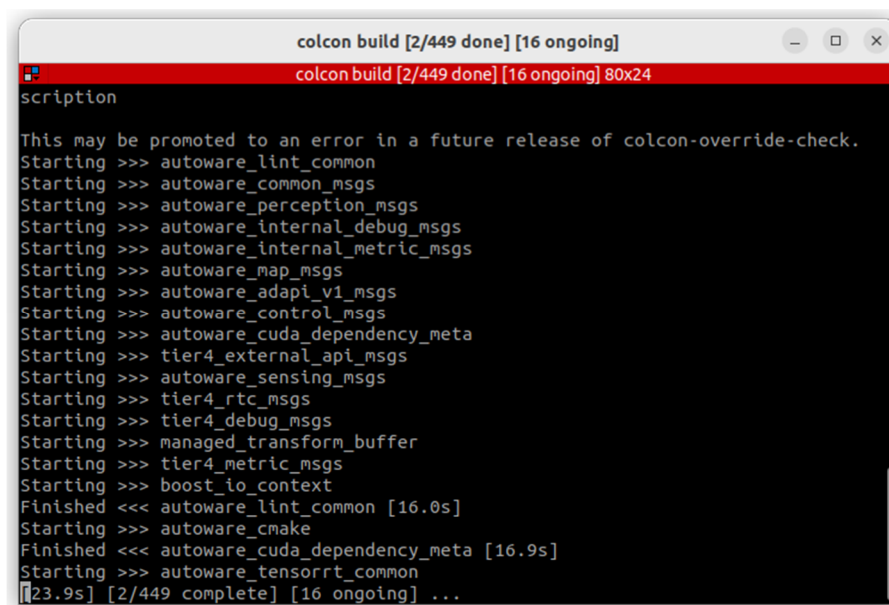
Building initial module nvidia/580.95.05 for 5.15.0-157-generic
Deprecated feature: CLEAN (/var/lib/dkms/nvidia/580.95.05/source/dkms.conf)
Sign command: /usr/bin/kmodsign
Signing key: /var/lib/shim-signed/mok/MOK.priv
Public certificate (MOK): /var/lib/shim-signed/mok/MOK.der

Building module(s)...
Progress: [ 84%] [#####.....]
```

Figure 8: Running sudo apt udate && sudo apt upgrade

- a. Now that we have installed ROS dependency packages, we will build our workspace. While in the autoware directory, run the following command. Make sure that you are in your autoware directory in terminal before running this command. **There are over 400 packages to build so this process has taken 1 hour and 20 minutes on average. There may be standard errors (stderr) when building some of the packages due to missing dependencies or dependency discrepancy/redundancy in the respective package's code but these can be ignored.**

```
colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release
```

A terminal window titled 'colcon build [2/449 done] [16 ongoing]' showing the output of 'colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release'. The output lists various packages being built, including autoware_lint_common, autoware_common_msgs, autoware_perception_msgs, autoware_internal_debug_msgs, autoware_internal_metric_msgs, autoware_map_msgs, autoware_adapi_v1_msgs, autoware_control_msgs, autoware_cuda_dependency_meta, tier4_external_api_msgs, autoware_sensing_msgs, tier4_rtc_msgs, tier4_debug_msgs, managed_transform_buffer, tier4_metric_msgs, boost_io_context, autoware_lint_common, autoware_cmake, autoware_cuda_dependency_meta, and autoware_tensorrt_common. The progress bar at the bottom shows 2/449 complete and 16 ongoing.

```
colcon build [2/449 done] [16 ongoing]
colcon build [2/449 done] [16 ongoing] 80x24
scripton

This may be promoted to an error in a future release of colcon-override-check.
Starting >>> autoware_lint_common
Starting >>> autoware_common_msgs
Starting >>> autoware_perception_msgs
Starting >>> autoware_internal_debug_msgs
Starting >>> autoware_internal_metric_msgs
Starting >>> autoware_map_msgs
Starting >>> autoware_adapi_v1_msgs
Starting >>> autoware_control_msgs
Starting >>> autoware_cuda_dependency_meta
Starting >>> tier4_external_api_msgs
Starting >>> autoware_sensing_msgs
Starting >>> tier4_rtc_msgs
Starting >>> tier4_debug_msgs
Starting >>> managed_transform_buffer
Starting >>> tier4_metric_msgs
Starting >>> boost_io_context
Finished <<< autoware_lint_common [16.0s]
Starting >>> autoware_cmake
Finished <<< autoware_cuda_dependency_meta [16.9s]
Starting >>> autoware_tensorrt_common
[23.9s] [2/449 complete] [16 ongoing] ...
```

Figure 7: Building the Autoware workspace

4. Now that the Autoware stack is built, we will set up automatic sourcing of the Autoware's setup.bash so ROS2 detects the Autoware packages.

- a. Open the bashrc file using the following command.

```
gedit ~/.bashrc
```

- b. Add a line near the bottom for sourcing our Autoware workspace's setup.bash file. The directory will vary depending on where you installed Autoware. See the highlighted source line on figure 8.



```
*.bashrc
~/
Open Save
108 # enable programmable completion features (you don't need to enable
109 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
110 # sources /etc/bash.bashrc).
111 if ! shopt -oq posix; then
112   if [ -f /usr/share/bash-completion/bash_completion ]; then
113     . /usr/share/bash-completion/bash_completion
114   elif [ -f /etc/bash_completion ]; then
115     . /etc/bash_completion
116   fi
117 fi
118
119 source /opt/ros/humble/setup.bash
120 source ~/ros2_ws/install/setup.bash
121 source ~/chanros2_ws/install/setup.bash
122 source ~/autoware/install/setup.bash
123 source ~/doc/autoware/install/setup.bash
124
---
```

Figure 8: bashrc file with Autoware's setup.bash file sourced

5. Now that we have installed Autoware and are automatically sourcing the Autoware workspace's setup, we will check that the Autoware packages are available to us with the following command:

Close your current terminal and open a new one so the Autoware setup is sourced prior to running `ros2 pkg list`.

```
ros2 pkg list
```

You should see packages related to Autoware and the middleware software that came with installation.

Recall that we installed over 400 packages, many of the autoware packages names start with Autoware and the `ros2 pkg list` command may not list packages starting with the letter A.

Another way to sanity check the list of Autoware packages is the following:

```
ros2 launch autoware # DO NOT RUN BUT TYPE THIS OUT
# hit the Tab key twice and it will list all the possible package beginning with "autoware"
```

```
jeffoh@trc: ~  
jeffoh@trc: ~ 92x33  
jeffoh@trc:~$ ros2 launch autoware  
Display all 315 possibilities? (y or n)  
autoware/  
autoware_accel_brake_map_calibrator  
autoware_adapi_adaptors  
autoware_adapi_specs  
autoware_adapi_v1_msgs  
autoware_adapi_version_msgs  
autoware_adapi_visualizers  
autoware_agnocast_wrapper  
autoware_ar_tag_based_localizer  
autoware_auto_common  
autoware_automatic_pose_initializer  
autoware_autonomous_emergency_braking  
autoware_bag_time_manager_rviz_plugin  
autoware_behavior_path_avoidance_by_lane_change_module  
autoware_behavior_path_bidirectional_traffic_module  
autoware_behavior_path_dynamic_obstacle_avoidance_module  
autoware_behavior_path_external_request_lane_change_module  
autoware_behavior_path_goal_planner_module  
autoware_behavior_path_lane_change_module  
autoware_behavior_path_planner  
autoware_behavior_path_planner_common  
autoware_behavior_path_sampling_planner_module  
autoware_behavior_path_side_shift_module  
autoware_behavior_path_start_planner_module  
autoware_behavior_path_static_obstacle_avoidance_module  
autoware_behavior_velocity_blind_spot_module  
autoware_behavior_velocity_crosswalk_module  
autoware_behavior_velocity_detection_area_module  
autoware_behavior_velocity_intersection_module  
autoware_behavior_velocity_no_drivable_lane_module  
autoware_behavior_velocity_no_stopping_area_module
```

Figure 9: List of Autoware packages populated

We can confirm that Autoware has been installed. We will experiment with Autoware's simulation capabilities in the next module.