ADE Documentation

Release 4.4.0

Ternaris

Dec 07, 2021

CONTENTS:

The ADE Development Environment (ADE - *pronounced [ey]-[dee]-[ee]*) is a modular Docker-based tool to ensure that all developers in a project have a **common, consistent development environment**.

With ADE developers:

1. Are confident that what works on one of their computers will also work on all ADE-enabled computers
2. Can comfortably install new packages and dependencies, knowing that the environment can easily be returned to its original state
3. Can mount different volumes (which can be standalone libraries, IDEs, and other tools) and easily switch between versions of these volumes without affecting other parts of the environment

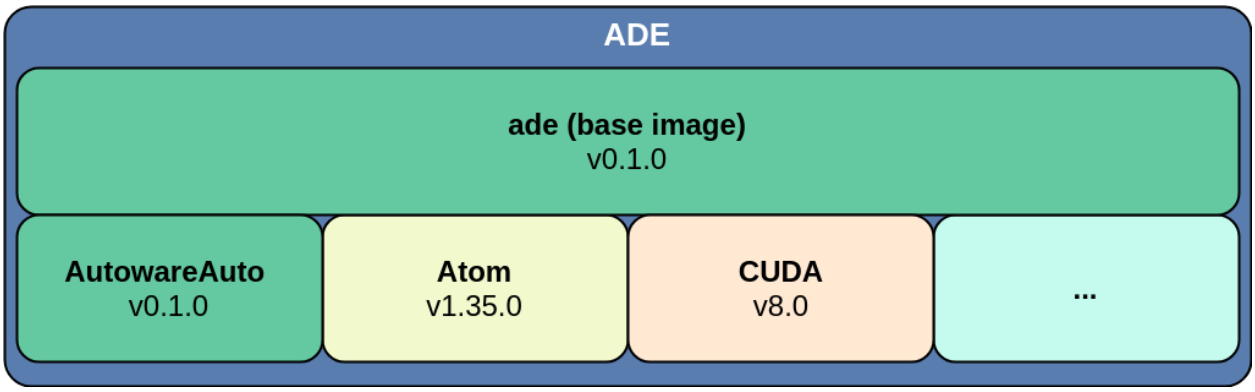
CHAPTER

ONE

INTRODUCTION

ADE creates a Docker container from a base image and mounts additional read-only volumes in `/opt`. The base image provides development tools (e.g. vim, udpreplay, etc.), and the volumes provide additional development tools (e.g., IDEs, large third-party libraries) or released software versions (e.g. the latest AutowareAuto release). Furthermore, ADE enables easy switching between versions of the images and volumes.

1.1 An example of ADE



The image above shows an example of an ADE configuration. The base-image, `ade`, is used to create a container where volumes for *AutowareAuto* (a software library), *Atom* (an IDE), and *CUDA* (a large third-party library) are mounted as volumes.

When ADE is started, it prints out a corresponding matrix: e.g.

```
ade                | v0.1.0    | v0-1-0   | registry.gitlab.com/autowareauto/
↪autowareauto/ade:v0-1-0
autowareauto       | v0.1.0    | v0-1-0   | registry.gitlab.com/autowareauto/
↪autowareauto:v0-1-0
atom               | v1.35.0   | latest   | registry.gitlab.com/apexai/ade-atom:latest
cuda               | v8.0      | v8-0     | registry.private-gitlab/cuda/cuda:latest
```

The first row of the table is the *base image*, all other entries are *volumes* and have a corresponding entry in `/opt`. Each row displays the image name, version information (git hash or tag), the Docker tag, and the FQN for each image. If no version information was provided during the creation of the Docker image, the second column will be empty.

1.2 Terminology

Docker Docker allows the creation of containers with isolated resources from the host OS: "Docker is a computer program that performs operating-system-level virtualization also known as containerization." — Wikipedia Docker (software).

Image Images are snapshots of the filesystem for a container. They are based on a `Dockerfile`, and are created using `docker build`. Images can be uploaded to a *registry*, where they can be downloaded by others.

Container A container is a running instance of an image. When a container is deleted, any changes to its filesystem are usually discarded. A container can publish parts of its filesystem as *volumes*, which can then be mounted by other containers.

Registry A registry is a server-side application that stores pre-built Docker images.

Fully-qualified name (FQN) The FQN is the term for the full URL-like path that describes an image. e.g. `registry.gitlab.com/apexai/ade-atom:latest`

Tag A tag is a specific version of an image. e.g. `latest` in the FQN above

Host The host is the native operating system on which Docker is running.

CHAPTER

TWO

INSTALLATION

2.1 Requirements

- Docker 19.03 or newer, follow the instructions for your OS
- **If the host machine has an NVIDIA graphics card, install `nvidia-container-toolkit`**
  - See the NVIDIA Docker's Repository for more details

2.2 Installation

2.2.1 Linux x86_64 and aarch64

1. Verify that the Requirements above are fulfilled
2. To make `ade` available globally, install it somewhere in your `PATH`: * `echo $PATH` will print a list of possible paths * Commonly used paths are `~/.local/bin` and `/usr/local/bin`
3. Download the statically-linked binary from the Releases page of the `ade-cli` project: e.g. for `x86_64`

```
$ cd /path/from/step/above
$ wget https://gitlab.com/ApexAI/ade-cli/-/jobs/1341322851/artifacts/raw/dist/ade+x86_
↪64
$ mv ade+x86_64 ade
$ chmod +x ade
$ ./ade --version
4.4.0
$ ./ade update-cli
$ ./ade --version
<latest-version>
```

For information on how to install ADE (`ade-cli`, ADE base image, and ADE volumes) on an offline machine, see *Offline Installation*.

2.2.1.1 Offline Installation

`ade-cli` provides facilities to install ADE (`ade-cli`, ADE base image, and ADE volumes) on a machine without an internet connection:

1. On a machine with an internet connection and the same architecture as the target machine, run:

```
$ cd adehome/path/to/.aderc
$ ade start
$ ade save ./offline-install
$ tar cf ade-offline-install.tar ../offline-install
```

2. Copy `ade-offline-install.tar` to the target machine
3. On the target machine, run:

```
$ tar xf ade-offline-install.tar
$ mv ./offline-install/ade path/in/PATH
$ mv offline-install ~/ade-home
$ cd ~/ade-home
$ ade load .
$ touch .adehome
$ ade start
```

2.2.1.2 Update

To update `ade-cli`, run `ade update-cli`. If a newer version is available, `ade` will prompt for confirmation, download the new version, and replace itself.

2.2.2 OSX (Experimental)

1. Verify that the Requirements above are fulfilled
2. Clone the repository and run the `osx-install` script:

```
$ git clone https://gitlab.com/ApexAI/ade-cli
$ cd ade-cli
$ ./osx-install
$ which ade
~/.local/bin/ade
$ ade --version
<version>
```

2.2.2.1 Update

To update on OSX, go to the `git` clone, and run:

```
$ git fetch
$ git checkout <desired version>
```

2.2.2.2 Running X11 Apps on OSX

Note: Running GUI apps on OSX is an experimental feature and requires some additional programs.

1. Install Homebrew
2. Use Homebrew to install `socat`:

```
$ brew install socat
```

3. Install XQuartz v2.7.11:

```
$ brew cask install xquartz
```

4. Restart your computer
5. Open XQuartz -> Preferences -> Security and check 'Allow connections from network clients'

> **Warning:** The security implication of allowing connections from network clients has not been fully analyzed. Enable at your own risk.

6. Test UI apps: e.g. in an Ubuntu-based ADE image:

```
$ sudo apt-get update && sudo apt-get install -y x11-apps
$ xeyes
```

2.3 Autocompletion

To enable autocompletion, add the following to your `.zshrc` or `.bashrc`:

```
if [ -n "$ZSH_VERSION" ]; then
    eval "$(_ADE_COMPLETE=source_zsh ade)"
else
    eval "$(_ADE_COMPLETE=source ade)"
fi
```

See *Usage* for next steps.

CHAPTER

THREE

SETUP

3.1 Creating a custom base image

3.1.1 The `entrypoint` file

As described in *Creating a custom base image*, one of the components of an ADE-compatible base image is the `entrypoint` file:

```
#!/usr/bin/env bash
#
# Copyright 2017 - 2018 Ternaris
# SPDX-License-Identifier: Apache 2.0

set -e

if [[ -n "$GITLAB_CI" ]]; then
    exec "$@"
fi


if [[ -n "$DEBUG" ]]; then
    set -x
fi


if [[ -n "$TIMEZONE" ]]; then
    echo "$TIMEZONE" > /etc/timezone
    ln -sf /usr/share/zoneinfo/"$TIMEZONE" /etc/localtime
    dpkg-reconfigure -f noninteractive tzdata
fi


groupdel "$GROUP" &>/dev/null || true
groupadd -og "$GROUP_ID" "$GROUP"

useradd -M -u "$USER_ID" -g "$GROUP_ID" -d "/home/$USER" -s /bin/bash "$USER"

groupdel video &> /dev/null || true
groupadd -og "${VIDEO_GROUP_ID}" video
gpasswd -a "${USER}" video


for x in /etc/skel/.*; do
```

(continues on next page)

(continued from previous page)

```
    target="/home/$USER/$(basename "$x")"
    if [[ ! -e "$target" ]]; then
        cp -a "$x" "$target"
        chown -R "$USER":"$GROUP" "$target"
    fi
done


if [[ -z "$SKIP_ADEINIT" ]]; then
    for x in /opt/*; do
        if [[ -x "$x/.adeinit" ]]; then
            echo "Initializing $x"
            sudo -Hu "$USER" -- bash -lc "$x/.adeinit"
            echo "Initializing $x done"
        fi
    done
fi


echo 'ADE startup completed.'
exec "$@"
```

Let's break down the main parts:

1. There are a few lines to help debug the `entrypoint`:

```
set -e
```

```
if [[ -n "$DEBUG" ]]; then
    set -x
fi
```

2. The Docker image should behave like a regular Docker image, if it's running in CI:

```
if [[ -n "$GITLAB_CI" ]]; then
    exec "$@"
fi
```

   This ensures that ADE and CI can work together to provide developers a single, reproducible environment.

3. The user's environment is configured inside ADE:

   Set the timezone

```
if [[ -n "$TIMEZONE" ]]; then
    echo "$TIMEZONE" > /etc/timezone
    ln -sf /usr/share/zoneinfo/"$TIMEZONE" /etc/localtime
    dpkg-reconfigure -f noninteractive tzdata
fi
```

   This ensures the programs, like `git commit` will have the correct time.

   Create a user inside the container and make sure it's in the correct groups

```
groupdel "$GROUP" &>/dev/null || true
groupadd -og "$GROUP_ID" "$GROUP"
```

(continues on next page)

(continued from previous page)

```
useradd -M -u "$USER_ID" -g "$GROUP_ID" -d "/home/$USER" -s /bin/bash "
↪$USER"

groupdel video &> /dev/null || true
groupadd -og "${VIDEO_GROUP_ID}" video
gpasswd -a "${USER}" video


for x in /etc/skel/.*; do
    target="/home/$USER/$(basename "$x")"
    if [[ ! -e "$target" ]]; then
        cp -a "$x" "$target"
        chown -R "$USER":"$GROUP" "$target"
    fi
done
```

This ensures that the user on the host and inside ADE behave the same way. This step includes creating a home directory based on `/etc/skel/`, not unlike a Linux distribution creates a home directory

4. It calls the `.adeinit` scripts of any volumes:

```
if [[ -z "$SKIP_ADEINIT" ]]; then
    for x in /opt/*; do
        if [[ -x "$x/.adeinit" ]]; then
            echo "Initializing $x"
            sudo -Hu "$USER" -- bash -lc "$x/.adeinit"
            echo "Initializing $x done"
        fi
    done
fi
```

This gives a way for volumes to provide scripts that must be run on startup to configure the ADE instance appropriately for the volume. See *Creating a custom volume* also.

5. Finally, it starts ADE:

```
echo 'ADE startup completed.'
exec "$@"
```

3.1.2 Writing the Dockerfile

The Dockerfile in the minimal-ade project shows a minimal example of how to create a custom base image:

```
FROM ubuntu:bionic

RUN apt-get update && \
    echo 'Etc/UTC' > /etc/timezone && \
    ln -s /usr/share/zoneinfo/Etc/UTC /etc/localtime && \
    apt-get install -y \
        sudo \
        locales \
        tzdata \
  && rm -rf /var/lib/apt/lists/*
RUN locale-gen en_US.UTF-8; dpkg-reconfigure -f noninteractive locales
```

(continues on next page)

(continued from previous page)

```
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US.UTF-8
ENV LC_ALL en_US.UTF-8

# After apt install sudo
RUN echo 'ALL ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers

COPY env.sh /etc/profile.d/ade_env.sh
COPY entrypoint /ade_entrypoint
ENTRYPOINT ["/ade_entrypoint"]
CMD ["/bin/sh", "-c", "trap 'exit 147' TERM; tail -f /dev/null & while wait $
↪{!}; test $? -ge 128; do true; done"]
```

Let's walk through it:

1. An existing Docker image

```
FROM ubuntu:bionic
```

2. Some convenience programs and requirements of the `entrypoint`:

```
RUN apt-get update && \
    echo 'Etc/UTC' > /etc/timezone && \
    ln -s /usr/share/zoneinfo/Etc/UTC /etc/localtime && \
    apt-get install -y \
        sudo \
        locales \
        tzdata \
  && rm -rf /var/lib/apt/lists/*
RUN locale-gen en_US.UTF-8; dpkg-reconfigure -f noninteractive locales
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US.UTF-8
ENV LC_ALL en_US.UTF-8

# After apt install sudo
RUN echo 'ALL ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
```

3. env.sh:

```
COPY env.sh /etc/profile.d/ade_env.sh
```

   - Configures the default environment on `ade enter`
   - Source the `.env.sh` files provided by volumes (see *Creating a custom volume*)

4. entrypoint :

```
COPY entrypoint /ade_entrypoint
ENTRYPOINT ["/ade_entrypoint"]
```

   See also *The entrypoint file*.

5. `CMD`:

```
CMD ["/bin/sh", "-c", "trap 'exit 147' TERM; tail -f /dev/null & while wait ${!};␣
↪test $? -ge 128; do true; done"]
```

   - This command attaches a process to PID 1 which will keep the container running even if no one is attached to the container

3.1.3 Building the image

Once the `Dockerfile` is ready, build the image by running `docker build` in the directory that contains the `Dockerfile`:

```
docker build -t <image_name>:<tag> .
```

Add `--label ade_image_commit_sha=<git-hash>` and/or `--label ade_image_commit_tag=<git-tag>` to embed VCS information in the Docker image:

```
docker build \
  --label ade_image_commit_sha=<git-hash> \
  --label ade_image_commit_tag=<git-tag> \
  -t <image_name>:<tag> .
```

For more information, see the docker build documentation

3.1.4 Using the image

To use the image, add it to the `.aderc` file as the first entry in `ADE_IMAGES`:

```
export ADE_IMAGES="
   <image_name>:<tag>
   ...
"
```

For more information, see *The .aderc File*

3.2 The `.aderc` File

As described in *Configuring ADE with environment variables*, it is possible to configure ADE using environment variables. In order to make these configurations project-wide, add the environment variables to the `.aderc`. At a minimum, the `.aderc` file should include the list of images

```
export ADE_IMAGES="
  image:latest
"
```

Often, it will also need to include the Gitlab instance and registry, so `ade` knows where to download images:

```
export ADE_GITLAB=gitlab.com
export ADE_REGISTRY=registry.gitlab.com
export ADE_IMAGES="
  registry.gitlab.com/autowareauto/autowareauto/ade:v0-1-0
  registry.gitlab.com/autowareauto/autowareauto:v0-1-0
  registry.gitlab.com/apexai/ade-atom:latest
"
```

Lastly, it will also need to include some extra arguments to `docker run` to enable debugging, for example:

```
export ADE_DOCKER_RUN_ARGS="
  --cap-add=SYS_PTRACE
"
export ADE_GITLAB=gitlab.com
```

(continues on next page)

(continued from previous page)

```
export ADE_REGISTRY=registry.gitlab.com
export ADE_IMAGES="
  registry.gitlab.com/autowareauto/autowareauto/ade:v0-1-0
  registry.gitlab.com/autowareauto/autowareauto:v0-1-0
  registry.gitlab.com/apexai/ade-atom:latest
"
```

See *Custom docker run/exec arguments* for more information.

3.2.1 Writing the `.aderc` file for pull+fork projects

The `.aderc` file supports using variables in the configuration file; this feature can be leveraged to create an `.aderc` file for a project that uses a fork+pull model.

In the example above, someone who forks the AutowareAuto project will need to edit their `.aderc` file to pull the images from their fork's registry instead of the AutowareAuto registry; this change will need to be reverted before the fork gets pulled. The project could avoid this hassle by providing the following instead:

```
export ADE_DOCKER_RUN_ARGS="
  --cap-add=SYS_PTRACE
"
export ADE_GITLAB=gitlab.com
export ADE_REGISTRY=registry.gitlab.com
export ADE_IMAGES="
  registry.gitlab.com/${AW_FORK_NAMESPACE:-autowareauto}/autowareauto/ade:v0-1-0
  registry.gitlab.com/${AW_FORK_NAMESPACE:-autowareauto}/autowareauto:v0-1-0
  registry.gitlab.com/apexai/ade-atom:latest
"
```

If the `AW_FORK_NAMESPACE` variable is not set, the default `autowareauto` namespace will be used. Meanwhile, the developer of the fork can set `AW_FORK_NAMESPACE` in order to pull images from their own registry: e.g.:

```
$ export AW_FORK_NAMESPACE=jpsamper
$ ade start
Starting ade with the following images:
ade          | v0.1.0   | v0-1-0  | registry.gitlab.com/jpsamper/autowareauto/ade:v0-
↪1-0
autowareauto | v0.1.0   | v0-1-0  | registry.gitlab.com/jpsamper/autowareauto:v0-1-0
atom         | v1.35.0  | latest  | registry.gitlab.com/apexai/ade-atom:latest
```

Furthermore, if they are working on `my-branch` instead of `master`, they can use the `--select` option::

```
$ export AW_FORK_NAMESPACE=jpsamper
$ ade start --select my-branch
Starting ade with the following images:
ade          | <sha>    | my-branch  | registry.gitlab.com/jpsamper/autowareauto/
↪ade:my-branch
autowareauto | <sha>    | my-branch  | registry.gitlab.com/jpsamper/autowareauto:my-
↪branch
atom         | v1.35.0  | latest  | registry.gitlab.com/apexai/ade-atom:latest
```

3.3 Creating a custom volume

3.3.1 Writing the Dockerfile

ADE volumes are a way to minimize the size of the base image by loading different resources from a different Docker image. ADE volumes can be used to provide third-party libraries, IDEs, and proprietary programs.

Let's look at how a volume for Atom can be created[1]:

1. A generic Docker file:

```
FROM alpine
COPY _opt /opt/atom
VOLUME /opt/atom
CMD ["/bin/sh", "-c", "trap 'exit 147' TERM; tail -f /dev/null & wait ${!}"]
```

   - Note the `CMD`, which is meant to keep the container running even if no one is attached

2. A script to create `/opt/atom`:

```
#!/usr/bin/env bash
#
# Copyright 2018 Apex.AI, Inc.
# SPDX-License-Identifier: Apache-2.0

set -xe

cd "$(dirname "$(realpath "$0")")"

ADE_VERSION="$1"; shift

# Download and install Atom
curl -LO "https://github.com/atom/atom/releases/download/${ADE_VERSION}/atom-
↪amd64.tar.gz"
tar xfz atom-amd64.tar.gz
mv atom-*-amd64 _opt_atom
chown root:root _opt_atom -R
mkdir -p _opt_atom/bin
ln -s ../atom _opt_atom/bin/
ln -s ../resources/app/apm/bin/apm _opt_atom/bin/
rm atom-amd64.tar.gz

# Environment Setup
cp env.sh _opt_atom/.env.sh
cp adeinit _opt_atom/.adeinit

# Use _opt_atom for readability, rename to _opt b/c it's expected by Dockerfile
mv _opt_atom _opt
```

   - Usage: `./build-opt 1.35.0`
   - **This is where the bulk of the work happens:**
     - Using the version number provided, the script downloads Atom and installs it into a directory that will become `/opt/atom` in the Dockerfile:

[1] The code for this example is kept in this Gitlab project

```
COPY _opt /opt/atom
```

3. A script to set up the environment on `ade enter`:

```
export PATH="$PATH:/opt/atom/bin"
export ATOM_HOME="$HOME/.atom"
export BROWSER='chromium-browser --no-sandbox'
```

   - At a minimum, `env.sh` needs to add the executable to `PATH`
   - In this case it sets the location where Atom will store it's configuration such that they will persiste between restarts of ADE
   - Note that `build-opt` takes care of copying this file into `/opt/atom` and naming it `.env.sh`:

```
cp env.sh _opt_atom/.env.sh
```

4. A script of commands that should be run during `ade start`:

```
#!/usr/bin/env bash

if [[ ! -d "${ATOM_HOME}" ]]; then
    echo "Doing initial install of atom plugins:"
    /etc/atom/atom-install-our-plugins || true
    echo "Initial install of atom plugins done."
fi
```

   - Note that `build-opt` takes care of copying this file into `/opt/atom` and naming it `.adeinit`:

```
cp adeinit _opt_atom/.adeinit
```

   - Note also that `.adeinit` must be executable (`chmod +x .adeinit`)

Placing these four files in a Gitlab project and adding a CI script to automate the `docker build` command, makes it simple to generate new images on demand by creating a `git` tag for the desired version

For a more complex example, where the software is developed in the same repository, see the AutowareAuto project. AutowareAuto generates ADE volumes which can be used to deploy a pre-built AutowareAuto build in another machine.

3.3.2 Building the volume locally

Note: It is recommended to build the image as part of a CI script; nevertheless, the following instructions can be used to test the build steps locally. Make sure to run any setup scripts (e.g. see `build-opt` above) before building the image.

Build the image by running `docker build` in the directory that contains the `Dockerfile` and other necessary resources (e.g. the `_opt` directory generated by `build-opt` in the example above):

```
docker build -t <volume_image_name>:<tag> .
```

Add `--label ade_image_commit_sha=<git-hash>` and/or `--label ade_image_commit_tag=<git-tag>` to embed VCS information in the Docker image:

```
docker build \
  --label ade_image_commit_sha=<git-hash> \
```

(continues on next page)

(continued from previous page)

```
  --label ade_image_commit_tag=<git-tag> \
  -t <volume_image_name>:<tag> .
```

For more information, see the docker build documentation

3.3.3 Using the volume

To use the image, add it to the `.aderc` file **after** the base image entry in `ADE_IMAGES`. The order of the *volumes* does not matter as long as the base image is first:

```
export ADE_IMAGES="
   <base_image>:<tag>
   <volume_image_name>:<tag>
"
```

For more information, see *The .aderc File*

To setup ADE for a project, prepare two main components:

1. A **base-image** which is the Docker image that provides all the programs and dependencies for the project. See *Creating a custom base image* for details.
2. An `.aderc` file which specifies the Docker images to start and any additional `docker run` arguments. See *The .aderc File* for details.

In addition to a **base-image** and an `.aderc` file, it is recommended to use **volumes** to provide large, stand-alone programs and libraries. **Volumes** allow the project to update different components of ADE without requiring users to download an increasingly large image. See *Creating a custom volume* for details.

3.4 External Interfaces

3.4.1 Starting ADE with macvlan network configuration

In situations where ADE needs to communicate with external resources (e.g., sensors), it is possible to use a Macvlan network to assign the ADE container a MAC address, such that the container appears to be physically connected to the network interface. Macvlan gives the advantage of allowing the container to be isolated from the host while appearing as a physical device on the network.

To use Macvlan, follow the offical Docker documentation to create a `docker network` (e.g. `pub_net`), then run `ade start` with an extra Docker arguments: e.g

1. On the host create a macvlan network:

   ```
   $ docker network create -d macvlan --subnet 192.168.1.0/24 --gateway 192.168.1.1 -
   ↪o parent=eno1 pub_net
   ```

   Note The subnet and gatewar must match the local network settings

2. Start `ade` with `pub_net` and an unused IP address:

   ```
   $ ade start -- --net=pub_net --ip=192.168.1.101
   ```

3. To avoid typing the arguments every time, add the option to `ADE_DOCKER_RUN_ARGS` in the `.aderc`: e.g,

```
export ADE_DOCKER_RUN_ARGS="
  --cap-add=SYS_PTRACE
  --net=pub_net
  --ip=192.168.1.101
"
export ADE_GITLAB=gitlab.com
export ADE_REGISTRY=registry.gitlab.com
export ADE_IMAGES="
  registry.gitlab.com/autowareauto/autowareauto/ade:v0-1-0
  registry.gitlab.com/autowareauto/autowareauto:v0-1-0
  registry.gitlab.com/apexai/ade-atom:latest
"
```

3.4.2 Mounting ports to ADE

Use the `ADDARGS` arguments of `ade start`. See *Custom docker run/exec arguments* for details.

3.4.3 Mounting USB devices in ADE

To mount USB devices to ADE, they must be connected to the computer **before starting ADE**. Unfortunately, this requirements means that reconnecting a device requires a restart of ADE. USB devices are mounted by passing native Docker commands to `ade start`. e.g.:

```
$ ade start -- --device /dev/bus/usb:/dev/bus/usb
```

As described in *Configuring ADE with environment variables* and *The .aderc File*, use the `ADE_DOCKER_RUN_ARGS` variable to avoid typing the argument every time.

3.4.3.1 Troubleshooting USB devices

The device is available when ADE is started, but then it disappears Check that the device is connected to a stable power source. Some PCI card and USB hubs will not provide constant power, so device will disconnect intermittently

A temporary workaround for USB problems is to mount all USB devcies: **Warning**: This is not recommended as a permanent solution because it requires the `--privileged` flag, which gives Docker nearly all the same access to the host as processes running outside containers on the host (see the Docker documentation):

```
$ ade start -- --privileged -v /dev/bus/usb:/dev/bus/usb -v /dev/ttyUSB0:/dev/
↪ttyUSB0
```

3.4.4 NVIDIA Docker in ADE

Warning: Support for NVIDIA Docker is deprecated and will be dropped in the next release of `ade-cli`

NVIDIA Docker allows developers to leverage NVIDIA GPUs inside Docker containers.

`ade start` will automatically detect if NVIDIA docker is installed by checking if `/dev/nvidia0` exists, adding the necessary `docker run` arguments to make the GPUs available inside the container.

NVIDIA Docker has been deprecated as of Docker version 19.03, so `ade-cli` may drop support for NVIDIA Docker in future releases.

3.4.4.1 ADE - NVIDIA Docker Compatibility

The following table describes compatible versions of ADE, NVIDIA Docker, and the container's base image:

ADE Version > 3.4.1		**Container Base Image**	
		`ubuntu:xenial`	`ubuntu:bionic`
NVidia Docker Version	1	Supported	Supported
	2	Supported with a *workaround*	Supported

- **Note**: ADE Version <= 3.4.1 only supports NVIDIA Docker 1
- **Note**: ADE Version >= 4.1.0 does not require NVIDIA Docker, if Docker 19.03 or newer is installed

3.4.4.2 Troubleshooting ADE with NVIDIA Docker

1. ADE fails to start with the following message:

```
$ ade start
...
subprocess.CalledProcessError: Command 'curl -s http://localhost:3476/docker/cli'␣
↪returned non-zero exit status 7
```

- **This error usually means that you are trying to use NVIDIA Docker 2 with ADE Version <= 3.4.1**
  - **Fix:** Upgrade ADE or `export ADE_DISABLE_NVIDIA_DOCKER=1`
  - Note that if NVIDIA Docker is disabled, GUIs will not work

2. ADE starts successfully, but fails to open GUIs: e.g.

```
ade$ glxgears
libGL error: No matching fbConfigs or visuals found
libGL error: failed to load driver: swrast
```

- This error means that the base image does not have the correct libGL libraries available
- **The error could occur if `ADE_DISABLE_NVIDIA_DOCKER` is set before `ade start`**
  - **Fix:** Exit ADE, `unset ADE_DISABLE_NVIDIA_DOCKER`, and restart ADE (`ade start -f`)
- **More often, the error occurs when trying to use NVIDIA Docker 2 with an `ubuntu:xenial` image.**
  - See *NVIDIA Docker 2 with an ubuntu:xenial image* for details on how to work around this issue

NVIDIA Docker 2 with an `ubuntu:xenial` image

This section applies to developers who are trying to run ADE with the following versions:

- ADE > 3.4.1
- NVIDIA Docker 2
- `ubuntu:xenial`-based base image

A workaround is needed to get NVIDIA Docker 2 to work with an `ubuntu:xenial`-based Docker container because:

1. The libGL libraries that NVIDIA Docker 2 expects are different from the libGL libraries shipped with Ubuntu Xenial
2. NVIDIA Docker 1 ships the expected libraries as a volume, but NVIDIA Docker 2 does not

Therefore, the workaround is to load the expected libGL libraries into the Ubuntu Xenial container using an ADE volume. Add `registry.gitlab.com/apexai/ade-nvidia-cudagl:latest` to the `.aderc` configuration. e.g.

```
export ADE_IMAGES="
  xenial_base_image:latest
  registry.gitlab.com/apexai/ade-nvidia-cudagl:latest
"
```

For more details on the volume, see the ade-nvidia-cudagl project.

ADE can also be configured to use external interfaces (network devices, ports, and/or USB devices) to get, for example, information from sensors:

- *Starting ADE with macvlan network configuration* and *Mounting ports to ADE* explain how to configure the network interfaces of ADE
- *Mounting USB devices in ADE* explains how to make USB devices available inside ADE
- *NVIDIA Docker in ADE* explains how to integrate NVidia Docker with ADE

CHAPTER

FOUR

USAGE

The goal of this section is to explain how to use ADE on a project that provides an ADE configuration. See *Setup* for more information.

4.1 ADE Home

ADE needs a directory on the host which will be mounted as the user's home directory within the container. It will be populated with dotfiles and must be different than the user's home directory on the host. In case you use ADE for multiple projects it is recommended to use dedicated ADE home directories for each project.

ADE will look for a directory containing a file named `.adehome` starting with the current working directory and continuing with the parent directories to identify the ADE home directory to be mounted.

```
mkdir adehome
cd adehome
touch .adehome
```

4.2 Quick start

To start `ade`, an `.aderc` file is needed. For an example, see the Autoware.Auto project. For instructions to create an `.aderc` file for a project, see *Setup*.

When `ade` is run for the first time, it will first prompt for an authentication token, and then download all the Docker images configured in the `.aderc` file

```
$ cd adehome/path/to/.aderc
$ ade start --update
...
ADE startup completed.
$ ade enter
ade$ <= Note that the prompt changes
```

4.3 CLI

4.3.1 ade

ADE Development Environment.

```
ade [GLOBAL_OPTIONS] COMMAND [OPTIONS] -- [ARGS]...
```

Options

--version

--rc <rc>
Specify a different ADE configuration file. The file should be in adehome or its subdirectories. [default: .aderc]

--name <name>
Specify a custome name for the ADE instance [default: ade]

4.3.1.1 start

Start environment from within ADE project.

DOCKER_RUN_ARGS are passed directly to docker run. See https://docs.docker.com/engine/reference/commandline/run/ and *Custom docker run/exec arguments* for more information.

```
ade start [OPTIONS] -- DOCKER_RUN_ARGS
```

Options

--update, --no-update
Pull docker registries for updates. Using –update will imply –force.

--enter, --no-enter
Enter environment after starting.

-f, --force, --no-force
Force restart of running environment.

--select <select>
Select image tags to be used instead of configured defaults. Valid image tags are git tags and branches. To select one for a specific image use `IMAGE:TAG` (e.g. `ade:ftr123`) and only `TAG` to select it for all images for which it exists (e.g. `release-42`).

Arguments

-- DOCKER_RUN_ARGS
Optional argument(s)

4.3.1.2 enter

Enter environment, running optional command CMD.

DOCKER_EXEC_ARGS are passed directly to docker exec. See https://docs.docker.com/engine/reference/commandline/exec/ and *Custom docker run/exec arguments* for more information.

To enter a specific ADE instance, set the `ADE_NAME` environment variable or the global option `--name`. By default, `ade enter` will enter the `ade` environment.

If the specified environment is not running, select among the available environments from an interactive prompt.

```
ade enter [OPTIONS] [CMD] -- DOCKER_EXEC_ARGS
```

Options

-u, --user <user>
Enter ade as given user (e.g. root) instead of default

Arguments

CMD
Optional argument

-- DOCKER_EXEC_ARGS
Optional argument(s)

Environment variables

ADE_ENTER_CMD

Provide a default for *CMD*

4.3.1.3 stop

Stop ade environment.

```
ade stop [OPTIONS]
```

4.3.1.4 save

Save configuration and images of running ADE into DIRECTORY.

```
ade save [OPTIONS] DIRECTORY
```

Arguments

DIRECTORY
Required argument

4.3.1.5 load

Load images from DIRECTORY.

```
ade load [OPTIONS] DIRECTORY
```

Arguments

DIRECTORY
Required argument

4.3.1.6 update-cli

Update ade command-line interface.

```
ade update-cli [OPTIONS]
```

Options

--finish-update <finish_update>
Used internally by update mechanism

4.4 Environment Variables

4.4.1 Custom `docker run/exec` arguments

`ade` encapsulates most of the options needed to start and run the Docker container with the correct `docker run` options. However, a power user of Docker may find certain options missing. `ade start` allows users to pass additional arguments to `docker run`.

To pass options, separate them with `--` from the preceding `ade` options. For example, use the following command to mount port 1234 into `ade`:

```
ade start -- --publish 127.0.0.1:1234:1234/udp
```

Similarly, `ade enter` allows users to pass additional arguments to `docker exec`:

```
ade enter -- --env MY_DYNAMIC_VAR
```

See https://docs.docker.com/engine/reference/commandline/run/ for more information on `docker run` and https://docs.docker.com/engine/reference/commandline/exec/ for more information on `docker exec`.

See *Starting ADE with macvlan network configuration* and *Mounting USB devices in ADE* for other examples of `DOCKER_RUN_ARGS`.

See *The .aderc File* for ways to make these options permanent for a project.

4.4.1.1 Starting ADE with `--net=host`

The Docker option `--net=host` configures the Docker container (and by extension the ADE instance) to share its network stack with the host. This option is useful when trying to debug networking applications; however, the option has been known to cause issues with certain UI applications such as `glxgears` and `rviz`. To get the UI applications running, use `--privileged` in addition to `--net=host`: e.g

```
ade start -- --net=host --privileged
```

Warning: Using `--privileged` is dangerous since it gives Docker nearly all the same access to the host as processes running outside containers on the host (see the Docker documentation). If it is necessary for the ADE instance to appear as a device on the host's network, try starting ADE with Macvlan instead (see *Starting ADE with macvlan network configuration*).

4.4.2 Configuring ADE with environment variables

For every `ade` command's arguments, there is a corresponding environment variable that can be used to configure each argument's default behavior. The environment variables follow the following pattern `ADE_[ADE_COMMAND]_[OPTION_OR_ARGUMENT]`.

In addition to the command-specific variables, the `ADE_NAME` can be used to change the name of the ADE instance. This variable is especially useful when *Starting multiple ADE instances* at once.

For example, the default behavior of `ade start` is just to start the container, but often, it is desired to also enter the container directly after starting, so `ade start` has the `--enter` option:

```
$ ade start --enter
ade$ # Inside ADE prompt
```

It is possible to set the default behavior of `ade start` by setting the `ADE_START_ENTER` variable:

```
$ export ADE_START_ENTER=true
$ ade start
ade$ # Inside ADE prompt
```

To make the configuration permanent, add the `export` to:

- The `~/.bashrc` file on the host, for personal configurations
- The `.aderc` file on the project, for project-wide configuration, see *The .aderc File*

It is also possible to configure the default behavior of `ade` using environment variables and the `.aderc` file. For more information, see *Configuring ADE with environment variables* and *The .aderc File*.

4.5 Option Precedence

The value of each `ade` option is taken from the following sources in order of highest to lowest precedence:

1. Command line options (*CLI*)
2. *Environment Variables*
3. Values in *The .aderc File*
4. Default values (if any)

For example::

```
$ export ADE_RC=/path/to/rc
$ ade --rc /path/to/other_rc start  # --rc overrides ADE_RC
$ export ADE_NAME="ade2"
$ ade start  # starts 'ade2' with configurations from /path/to/rc
$ ade --rc /path/to/other_rc start  # starts 'ade2' with configurations from /path/to/
↪other_rc
```

Note that in the last command the `ADE_NAME` environment variable takes precedence over the `ADE_NAME` specified in the `aderc` file, because even though the `aderc` file is specified via the command line, the `--rc` option only becomes relevant after checking options *directly* specified in command line options or via an environemnt variables.

For example::

```
$ ADE_NAME='envvar' ade --name 'ade_opt' --rc /path/to/rc start  # starts 'ade_opt'
$ ADE_NAME='envvar' ade --rc /path/to/rc start  # starts 'envvar'
$ ade --rc /path/to/rc start  # starts 'ADE_NAME' from '/path/to/rc'
$ ade start  # starts 'ade'
```

Note that *arguments* to the ADE commands are additive. Specifically, arguments passed to `ade start` will be appended to the values in `ADE_DOCKER_RUN_ARGS`, and arguments passed to `ade enter` will be appended to the values in `ADE_DOCKER_EXEC_ARGS`.

4.6 Cleanup

Over time, unused Docker images, containers, and volumes will clutter the machine's hard drive. ADE does not store anything valuable inside the Docker containers, so it is possible to use native Docker commands to clean up. Anything valuable that needs to persist should be placed in `ade-home`, which is stored on the host and mounted in ADE.

To get an overview of Docker's disk usage, run:

```
$ docker system df
TYPE                TOTAL               ACTIVE              SIZE                ␣
↪RECLAIMABLE
Images              13                  11                  14.03GB             916.
↪9MB (6%)
Containers          11                  0                   2.311MB             2.
↪311MB (100%)
Local Volumes       17                  15                  5.411GB             17.
↪8MB (0%)
Build Cache         0                   0                   0B                  0B
```

Docker provides a cleanup command that will remove everything that is unused. To properly determine what is unused, **make sure that all Docker containers you want to keep are running.** To avoid having to re-download ADE images, run `ade start`. Once you are certain that everything you want to keep is in use, run:

```
$ docker system prune -a --volumes
```

4.7 Debugging

To see the native Docker commands and other commands *ade* is executing, set the `ECHO` variable before running *ade*, e.g.

```
$ ECHO=1 ade start
```

4.8 Starting multiple ADE instances

When working on multiple projects with ADE configurations, it is desired to keep an instance of ADE running for each project. To run multiple instance, use the `ADE_NAME` environment variable to switch between the two instances.

Let's say that there are two projects minimal-ade and AutowareAuto, cloned in `ade-home`:

Note It is recommended to keep separate ADE homes for each project; however, nothing prevents using the same ADE home for two projects.

Start the ADE instance for *minimal-ade*:

```
$ cd ~/ade-home/minimal-ade
$ export ADE_NAME=minimal
$ ade start
```

Then start the ADE instance for *AutowareAuto*:

```
$ cd ~/ade-home/AutowareAuto
$ export ADE_NAME=autoware
$ ade start
```

Now, in a new terminal, it is possible to select the ADE instance by setting `ADE_NAME`:

```
$ cd ~/ade-home
$ export ADE_NAME=minimal
$ ade enter
minimal$ # minimal ADE prompt
```

If `ADE_NAME` is not defined, the default name `ade` is used for a new environment.

In case the specified ADE is not found, `ade enter` offers an interactive prompt to select among the running environments. Press `Enter` to select the environment offered in brackets, or type `minimal` and press `Enter` to choose the other option:

```
$ cd ~/ade-home
$ ade enter
ERROR: An ADE instance named ade is not running.
Select one of {'autoware, minimal'} [autoware]:
Set ADE_NAME to suppress this prompt.
Entering autoware with following images:
...
```

Note The host name in the ADE container will match `ADE_NAME`, indicating the current ADE instance

Similarly, specific ADE instances can be stopped:

```
$ cd ~/ade-home
$ export ADE_NAME=autoware
$ ade stop
Stopping autoware
...
```

4.8.1 Limitations

Starting multiple ADE works well when using the default network configuration; however, ADE instances may conflict if they use custom network configurations:

- If one ADE instance binds to a port (see *Custom docker run/exec arguments*), a second instance will not be able to bind to the same port.
- If one instance uses a Macvlan network (see *Starting ADE with macvlan network configuration*), a second Macvlan network must be created for the second instance.

CHAPTER

FIVE

CHANGELOG

This project follows SemVer.

5.1 4.4.0 (2021-12-02)

5.1.1 Added

- Allow running image without registry namespace in `.aderc` file
- Allow passing additional arguments to ade enter (docker exec, configured in `.aderc` file)
- Propagate COLORTERM environment variable into the docker environment
- Add global option `--name` to set a custom name for the ADE instance
- Document precedence of ADE options

5.1.2 Changed

- Use the host IP as the value of `DISPLAY` in the container
- Improved ADE usage help text
- Fail if the specified `aderc` file does not exist

5.1.3 Removed

- Removed default mount of `/dev/shm` of host into ade

5.1.4 Fixed

- Update link to Gitlab's personal access token page
- Typos in the documentation

5.1.5 Security

- Changed recommended token scope to `read_api` and `read_registry`

5.2 4.3.0 (2021-06-11)

5.2.1 Added

- Support images from Dockerhub
- Add `ade_base_image` label when starting ADE
- Prompt when an ADE instance is running, but ADE_NAME is not set

5.2.2 Changed

- Allow using images with the same base name

5.3 4.2.0 (2020-06-03)

5.3.1 Added

- Support images with only namespace
- Support update of images with only namespace
- Support adding custom `ade enter` command through environment variable
- Experimental support for OSX
- Add option to specify a different ADE configuration file

5.3.2 Changed

- Start ADE volumes without a network connection
- Reprompt for token when authentication fails with existing token

5.3.3 Deprecated

- Support for `nvidia-docker` and `nvidia-docker2` is deprecated and will be removed in the next release

5.3.4 Fixed

- Support Docker versions ending in -ce
- Support images with only namespace
- Fix `ade start --enter` for devices with broken `docker exec`
- Suppress misleading error message on startup
- Update images chosen with 'select' option
- Add note in documentation about making adeinit scripts executable
- Fix error message that described how to debug a failed `ade start`

5.4 4.1.0 (2020-01-26)

5.4.1 Added

- Add support for `--gpus` option in Docker 19.03+

5.4.2 Fixed

- Support images from registries with explicit port specification
- Fix nvidia-docker 1 support

5.5 4.0.0 (2019-11-18)

5.5.1 Added

- Enable self-update from ADE releases on GitLab
- Support saving and loading images to run on offline machines

5.5.2 Changed

- Switch to single-binary as default means of distribution, using pyinstaller and staticx

5.5.3 Removed

- Drop support for Python 3.5 and 3.6, instead Python 3.7 is required for development

5.6 3.5.1 (2019-09-26)

5.6.1 Added

- Make host location of ade-home known inside container #39
- Document usage of ADE with the fork+pull model #19
- Document how to build custom base images and volumes #50
- Document how to run ADE with `--net=host` #33

5.6.2 Fixed

- Let ade enter return with exit code of shell running within container #46
- Ask for API scope credentials when prompting for Gitlab API token #42
- Update instructions to address aarch64 installation issues #23

5.7 3.5.0 (2019-08-08)

5.7.1 Added

- Add note about starting ADE with –net=host
- Add support for NVidia Docker 2 #31
- Add initial set of documentation
- Extend `ade start --help` to include information on publishing ports

5.8 3.4.1 (2018-11-16)

5.8.1 Changed

- Pinned aiohttp<3.5.0 for compatibility with Python 3.5.2

5.9 3.4.0 (2018-11-16)

5.9.1 Changed

- Print n/a in image matrix for images that have not been pulled

5.10 3.3.1 (2018-11-15)

5.10.1 Added

- Add environment variables to introspect loaded image versions
- Print git commit or tag info in image matrix
- Add CI job for release validation and publication

5.11 3.2.0 (2018-11-14)

5.11.1 Added

- Add `--version` option
- Add nvidia-docker support
- Add workaround for broken `docker exec`
- Add tests and initial CI jobs
- Add ADE_CLI_VERSION environment variables
- Add infrastructure to add requirements for development on ade-cli

5.11.2 Changed

- Renamed `--check` to `--update`
- Pin to release Click 7.0
- Code cleanup

5.12 3.1.0 (2018-10-10)

5.12.1 Added

- Support setting docker run args via configuration

5.12.2 Fixed

- Fix passing docker run arguments to ade start

5.13 3.0.1 (2018-09-11)

5.13.1 Fixed

- Fix homepage and project URLs

5.14 3.0.0 (2018-09-04)

5.14.1 Added

- Initial public release of ADE

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

C

D