DevOps Dojo: Build & Run - DE and PE Implementation Discussion #3651

Unanswered **kaspermeck-arm** asked this question in **Ideas**

kaspermeck-arm on Jul 10, 2023 Collaborator edited

Background

The summary below is still WIP and will be updated as discussion leads to decisions.

DevOps Dojo: Build & Run aim to separate the development environment (DE) and the production environment (PE).

Environments

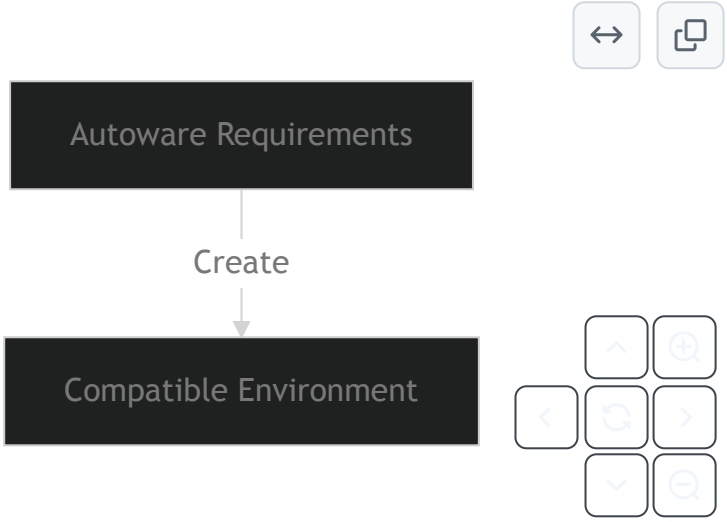
Autoware compatible environments can be implemented in the following ways:

- Bootable image
  - No virtualization
  - Virtual Machine (VM)
- Container

Ideally, it'd be possible to create and configure these environments from a single source of truth which meets the Autoware ROS package requirements.

Dependencies

Software dependencies are required to run Autoware.



Category

Ideas

Labels

DevOps Dojo: Build ...

6 participants

Autoware dependencies:

- ROS upstream packages
  - E.g., RCL
- ROS Autoware packages
  - E.g., Autoware messages
- Drivers
  - E.g., NVIDIA driver
- System-level dependencies
  - 3rd party libraries, e.g., Eigen

Installing Dependencies

Currently, three methods are used to create (and configure) the environment.

Ansible

- multiple playbooks (making it modular)

`rosdep` registry

The idea is to extend the registry with Autoware-specific dependencies. See `rosdep` tutorial for more information.

Pros

Cons

- does not provide version control
  - https://answers.ros.org/question/376259/rosdep-install-specific-version-of-dependencies/

Dockerfile

Configuration

Once the required dependencies are installed, the environment needs to be configured correctly.

Configuration of the environment should be separate from the dependencies. (Note: ROS node parameter and launch files are not part of the environment configuration.)

Approach

What approach should we take to implement the DE and PE?

- Please comment and share your ideas!

Gather Environment Dependencies

1. Gather all the current dependencies for the DE needed to build, test and run Autoware
2. Identify which of these dependencies are relevant for the PE to run Autoware

↑ 2

11 comments · 40 replies

Oldest | Newest | Top

oguzkaganozt on Jul 12, 2023 Maintainer

Hi,

We could start by defining the ideal Dev Environment in case of dependencies. Then from there, we can move on with the runtime environment which means refining dependencies into minimum required ones for runtime only.

We have dependencies from **ansible** which are 3rd party requirements and from **rosdep** which comes directly from the codebase. From a practical perspective, we could start from ansible playbook and try to refine our requirements as DE or PE.

↑ 2 👍 2 2 replies

kaspermeck-arm on Jul 12, 2023 Collaborator Author

That's a good idea and starting point!

1. Gather all the current dependencies for the DE needed to build, test and run Autoware
2. Identify which of these dependencies are relevant for the PE to run Autoware

doganulus on Jul 13, 2023 Collaborator

The Ansible-first approach seems a good solution for several challenges in streamlining robotic application development.

It would be great if the user interacts only with Ansible to perform build, test, and deploy tasks. Its characteristic ability to operate over SSH is also well-fitted for handling bare metal, VMs, and containers alike.

I and my students would work on Ansible scripts starting from this previous work this summer. We love to align it with this Dojo.

kaspermeck-arm on Jul 27, 2023 Collaborator Author

Discussion points:

- Pros/cons with keeping Ansible?
- Pros/cons replacing Ansible with an extended `rosdep` registry?
- How do we create and manage a modular approach for Autoware dependencies?

Please share you thoughts and opinions!

↑ 2 4 replies

oguzkaganozt on Jul 27, 2023 Maintainer

Maybe we can add "Should we store the requirements list inside our codebase ?"

👍 1

doganulus on Jul 27, 2023 Collaborator edited

One problem with `rosdep` is that we cannot install specific versions of dependencies. Please see the following question from a ROS user and answered by Tully Foote:
https://answers.ros.org/question/376259/rosdep-install-specific-version-of-dependencies/
The question is two years old, but I have seen much improvement on `rosdep` since then. And the answer seems definite and says that ROS wants us to use their selection of software versions. Fine, this sounds excellent for beginners, but I think any professional software development activity requires better control over their dependencies. That's why the software industry particularly prefers container-based development and deployment in the first place. Hence, soon, we will want to install new software outside `rosdep` database; and then, we need to use Ansible again or wait for someone to update the database.

@oguzkaganozt Yes, keeping the development environment specification next to the code and under version control is a good software practice. It may be in another repo for large projects with multiple repos, but the overall idea is the same.

👍 1

kaspermeck-arm on Aug 3, 2023 Collaborator Author

> One problem with `rosdep` is that we cannot install specific versions of dependencies. Please see the following question from a ROS user and answered by Tully Foote:
> https://answers.ros.org/question/376259/rosdep-install-specific-version-of-dependencies/ The question is two years old, but I have seen much improvement on `rosdep` since then. And the answer seems definite and says that ROS wants us to use their selection of software versions. Fine, this sounds excellent for beginners, but I think any professional software development activity requires better control over their dependencies. That's why the software industry particularly prefers container-based development and deployment in the first place. Hence, soon, we will want to install new software outside `rosdep` database; and then, we need to use Ansible again or wait for someone to update the database.

A consequence of not following the ROS versioning is that we will have to verify inter-package compatibility. I have two follow-up questions:

- Is not being able to select exact versions of software currently limiting the AWF community?

- If we extend the current `rosdep` registry, are we in charge of the software version then?

> **@oguzkaganozt** Yes, keeping the development environment specification next to the code and under version control is a good software practice. It may be in another repo for large projects with multiple repos, but the overall idea is the same.

+1
DE and PE should definitely be versioned controlled.

esteve on Aug 11, 2023 Maintainer

> Is not being able to select exact versions of software currently limiting the AWF community?

AFAIK, AWF packages just depend on the latest version available, and within Autoware packages, it's expected to build the entirety of the source tree all at once (there's no ABI guarantees, for example)

> If we extend the current rosdep registry, are we in charge of the software version then?

No. The rosdep registry only translates from keys to system-native packages names, versioning is declared in the `package.xml` files. However, info about versions is passed onto packaging system (e.g. Debian) which can correctly use it to restrict versions.

doganulus on Jul 29, 2023 Collaborator edited

Related to this discussion, we have recently submitted a proposal to Autoware Challenge 2023. I hope we are not too far off for the challenge themes, yet this is a topic we would like to pursue regardless. I want to keep this group informed.

In essence, we propose to organize **Autoware repositories as Ansible collections**. Under this model, we refer to `autoware.core` and `autoware.universe` as package collections, but we generalize the concept further to cover any collection of Autoware (ROS) packages, either from universities, companies, or individuals using the power of Ansible and its excellent networking, customization, and social mechanisms. This will be the key to holding everything in place. Overall I think Ansible has great potential to be exploited by the robotics community.

Please find the proposal document here:
https://docs.google.com/document/d/1ZELu47YN2Hq13sXjCXmzJ9YYh1aDqr4jeWLWJ9TrDRQ/edit?usp=sharing
Any comment or suggestion is welcome here or on the document.

@Voursstrreds, **@mehmethilmidundar**

↑ 3 👍 1 0 replies

kaspermeck-arm on Aug 3, 2023 Collaborator Author

Open AD Kit meeting August 3

- focus on building containers to start
- **DE** with CUDA needs to be reduced (currently 16GB)
  - GitHub default runners guarantee 14GB max
  - Ideally, it should be max 12GB

Approach:

1. Understand the current method of building:

- https://github.com/autowarefoundation/autoware/blob/main/.github/workflows/docker-build-and-push-main.yaml
- https://github.com/autowarefoundation/autoware/tree/main/docker
- https://github.com/autowarefoundation/autoware/pkgs/container/autoware-universe
- https://github.com/autowarefoundation/autoware/blob/main/.github/workflows/docker-build-and-push-main.yaml

2. Reduce the container image size

↑ 1 👍 1 0 replies

oguzkaganozt on Aug 7, 2023 Maintainer

Since the last Open AD Kit meeting, I think we've finished the main part of the theoretical discussion. **@kaspermeck-arm**'s summary sums it up perfectly. Therefore, our first goal is to reduce the **DE** container image size, this will clear the ground for optimal runtime images as well as the fix for the CI issue which building container takes a lot of time (autowarefoundation/autoware.universe#4497)

In that regard I want to start the technical discussion on how we can reduce the size of the image:

- Organizing Ansible playbooks for modular installation of packages. As **@doganulus** already commented on this topic at https://github.com/orgs/autowarefoundation/discussions/3651#discussioncomment-6581904
- Do we need to run Autoware for build-test CI task ? If not we can extract runtime dependencies from **DE** such as **CUDA, TensorRT runtime** libs which can be approx. 1Gb and also remove installation of **egm2008-1** which holds up to 500 Mb
- Using **ROS base image** instead of installing it on Ubuntu base. -> Not reduces the image size significantly but we can still leverage the ROS images

↑ 1 👍 1 2 replies

doganulus on Aug 7, 2023 Collaborator

> Do we need to run Autoware for build-test CI task?

I believe we do not need to run Autoware in the most minimal development environment. The container just needs to build all or a subset of Autoware packages. Testing Autoware needs a much larger effort and infrastructure anyway.

> Using ROS base image instead of installing it on Ubuntu base.

I am fine with the ROS image or Ubuntu image. I think it is an easily-revertible decision so we should pick one and start experimenting using it.

kaspermeck-arm on Aug 7, 2023 Collaborator Author

> Using ROS base image instead of installing it on Ubuntu base. -> Not reduces the image size significantly but we can still leverage the ROS images

Another pro is that using the ROS base image removes one layer which AWF needs to maintain.

doganulus on Aug 8, 2023 Collaborator

The following two Nvidia packages already require 9.2GB disk space without ROS (~1GB) and Autoware source dependencies (~5GB) installed.

```
apt install cuda-libraries-dev-12-2 libnvinfer-dev
```

If these packages are not absolutely necessary, then there is a chance. But I am not familiar to the perception stack.

Otherwise, the target of 12 GB does not seem possible under the current monorepo approach.

↑ 2 👍 1 10 replies

Show 5 previous replies

doganulus on Aug 10, 2023 Collaborator

Can we re-evaluate the following dependencies in the build scripts too?

```
find_package(CUDA)
find_package(CUDNN)
```

TensorRT now says CuDNN is an optional dependency (so we can discard it, right?), and I think perception packages depend on `CUDAToolkit` (libraries) rather than `CUDA` (language&compiler) itself. This is another area we can save some space.

👍 1

oguzkaganozt on Aug 15, 2023 Maintainer

I think these are all valuable insights as the dev image size mostly affected by CUDA and TensorRT dependencies. But if we can use `nvcr.io/nvidia/tensorrt` image then we would use at least version 23.05 and it requires driver version at least **530**. Also it means we will be using:

- NVIDIA CUDA® 12.1.1
- NVIDIA cuBLAS 12.1.3.1
- NVIDIA cuDNN 8.9.1.23
- NVIDIA NCCL 2.18.1

Does that introduce any compatability issues in Autoware ?
@mitsudome-r **@xmfcx** **@esteve**

oguzkaganozt on Aug 15, 2023 Maintainer

Just built and played rosbag replay simulation successfully with `nvcr.io/nvidia/tensorrt:23.05` and excluded `egm2008-1.pgm`. The size is now 14.2 Gb

👍 1

esteve on Aug 17, 2023 Maintainer

@doganulus @oguzkaganozt AFAIK only having shared libraries should be fine

👍 2

ambroise-arm on Aug 18, 2023 Collaborator

> Does that introduce any compatability issues in Autoware ?

The current versions that Autoware is built with are defined at https://github.com/autowarefoundation/autoware/blob/main/amd64.env , so it would be a bump in version.
There is an issue to track work on validating the different packages with an update in the tensorrt verion: autowarefoundation/autoware.universe#2330.
In parallel, there has been some work put into updating the cuda version already: #3684, although I don't know exactly what had been done and what remains.

oguzkaganozt on Aug 10, 2023 Maintainer

So how about first defining the ideal modular Ansible scheme and then refining the packages inside of each Ansible rule ?

↑ 1 1 reply

doganulus on Aug 10, 2023 Collaborator

What is the structure you have in mind?

kaspermeck-arm on Aug 10, 2023 Collaborator Author

OADK Meeting August 10th

Development Environment (DE) validation

Validates ROS and up.

1. Colcon build

- see if container can build Autoware or not

2. Run CI test locally on a machine to test new smaller DE container

- if the pipeline passes, then the container should be OK (90%)
- pretty much only tests building

3. Scenario simulator

- https://autowarefoundation.github.io/autoware-documentation/main/tutorials/scenario-simulation/
- regression testing
- scenario simulator is used inside Web.Auto (from Tier IV) and runs scenarios defined by the ODD WG
  - Web.Auto is web-based and can't be run locally

4. Launch comprehensive launch script

- planning simulator (not NVIDIA GPU)
  - https://autowarefoundation.github.io/autoware-documentation/main/tutorials/ad-hoc-simulation/planning-simulation/
- `rosbag` replay (perception, localization)
  - https://autowarefoundation.github.io/autoware-documentation/main/tutorials/ad-hoc-simulation/rosbag-replay-simulation/
- AWSIM (simulation)
  - high threshold, compute and complexity
  - https://autowarefoundation.github.io/autoware-documentation/main/tutorials/ad-hoc-simulation/digital-twin-simulation/awsim-tutorial/

Once 1-4 have been successfully run, the container has passed and is considered validated.

Production Environment (PE) validation

Validates ROS and up. The PE is a subset of the DE.

1. Run the step 3 and 4 from the steps above

- with and without NVIDIA support
- simulator should run on a separate machine or container

↑ 1 0 replies

kaspermeck-arm on Aug 17, 2023 Collaborator Author edited

OADK Meeting August 17th

Development Environment (DE) using NVCR container

There is an upper limit of 14GB container size in GitHub. Using `nvcr.io/nvidia/tensorrt:23.05` as a base, and with all build artifacts, it is `14.2GB`. Previously, some disk analysis has been done, see:

- https://github.com/orgs/autowarefoundation/discussions/2780

The next steps are:

1. reduce the container about another 1GB
2. test in the actual CI using the new container image
    - if this works, then we should be complete

↑ 1 0 replies

doganulus on Aug 18, 2023 Collaborator

@oguzkaganozt **@kaspermeck-arm**

Here is my container experiment. After installing the required CUDA libraries, CUDNN, and TensorRT using Nvidia debs on the ROS base image, I have removed their static libraries. That saved a great deal of disk space and the image is now 7.2G and can build Universe.

Could you please check and test the following build and prebuilt images, if possible?

Container files: https://github.com/bounverif/autoware-istanbul/tree/main/containers
Containers: https://github.com/orgs/bounverif/packages?repo_name=autoware-istanbul

The prebuilt image, which includes build artifacts (~2.5G), is also now reduced below 12G. I believe this image would be similar to what **@xmfcx** wants to use for the CI workflow, as explained at autowarefoundation/autoware.universe#4497.

↑ 1 👍 2 16 replies

Show 11 previous replies

ambroise-arm on Sep 20, 2023 Collaborator

I could save an additional ~190MB by adding `--no-install-recommends` to the `apt install` commands, and by forcing the `rosdep install` to also use this flag (by using simulate on the install command and writing the result to file, then adding the `--no-install-recommends`, and executing the file). It is a small optimization.
For the prebuilt image I think there is also a small optimization to be done by avoiding building the tests (`-DBUILD_TESTING=OFF`). So that it saves build time. Although in autoware.universe that's only a small number of packages that have tests, so it won't make a big difference.

oguzkaganozt on Sep 20, 2023 Maintainer

I tried running lidar_centerpoint and it succeded but when I tried running scenario simulator with
`ros2 launch autoware_launch planning_simulator.launch.xml map_path:=/autoware_map/sample-map-planning vehicle_model:=sample_vehicle sensor_model:=sample_sensor_kit`

I got dependency errors it appears that there is no rosdep installation for exec dependencies so I tried installing all required ones but this time I am getting strange errors at runtime of the scenario simulation.

doganulus on Sep 20, 2023 Collaborator

The container `autoware-build` and even `autoware-prebuilt` are not meant to run Autoware. The former is to build Autoware from source, the latter is just a cache for build artifacts. We must create another `autoware-runtime` image where `exec` dependencies are installed and build artifacts are copied. These runtime images are for the end-users. Depending on the strategy, we may want to run these workflows daily or weekly.

Then, we should be able to use them like this (perhaps with a default launch file):

```
docker run --rm --gpus=all ghcr.io/bounverif/autoware-runtime lidar_centerpoint lidar_centerpoint.launch.xml
```

For simulators and any other third-party tools, I would prefer them separate in their own containers. A compose file may keep things in order.

oguzkaganozt on Sep 20, 2023 Maintainer

Yes I got your point but to be clear I used `autoware-runtime` dockerfile and tried installing exec dependencies on top of it.

doganulus on Sep 20, 2023 Collaborator

> Ah, I got it. That image was experimental (not mentioned in this thread), as I didn't want to install runtime dependencies in the ROS way. For me, declaring dependencies at the package level is too much. I prefer having better control over my dependencies rather than trusting individual developers. But this is another issue.

lukewarmtemp on Oct 6, 2023

Hi, I'm an intern at Red Hat and currently work on the Fedora CoreOS team. I have limited experience with Autoware, but I've worked with ROS for some of my university courses. As part of my personal side project, I have a potential idea I want to share and would love to get some feedback on it. I'll try my best to summarize the discussion so far, but please forgive me if I've misinterpreted anything and feel free to correct me:

What we are looking for is an easy way to set up development and/or production environments for Autoware, which implies installing ROS and Autoware dependencies and configuring the environment. There seems to be some talk about using Ansible scripts to install the dependencies, but I think there's a current focus on running Autoware in a container with the dependencies and environment pre-installed/configured. There also seems to be some success in reducing the size of this container image.

From learning about the context of this discussion thread, CoreOS seems like a great fit here because it supports containerized workloads, lightweight, and immutable (important for security). To provide some context, Red Hat CoreOS is used as the base operating system for OpenShift, Red Hat's enterprise Kubernetes Platform.

Here's the idea that I wanted to share: Fedora CoreOS (the upstream of Red Hat CoreOS) currently supports ostree native containers (https://coreos.github.io/rpm-ostree/container/) which allows the OS to boot and upgrade from a container image. In other words, the container image is not run as a container, but it changes the OS itself. Therefore, you can use a Fedora CoreOS base image, layer dependencies on top of it in a Dockerfile, and then push it to a remote registry. Other people who are running Fedora CoreOS can rebase their system to the container image, and their system will have the dependencies.

As a proof of concept, I've been able to layer the ROS2 source installation on the official Fedora CoreOS container image and can rebase my system to the image. A ROS2 folder is now on my system, and after sourcing the environment, I can run the demo talker and listener nodes successfully.

Dockerfile: https://github.com/lukewarmtemp/ros2-fedora-coreos
Container Image: ghcr.io/lukewarmtemp/ros-fedora-coreos:latest

Here is the document providing the motivation behind layering ROS2 on top of Fedora CoreOS: https://hackmd.io/i1WtwJdDR9KPo8TDQwZw1g?view. I would love to have a discussion as to whether layering Autoware on my image and running it natively on the OS (and not in a container) is something worth pursuing. As a side note, I believe that bootable container images will be a large part of the future and a lot of recent efforts have been working to achieve this (ie. bootc: https://github.com/containers/bootc)

↑ 1 5 replies

doganulus on Oct 6, 2023 Collaborator

I have recently been interested in immutable operating systems outside the Autoware context. However, I am not familiar with internal workings and all existing concepts yet. When you say a bootable container, what are the steps to run the container, starting from writing a Containerfile? A single machine directly runs a single container then? What do we need to install to that machine to run the native container? CoreOS? Thank you very much. This is a really interesting topic.

lukewarmtemp on Oct 10, 2023

Thank you for being interested in my idea and asking good questions! Feel free to ask more!

> When you say a bootable container, what are the steps to run the container, starting from writing a Containerfile

Yes, the first step is to start with a Containerfile that has the Fedora CoreOS image as the base. As seen in the Containerfile in my repository (https://github.com/lukewarmtemp/ros2-fedora-coreos/blob/main/Dockerfile), I have built ROS2 from source on top of the Fedora CoreOS base image in the `/etc/ros2` directory (note: it's not ideal to have it in `/etc` but it's a temporary workaround). You then build the image and push it to your remote repository.

> What do we need to install to that machine to run the native container? CoreOS?

You will need to be on Fedora CoreOS in order to `rebase` to the image. I've added a `README.md` to my repository explaining how to switch over to a native container: https://github.com/lukewarmtemp/ros2-fedora-coreos/blob/main/README.md.

When you successfully `rebase` to the native container, your source of truth is now the remote registry. Therefore, if you layer more packages or make any changes to the Containerfile, rebuild, and re-push it to your remote registry, you can run:

```
$ sudo rpm-ostree upgrade
```

on your system and it will pull the changes to your local system.

doganulus on Oct 12, 2023 Collaborator

Hi **@lukewarmtemp**, thanks for the extensive reply. I think I got the intended workflow when using the native containers. But of course, they brought me new questions:

- How can we run multiple containers on the vehicle? I have the impression that we can rebase only one container on a single virtual or bare metal machine. Would you suggest virtualization (e.g. KVM) on one large machine if we want to use several containers (autoware-perception, autoware-planning, etc.). Modularization is an ongoing discussion in this group.
- How does the size of the container affect the upgrade? Would it be any problem for large images?
- We and the larger Autoware community would love to see a performance comparison between bare-metal installation and native containers. Maybe the work in this group may lead to this as well.

Also, this post in the ROS discourse might be of interest: https://discourse.ros.org/t/automatic-deployment-of-ros2-based-system-to-remote-devices-dual-copy-or-containers/33884

lukewarmtemp on Oct 20, 2023 edited

Hi **@doganulus**, thanks for your questions and apologies for the late response:

> How can we run multiple containers on the vehicle?

Rebasing to a container does not imply that you cannot run containers in your operating system. One of the best examples would be when using Fedora SilverBlue. On Silverblue, your OS is immutable meaning that you will need to use Fedora Toolbox (which is a container) in order to do development on your system. I think that in this context, native containers might be good for setting up the dependencies for Autoware on your OS but users would still take it in their own hands to run the different containers (autoware-perception, autoware-planning, etc.)

> How does the size of the container affect the upgrade? Would it be any problem for large images?

This one is a little complicated to answer. For native containers, there is an algorithm that splits the container image into layers. Therefore, when you run `rpm-ostree upgrade`, it will try and minimize the impact by keeping the layers that are the same and only downloading layers that have changed. In the case that you removed or added a package to the container image, whichever layer the package is part of will be re-downloaded. However, in some cases, packages that didn't change might be re-dowloaded because they lie on the same layer as the package that did change.

> We and the larger Autoware community would love to see a performance comparison between bare-metal installation and native containers. Maybe the work in this group may lead to this as well.

For sure! This is something I would be more than happy to look into.

lukewarmtemp on Oct 24, 2023

> but users would still take it in their own hands to run the different containers

Sorry, after thinking about it a bit more, I had a new idea that might be worth looking into. Rather than only setting up the dependencies when you rebase your system to a native container, you could have a bash script/docker compose file within the native container and a `cronjob` / `systemd` service that starts the autoware containers on boot. This way when you rebase the system, everything is up and running for development out the box. A production context application could also be that updates will require little manual intervention since everything will be handled by the rebase.

Might be thinking a little bit ahead since I need sort out the autoware dependencies within the native container first, but would love to hear opinions regarding this idea.