



Virtualized Hardware Lab

User Guide

ADAS ESMINI AMI User Guide v1.0

Table of Content

Table of Content

○ Table of Content.....	1
○ Table of Figures.....	4
Document Information	
○ 1.1. Abbreviations and Definitions.....	5
○ 1.2. Purpose.....	6
Overview	
○ 2.1. Introduction.....	7
○ 2.2. Solution Benefits.....	8
Solution Prerequisites	
○ 3.1. Instance Requirements.....	9
EC2 Creation Steps	
○ 4.1. Recommended EC2 Instances.....	9
○ 4.2. IAM Role Creation.....	10
○ 4.3. Base Image Selection.....	11
○ 4.4. Configuration Process.....	11
○ 4.5. Instance Launch.....	15
○ 4.6. SSM Connection.....	15
○ 4.7. AWS SSO Authentication.....	16
■ 4.7.1. Step 1: Install Required Tools.....	16
■ 4.7.2. Step 2: One-Time SSO Configuration.....	16
■ 4.7.3. Step 3: Daily Usage.....	17
○ 4.8. NiceDCV Connection.....	18
Solution's Content	
○ 5.1. User Workspace.....	22
■ 5.1.1. Scripts.....	22
■ 5.1.2. Simulation Configuration File.....	23
■ 5.1.3. Virtualized_Lab.....	23
■ 5.1.4. Templates and Libraries.....	24
● 5.1.4.1. ASAMScenarios Directory.....	24
● 5.1.4.2. L4-VECU-templates Directory.....	25
■ 5.1.5. ADAS Application.....	26
● 5.1.5.1. Application Structure.....	26
● 5.1.5.2. Available APIs.....	26
● 5.1.5.3. Valeo Examples.....	35
○ 1. ADAS_TIMESTAMP_001.....	36
○ 2. ADAS_LIVECNT_001.....	37

Table of Content

○ 3. ADAS_ESMINI_STATIC_001_12_SENSORS.....	37
○ 4. ADAS_ESMINI_STATIC_001_8_SENSORS.....	38
○ 5. ADAS_ESMINI_DYNAMIC_001_12_SENSORS.....	39
○ 6. ADAS_ESMINI_DYNAMIC_001_8_SENSORS.....	40
○ 5.2. HW-Virtualization-lab.....	41
■ 5.2.1. Silkit.....	41
■ 5.2.2. simulation_tools.....	42
■ 5.2.3. L4_vECU.....	42
■ 5.2.4. HW Lab Configuration Files.....	42
■ 5.2.5. Ultrasonics SM FMU.....	44
■ 5.2.6. Ultrasonics SM FMU Configuration File.....	44

Environment Setup

○ 6.1. Virtualized Lab GUI.....	49
■ 6.1.1. Accessing the Virtualized Lab.....	49
■ 6.1.2. Virtualized Lab Main Window.....	50
● 6.1.2.1. Interactive L4 Terminal.....	50
● 6.1.2.2. Log Viewer.....	50
● 6.1.2.3. Button Panel.....	50
■ 6.1.3. Virtualized Lab Usage Workflow.....	51
■ 6.1.3.1. Valeo ADAS App Workflow.....	51
■ 6.1.3.2. User ADAS App Workflow.....	53
○ 6.2. Manual Setup.....	55
■ 6.2.1. Environment Initialization.....	55
● 6.2.1.1. Initialization Setup Process.....	55
● 6.2.1.2. Environment Verification.....	56
■ 6.2.2. Environment Running.....	56
● 6.2.2.1. Running Setup Process.....	56
● 6.2.2.2. ADAS Application Procedures.....	57

Simulation Tool

○ 7.1. Pre-requisite.....	60
○ 7.2. Integration Process.....	60
○ 7.3. ESMINI Simulation Tool.....	66
■ 7.3.1. Tool Overview.....	66
■ 7.3.2. Pre-Simulation Setup.....	67
■ 7.3.3. Running Options.....	67
● 7.3.3.1. Running Using GUI.....	67
● 7.3.3.2. Running Using Execution Commands.....	67
■ 7.3.4. Running Components.....	68

Table of Content

- 7.3.5. Managing Scenarios..... 69
 - 7.3.5.1. Using an Existing Scenario.....69
 - 7.3.5.2. Creating a Custom Scenario..... 69
- 7.4. Integration With External Tool.....70
 - 7.4.1. Pre-requisites..... 70
 - 7.4.2. Integrated Environment Initialization.....70
 - 7.4.3. Integrated Environment Running.....71

Troubleshooting Guide

- 8.1. AMI Issues.....73
 - 8.1.1. Describe Instance.....73
 - 8.1.1.1. Issue Description..... 73
 - 8.1.1.2. Resolution Steps.....73
 - 8.1.2. Metadata Configuration..... 74
 - 8.1.2.1. Issue Description..... 74
 - 8.1.2.2. Resolution Steps..... 74
- 8.2. AWS SSO Issues..... 74
 - 8.2.1. Authentication Issues.....74
 - 8.2.1.1. Issue Description..... 74
 - 8.2.1.2. Resolution Steps..... 75
- 8.3. Troubleshooting via Execution Commands.....76
 - 8.3.1. Init Troubleshooting.....76
 - 8.3.1.1. Common Issues..... 76
 - 8.3.1.2. Resolution Steps..... 76
 - 8.3.2. Valeo's vECU Troubleshooting..... 77
 - 8.3.2.1. Access Issues..... 77
 - 8.3.2.2. Linking Errors..... 77
 - 8.3.3. Running Troubleshooting..... 77
 - 8.3.3.1. Runtime Issues..... 78
 - 8.3.3.2. Recovery Procedures..... 78
- 8.4. Troubleshooting via Virtualized Lab.....78
 - 8.4.1. Run Simulation Troubleshooting.....78
 - 8.4.1.1. Common Issues..... 78
 - 8.4.1.2. Resolution Steps..... 79

Open-Source Tools and License

Table of Figures

○ Figure [1]: Solution Environment.....	7
○ Figure [2]: Instance Type Selection.....	12
○ Figure [3]: Network Configuration.....	12
○ Figure [4]: Storage Configuration.....	13
○ Figure [5]: IAM Role Attachment to EC2.....	13
○ Figure [6]: MetaData Configuration.....	14
○ Figure [7]: Instance Launching.....	15
○ Figure [8]: SSM Session Port Forwarding.....	18
○ Figure [9]: NiceDCV Security Page.....	19
○ Figure [10]: NiceDCV Login.....	20
○ Figure [11]: Virtualized HW Lab Package.....	21
○ Figure [12]: Virtualized HW Lab Package Content.....	21
○ Figure [13]: User Workspace Content.....	22
○ Figure [14]: Templates and Libraries Content.....	24
○ Figure [15]: ASAM Scenarios Content.....	24
○ Figure [16]: L4-VECU-templates Content.....	25
○ Figure [17]: ADAS APP Template Content.....	26
○ Figure [18]: HW Virtualization lab Content.....	41
○ Figure [19]: Virtualized HW Lab Desktop Icon.....	49
○ Figure [20]: Virtualized HW Lab.....	50
○ Figure [21]: Configuration Window For Valeo ADAS App.....	51
○ Figure [22]: Run Test Window For Valeo ADAS App.....	52
○ Figure [23]: Configuration Window for User's ADAS App.....	53
○ Figure [24]: Transfer Window.....	54
○ Figure [25]: External Simulation Tool Environment.....	65
○ Figure [26]: ESMINI Simulator.....	66
○ Figure [27]: Solution Environment With ESMINI.....	67
○ Figure [28]: Silkit Communication With ESMINI.....	68
○ Figure [29]: Describe Instance Issue.....	73
○ Figure [30]: Metadata configuration Issue.....	74
○ Figure [31]: Troubleshooting Runs in the Virtualized Lab.....	79

1. Document Information

1.1. Abbreviations and Definitions

Abbreviation	Description
AMI	Amazon Machine Image
EC2	Elastic Compute Cloud
RDP	Remote Desktop Protocol
VM	Virtual Machine
ADAS	Advanced Driver-Assistance System
vECU	Virtual Electronic Control Unit
DFT	Design For Test
SM	Sensor Model
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Units

Abbreviation	Description
ASAM	Association for Standardisation of Automation and Measuring Systems

1.2. Purpose

The primary purpose of this user guide is to provide comprehensive documentation for utilizing the Valeo Virtualized HW Lab, specifically focusing on the Valeo ADAS AMI. This guide serves to:

- Explain the system architecture and component interactions
- Guide users through the setup and configuration process
- Provide troubleshooting assistance and best practices
- Guide the integration capabilities with various simulation tools

2. Overview

2.1. Introduction

The Valeo Virtualized HW Lab is a cloud-based solution designed to facilitate the implementation, testing, and validation of ADAS software algorithms. This virtual environment provides a pre-configured vECU integrated with Valeo's ultrasonic sensor model, allowing software developers to conduct thorough work in a cloud-based setting.

As illustrated in the architecture diagram in [Figure \[1\]](#), the solution comprises several interconnected components, including a Valeo sensor model with an L3 vECU, a simulation tool, and SilKit for communication.

The solution is available as an AMI on the AWS marketplace; this setup allows users to integrate the Valeo ultrasonic sensor model with their preferred simulation tool for defining the required testing scenarios.

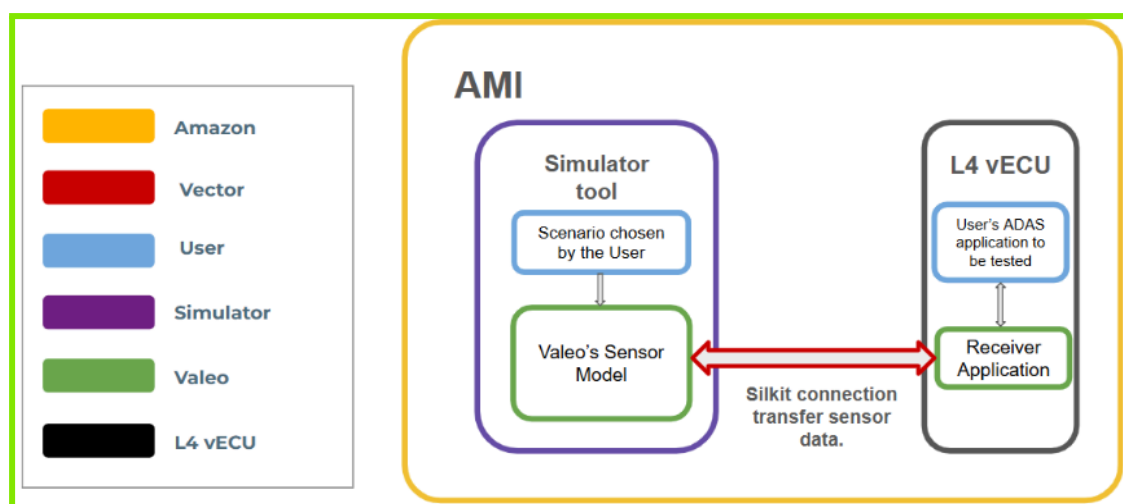


Figure [1]: Solution Environment

2.2. Solution Benefits

The Valeo Virtualized HW Lab delivers advantages for ADAS software development:

- **Accelerated Development Cycle**
Speed up software development by up to 40% using virtual ECUs for immediate testing
- **Reduce Hardware Cost**
Lower R&D hardware expenses by 30% by eliminating the need for physical ECUs and sensors
- **Shift-Left Development Approach**
Identify and resolve issues earlier in the development cycle for more reliable software, faster feedback and lowering fixing costs
- **Unlimited Availability for Development and Testing**
24/7 access to development and testing environments. No hardware procurement delays. Continuous development and testing capabilities
- **Cloud-Based Scalability**
Utilize a flexible testing platform that adapts to your development needs on the cloud. Availability 24/7 from anywhere in the world

3. Solution Prerequisites

3.1. Instance Requirements

1. Active AWS Account with appropriate permissions, Sign into [AWS](#).
2. Familiarity with the [AWS EC2 user guides](#).
3. The organization has provided access to its AWS environment and resources.
4. All attached volumes must be encrypted
5. If Windows is in use, install the Windows Subsystem for Linux (WSL) to connect to the instance's graphical user interface.

4. EC2 Creation Steps

4.1. Recommended EC2 Instances

The [EC2 instance](#) recommended and tested for this solution is:

Instance Type	Supported Simulation Tool	Architecture
t3.xlarge	ESMINI v2.46.0	Linux x86-64
t4g.xlarge	ESMINI v2.46.0	Linux arm-64

4.2. IAM Role Creation

The Valeo solution requires specific roles to be associated with the instance. Reference the [AWS IAM Role documentation](#) to understand how to create this new custom role, which must include the following policies:

- [AmazonS3FullAccess](#)
- [AmazonS3ReadOnlyAccess](#)
- [AmazonSSMManagedInstanceCore](#)
- **Custom Policy 1 (Required for GUI): NiceDCV License Access** This policy allows the instance to retrieve the necessary license information from Amazon's S3. *This License can be found in [Amazon License Page](#).*
- **Custom Policy 2 (Required): Service Description Access** This policy grants permissions to describe necessary EC2 and License Manager resources. Failure to address this will result in a violation.

None

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeVolumes",
        "license-manager:ListLicenses"
      ],
      "Resource": "*"
    }
  ]
}
```

- Failing to correctly attach this policy will prevent the instance from running:

- It will automatically terminate about **4 minutes without launch** to prevent this **violation**
- It will automatically terminate about **30 seconds after launch** to prevent this **violation**.

4.3. Base Image Selection

The initial step in creating the environment is selecting the appropriate Valeo-provided AMI.

1. **Access the EC2 Console:** Navigate to the [Amazon EC2 console](#) and select "Instances."
2. **Initiate Launch:** Under Application and OS Images (Amazon Machine Image), search for Valeo, choose AWS Marketplace AMIs, then choose the desired Valeo AMI. This is the operating system (OS) image for the selected instance.
3. **Locate Valeo AMI:** The available AMIs are tailored for different simulation tools and architectures, Search for and select the specific Valeo AMI required for your project.

4.4. Configuration Process

Once the base AMI is selected, it must be configured according to the following specifications to ensure proper functionality.

1. **Instance Type Selection (Required):** Refer to [Figure \[2\]](#) and [§4.1 Recommended EC2 Instances](#) to select a recommended instance type for the chosen AMI.

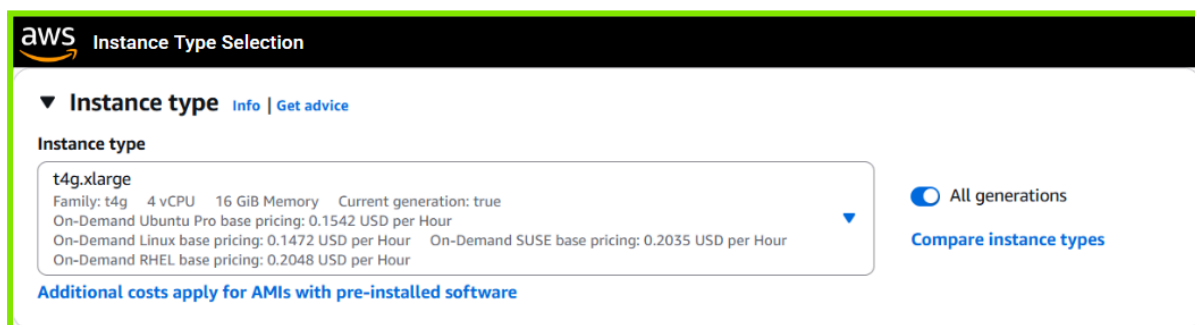
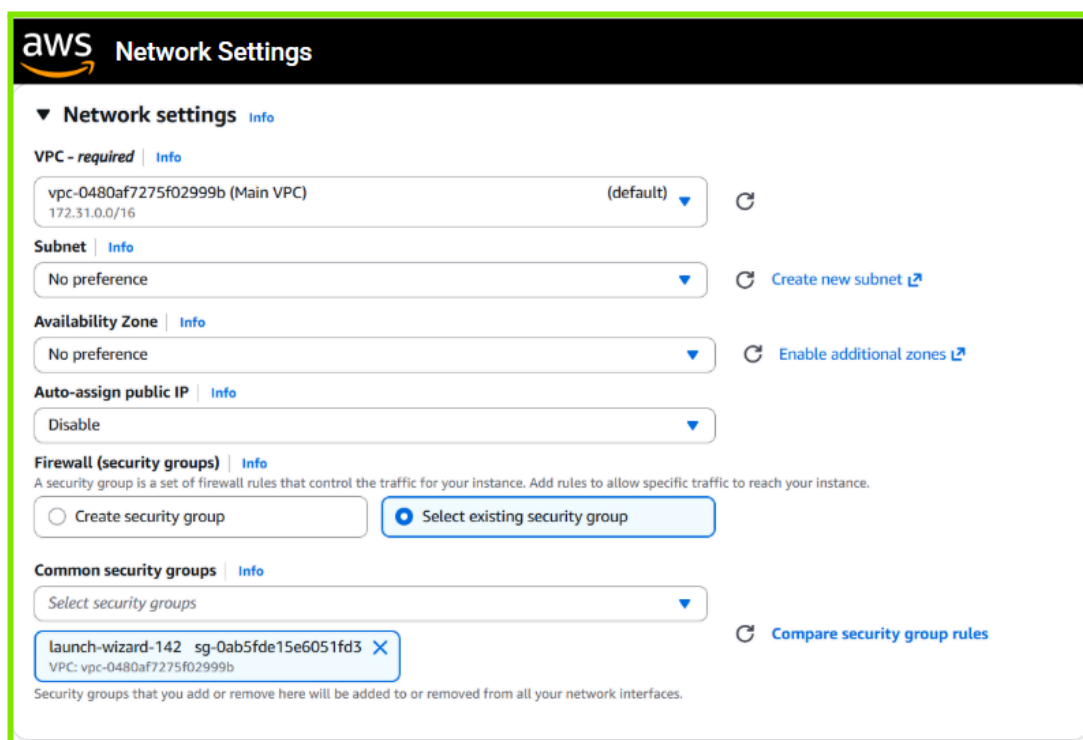


Figure [2]: Instance Type Selection

2. **Network Configuration:** As shown in [Figure \[3\]](#), Adjust network settings for the environment from the Network menu as follow:
 - Auto-assign public IP: **Disabled**
 - Select existing security group
 - Select the following security group: **launch-wizard-142 Sg-0ab5fde15e6051fd3**



The screenshot shows the AWS Network Settings page. The 'Network settings' section is expanded, showing the following configurations:

- VPC:** vpc-0480af7275f02999b (Main VPC) (default)
- Subnet:** No preference
- Availability Zone:** No preference
- Auto-assign public IP:** Disable
- Firewall (security groups):** Select existing security group (radio button selected)
- Common security groups:** launch-wizard-142 sg-0ab5fde15e6051fd3

Buttons for 'Create new subnet', 'Enable additional zones', and 'Compare security group rules' are visible on the right side of the configuration area.

Figure [3]: Network Configuration

3. **Storage Configuration (Required):** Search for and select the specific Valeo AMI required for your project. The available AMIs are tailored for different simulation tools and architectures:
 - Configure the root volume with a minimum of 100G of storage and the Type to be **gp3** as shown in [Figure \[4\]](#).
 - It is a mandatory requirement that all volumes attached to the instance must be Encrypted. Refer to [§3.1 Instance Requirements](#)

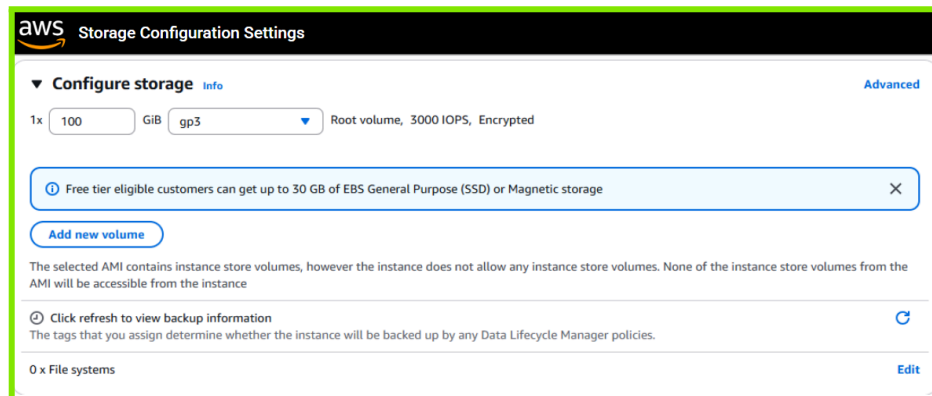


Figure [4]: Storage Configuration

4. IAM Role Attachment to EC2 (Required):

- In the "Advanced details" section, select the created role in [§4.2 IAM Role Creation](#) as illustrated in [Figure \[5\]](#)
- If this role is omitted during the initial configuration, it can be subsequently attached, refer to [AWS Attach Role to EC2 instance](#) for guidance.

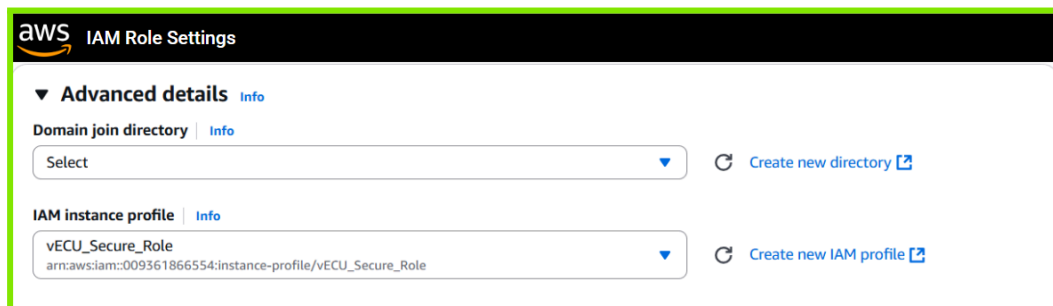
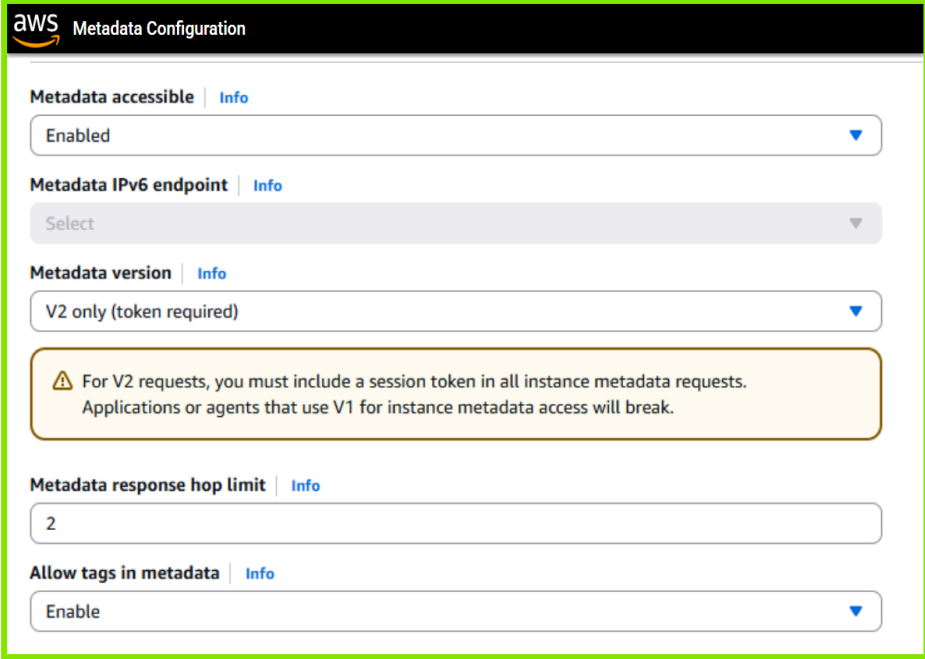


Figure [5]: IAM Role Attachment to EC2

5. Metadata Configuration (Required): In the "Advanced details" section, apply the following mandatory settings as shown in [Figure \[6\]](#):

- **Metadata accessible:** Set to "Enabled".
- **Metadata version:** Set to "V2 only (token required)".

- **Metadata response hop limit:** Set to "2".



The screenshot shows the AWS Metadata Configuration console. The settings are as follows:

- Metadata accessible:** Enabled
- Metadata IPv6 endpoint:** Select
- Metadata version:** V2 only (token required)
- Metadata response hop limit:** 2
- Allow tags in metadata:** Enable

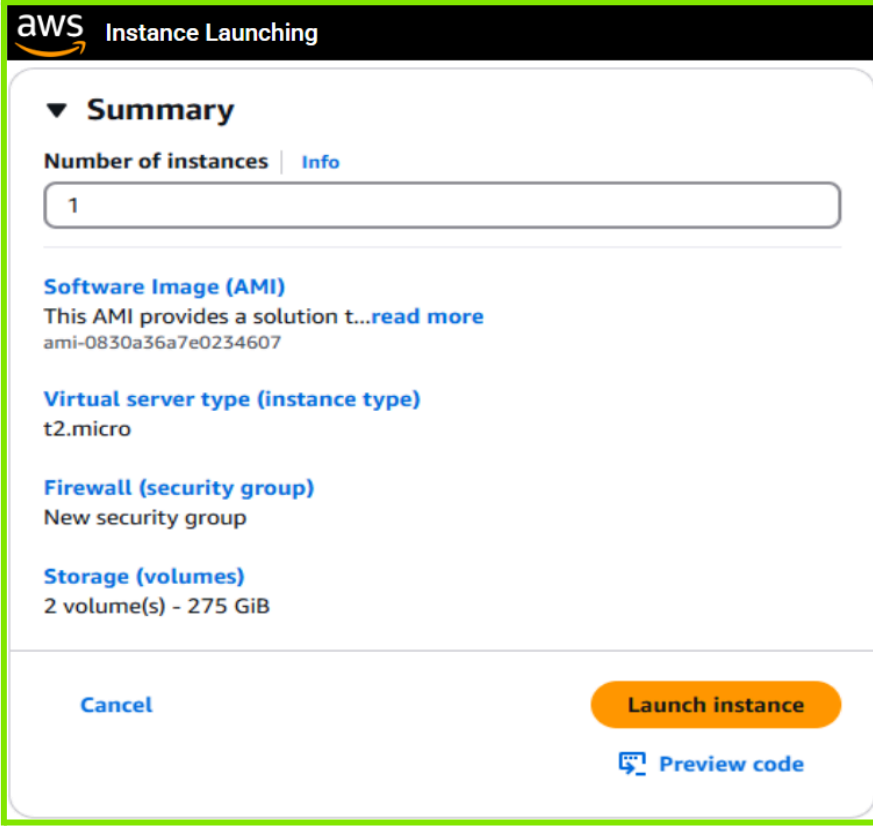
A warning message is displayed: "For V2 requests, you must include a session token in all instance metadata requests. Applications or agents that use V1 for instance metadata access will break."

Figure [6]: MetaData Configuration

- **Note:** This configuration can also be modified after the instance has been created refer to [AWS Modify Instance Metadata](#)
- Failing to correctly configure the metadata options will prevent the instance from running.
 - It will automatically terminate about **4 minutes without launch** to prevent this **violation**.
 - It will automatically terminate about **30 seconds after launch** to prevent this **violation**.

4.5. Instance Launch

Upon finalizing the desired settings, specify the required number of instances and initiate the launch by clicking the "Launch Instance" button, as depicted in [Figure \[7\]](#).



The screenshot shows the AWS Management Console 'Instance Launching' page. The 'Summary' tab is selected, displaying configuration details for a new instance. The 'Number of instances' is set to 1. The 'Software Image (AMI)' is 'ami-0830a36a7e0234607'. The 'Virtual server type (instance type)' is 't2.micro'. The 'Firewall (security group)' is 'New security group'. The 'Storage (volumes)' section shows '2 volume(s) - 275 GiB'. At the bottom, there are 'Cancel' and 'Launch instance' buttons, along with a 'Preview code' link.

Field	Value
Number of instances	1
Software Image (AMI)	ami-0830a36a7e0234607
Virtual server type (instance type)	t2.micro
Firewall (security group)	New security group
Storage (volumes)	2 volume(s) - 275 GiB

Figure [7]: Instance Launching

4.6. SSM Connection

1. Refer to the following document for SSM configuration and connection: [AWS SSM Connection Document](#)
2. Refer to [AWS Configure Session Preferences](#) to update the default user and general preferences run as **ubuntu**

4.7. AWS SSO Authentication

Accessing AWS resources requires configuring the local machine for Single Sign-On (SSO). This is a one-time setup.

4.7.1. Step 1: Install Required Tools

Two command-line tools from AWS are required. If these tools are already installed, ensure the AWS CLI is at least version 2.

1. **AWS CLI v2:** The AWS Command Line Interface is essential. Version 2 is required for SSO.
 - **To Install or Upgrade:** Follow the official instructions for the relevant operating system: Installing the AWS CLI.
 - **To Verify Installation:** Open a new terminal and run **aws --version**. The output should start with aws-cli/2.x.x.
2. **Session Manager Plugin:** This plugin is needed to connect to instances via the command line.
 - **To Install:** Follow the official guide for the relevant operating system: Installing the Session Manager Plugin.

4.7.2. Step 2: One-Time SSO Configuration

This step links the AWS CLI to the SSO account. It only needs to be done once.

1. Run the following command in a terminal:

```
Shell  
aws configure sso
```

2. Follow the interactive prompts. The prompts will ask for:
 - **SSO session name:** Press Enter to accept the default.
 - **SSO start URL:** Enter the URL for the company's AWS access portal.
 - **SSO region:** Enter the AWS Region for SSO, likely eu-west-1.

3. A browser will open for login. If it fails to open (e.g., in WSL), copy the URL from the terminal and paste it into a browser manually.
4. After logging in, return to the terminal. The prompts will continue for selecting:
 - The AWS Account to access.
 - The CORP-Administrators role.
 - **CLI Profile Name:** Crucially, replace the long default name with corp-admin. This short name will simplify future commands.
 - **Default Region:** Enter the primary working region (e.g., eu-west-1).
 - **Default output format:** Press Enter to accept json.
5. The SSO profile is now configured and ready for use.

4.7.3. Step 3: Daily Usage

Log In (Once per day): At the start of the workday, or if a session expires, run this single command. It will refresh credentials securely via the browser.

```
Shell  
aws sso login --profile corp-admin
```

4.8. NiceDCV Connection

To connect to the instance's graphical user interface, follow these procedures:

1. **IAM Role Attachment on AWS:** Ensure the "**Custom Policy 1**" attached to the EC2 instance's IAM role aligns with the requirements specified in [\[§4.2 IAM Role Creation\]](#).
2. **Session Preferences on AWS:** Make sure the default user for the session is **ubuntu** as specified in [§4.7 SSM Connection](#)

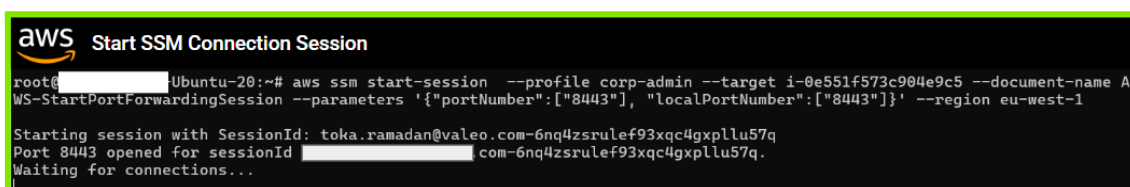
3. **Secure AWS SSO Authentication:** Make sure to achieve the steps specified in [§4.8 AWS SSO Authentication](#)
4. **Start SSM Connection Session:** If Windows is in use, open the WSL (Windows Subsystem for Linux) terminal. If on macOS or another Linux distribution, open the standard terminal application. And run the following command:

Shell

```
aws ssm start-session --profile <CLI Profile Name> --target INSTANCE_ID  
--document-name AWS-StartPortForwardingSession --parameters  
'{"portNumber":[8443], "localPortNumber":[8443]}' --region eu-west-1
```

And replace **INSTANCE_ID** with the actual EC2 Instance ID refer to [AWS Finding Instance ID](#) and replace the **<CLI Profile Name>** with the actual profile name created in [§4.8 AWS SSO Authentication](#)

It is expected to have something like what illustrated in [Figure \[8\]](#):



```
aws Start SSM Connection Session  
root@ [redacted] Ubuntu-20:~# aws ssm start-session --profile corp-admin --target i-0e551f573c904e9c5 --document-name A  
WS-StartPortForwardingSession --parameters '{"portNumber":["8443"], "localPortNumber":["8443"]}' --region eu-west-1  
Starting session with SessionId: toka.ramadan@valeo.com-6nq4zsrulef93xqc4gxpllu57q  
Port 8443 opened for sessionId [redacted] com-6nq4zsrulef93xqc4gxpllu57q.  
Waiting for connections...
```

Figure [8]: SSM Session Port Forwarding

5. **Web Browser Client:** Once the SSM tunnel is active, connect by navigating to a local address in any browser (<https://localhost:8443>)
6. **Bypass Security Warning:** The browser will likely display a security warning page (as shown in [Figure \[9\]](#) of the document) because the connection uses a self-signed certificate. Click the option to proceed to the site (e.g., "Proceed to ... (unsafe)").

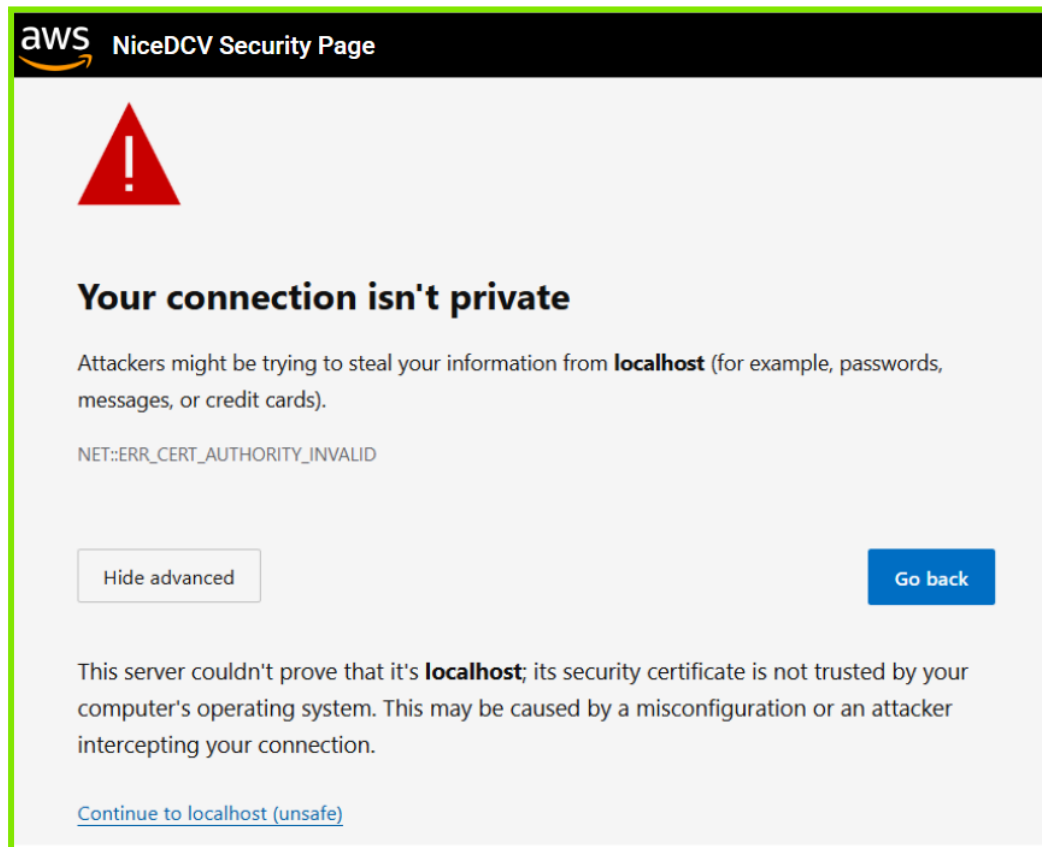


Figure [9]: NiceDCV Security Page

7. **Log In:** The NiceDCV login page will be displayed ([Figure \[10\]](#)). The provided Username and Password for the account should be entered.

Upon the first login, this will be the default login:

- Username is **ubuntu**
- Password is **ubuntu**

Password can be changed using the following command:

None

passwd ubuntu

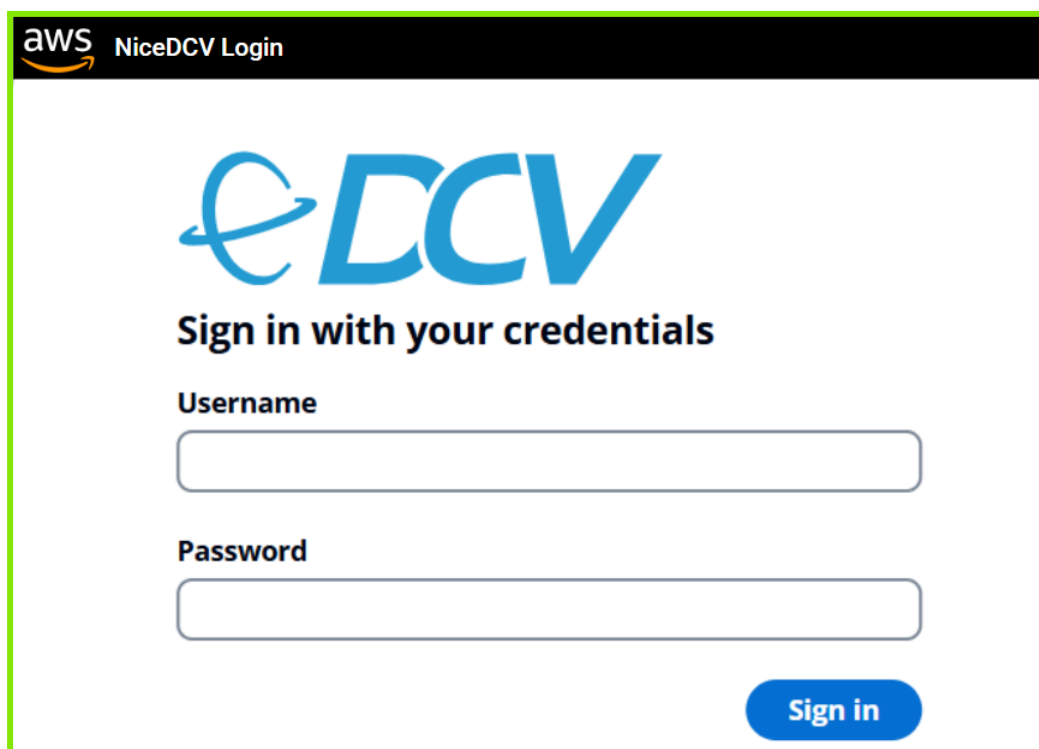


Figure [10]: NiceDCV Login

After a successful login, the connection will be established to the graphical desktop environment of the EC2 instance.

Nice DCV is also accessible via an SSM Connection, eliminating the need for a Public IP and inbound rule security group.

5. Solution's Content

The AMI package is located in the **/home/** directory , It is a folder called "ESMINI_VIRTUALIZED_HW_LAB_V*" as shown in [Figure \[11\]](#).

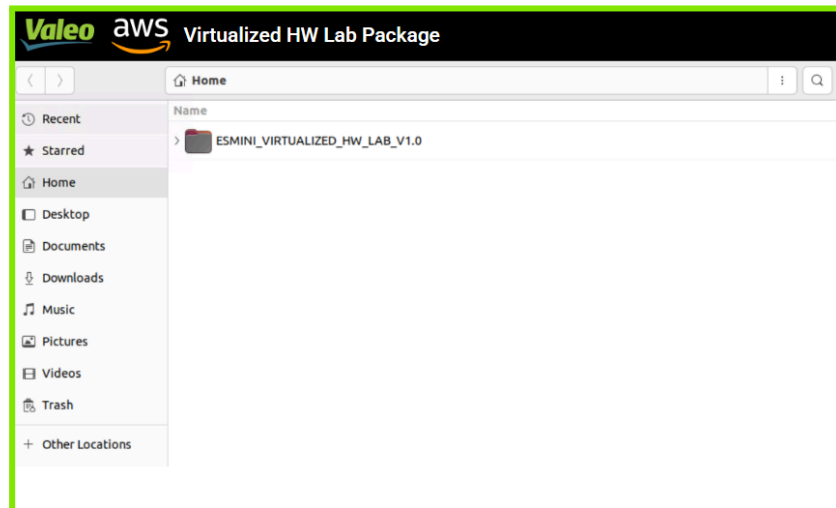


Figure [11]: Virtualized HW Lab Package

The AMI package contains two main folders: User_workspace and HW-Virtualization-lab as shown in [Figure \[12\]](#).

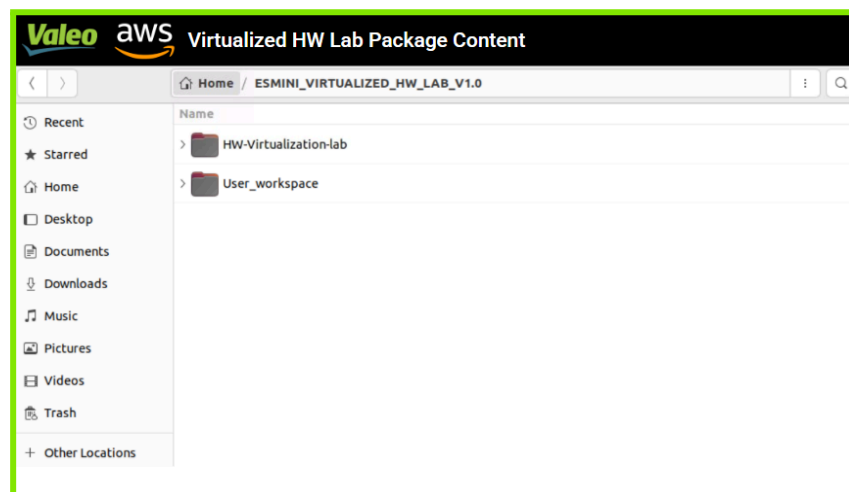
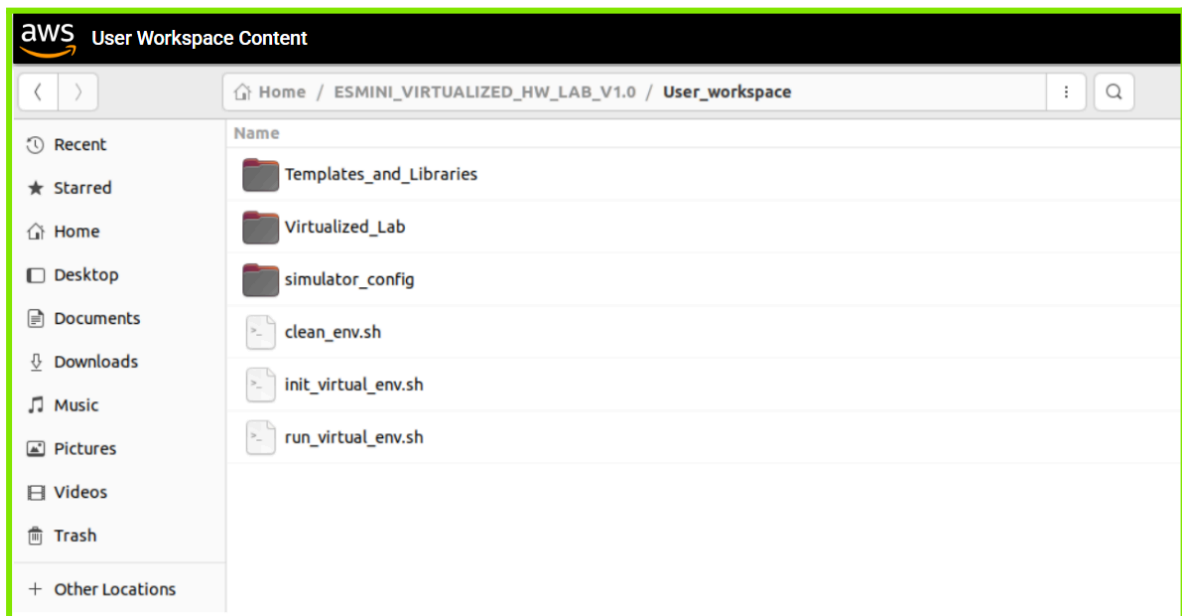


Figure [12]: Virtualized HW Lab Package Content

5.1. User Workspace

This directory is located at **/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace** as shown in

[Figure \[13\]](#)**Figure [13]: User Workspace Content**

and contains the primary components relevant to the user. Its contents include:

5.1.1. Scripts

- Environment Initialization Script [**init_virtual_env.sh**]
- Environment Running Script [**run_virtual_env.sh**]
- Clean Script [**clean_env.sh**]

5.1.2. Simulation Configuration File

- **Location:**
/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/simulat
or_config

- **Content:** Contains the configuration script for ESMINI tool.
 - Configuration file contains the following parameters:

```
JSON
{
  "ScenarioName": "<Scenario name> "
}
```

Field Name	Type	Description	Example
ScenarioName	String	The name of the scenario file to be run. This file must be an .xosc file located within the /home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/Templates_and_Libraries/ASAMScenarios/xosc directory.	MyScenario

5.1.3. Virtualized_Lab

- **Location:**
/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/Virtualized_Lab
- **Content:** Executable application to run and deal with Valeo solution

5.1.4. Templates and Libraries

- **Location:**
/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/Templates_and_Libraries as shown in [Figure \[14\]](#).

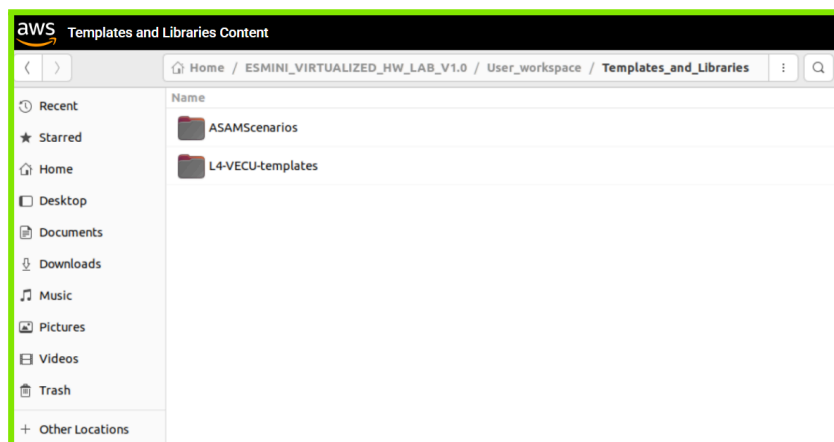


Figure [14]: Templates and Libraries Content

- **Content:** Includes all templates and libraries the user will need:

5.1.4.1. ASAMScenarios Directory

- **Location:**
/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/Templates_and_Libraries/ASAMScenarios as shown in [Figure \[15\]](#)

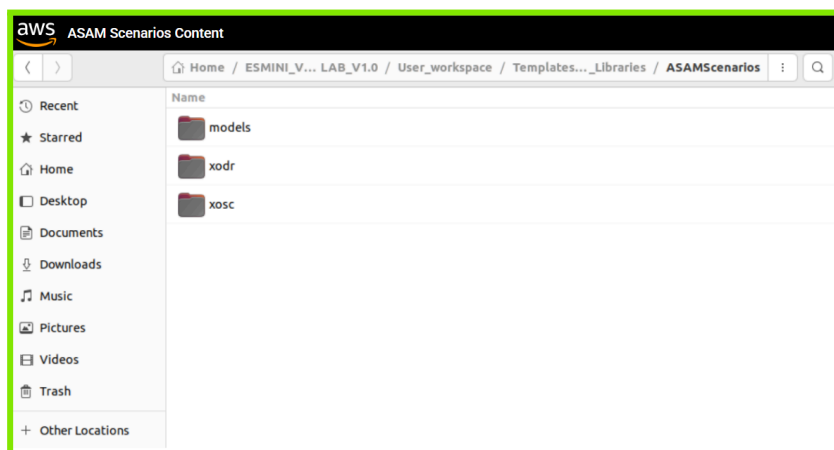


Figure [15]: ASAM Scenarios Content

- **Contains:** Valeo ASAMScenarios for use with ESMINI tool.
- **Usage:** ASAM scenarios examples can be used directly or as a reference for creating custom scenarios.

5.1.4.2. L4-VECU-templates Directory

- **Location:**
/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/Templates_and_Libraries/L4-VECU-template as shown in [Figure \[16\]](#)

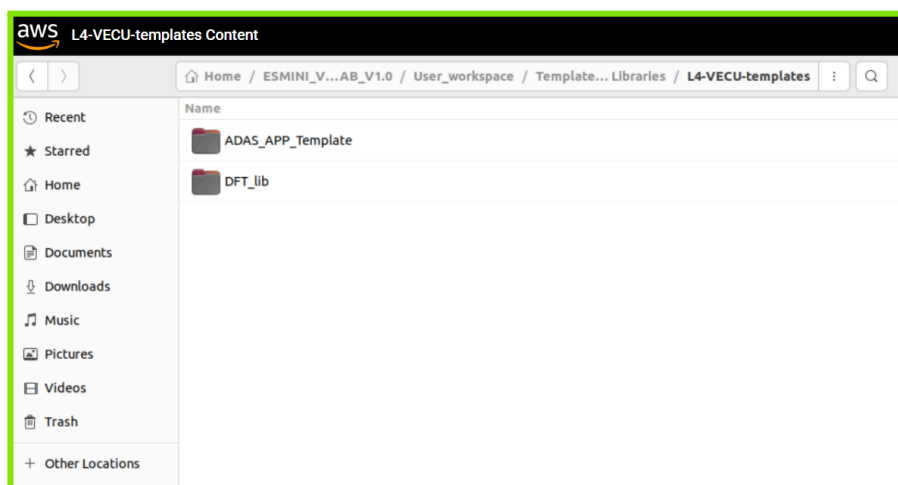


Figure [16]: L4-VECU-templates Content

- **Contains:**
 - **ADAS_APP_Template Directory:** contains an example ADAS application with a build script (cmake).
 - **DFT Directory:** contains:
 - **Lib Directory:** contains **libDFTReceiver.so** which is a provided receiver library for receiving data from Valeo's sensor model.
 - **Include Directory:** contains **DFTReceiver.h** which is a header file that contains the APIs needed to receive Sensor data in the application.

5.1.5. ADAS Application

5.1.5.1. Application Structure

Application structure The ADAS application is developed by the user to test and validate the sensor data received from the virtualized environment.

- **Location:**

/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/Templates_and_Libraries/L4-VECU-templates/ADAS_APP_Template as shown in [Figure \[17\]](#)

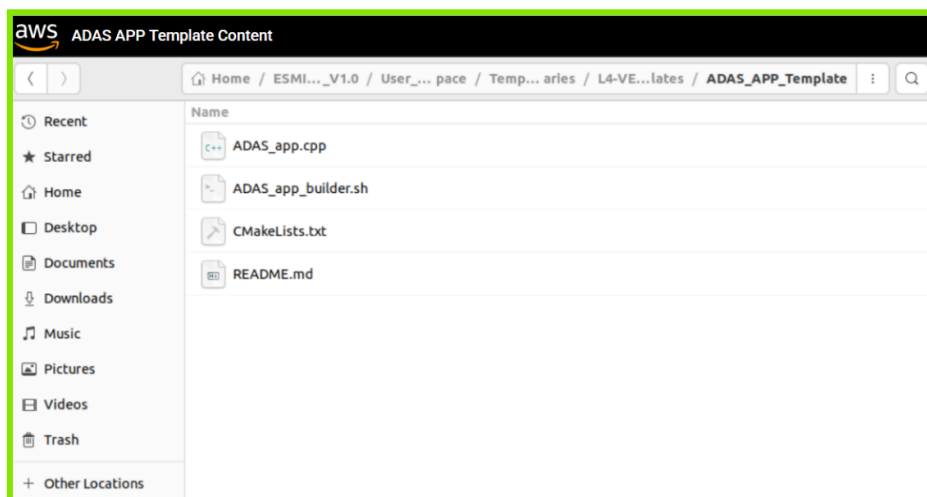


Figure [17]: ADAS APP Template Content

- **Structure:** The core structure is based on a **C++ application** (e.g., ADAS_app.cpp) that utilizes:

5.1.5.2. Available APIs

The primary APIs for the user-developed ADAS application are provided in the DFTReceiver.h header file.

The two main functions are:

- **initDftRecieverAPP():** This function:
 - Initializes and starts a detached background thread that continuously listens for incoming UDP packets containing sensor data. The received data is processed and made available for retrieval through the recieveSensorData() method.
 - Must be called after the constructor and before any data reception is attempted.
 - Returns DFT_SUCCESS (0) if the thread was started successfully, or DFT_FAILURE (1) if an initialization error occurred. The thread runs independently and will terminate if

a runtime error occurs or when the application is destructed.
If a runtime error happens, the thread will stop and set an internal error flag.

- **recieveSensorData(USFC_uls_det_itf_object_S &decoded_data):**

This is a blocking and thread-safe method that retrieves the latest sensor data that has been processed by the receiver thread. It waits until new data is available, fills the provided output struct with the decoded data, and then empties the internal queue.

- The function takes a reference to a **USFC_uls_det_itf_object_S** struct as an output parameter, which the caller must provide. If data is successfully retrieved, this struct will be populated.
- It returns DFT_SUCCESS (0) when data is successfully retrieved and the decoded_data struct is filled. It returns DFT_FAILURE (1) if no new data is available or if an error has occurred in the receiver thread.
- Important Notes:
 - Prerequisite: The application must be initialized via the constructor and initDftRecieverAPP() before this method can be called.
 - Usage: This method should be called periodically to check for and process new sensor data.

- **Received Struct (USFC_uls_det_itf_object_S):** With all structs it uses.

- **USFC_data_qualifier_E**

Data Qualifier enumerations. Expresses the validity and usability of the data

C/C++

```
enum USFC_data_qualifier_E
{
    USFC_DET_DQ_INVALID = 0,
    USFC_DET_DQ_NORMAL
};
```

Field Name	Type	Description
USFC_DET_DQ_INVALID	Enumeration Field	The data is invalid.
USFC_DET_DQ_NORMAL	Enumeration Field	The data can be used without restrictions.

- **USFC_iso_serving_sensors_S**

Represents the serving sensors for an interface.

C/C++

```
struct USFC_iso_serving_sensors_S
{
    u8 num_sensor_ids;
    u8 max_num_sensor_ids;
    u8 sensor_ids[USFC_DET_MAX_NUM_SERV_SENS];
};
```

Field Name	Type	Description
num_sensor_ids	u8	The number of sensors contributing to the interface.

Field Name	Type	Description
max_num_sensor_ids	u8	The maximum number of sensors that can contribute to the interface
sensor_ids[USFC_DET_MAX_NUM_SERV_SENS]	Array of u8	The IDs of the serving sensors.

○ USFC_iso_header_S

Iso Header. Contains metadata about the ISO-23150 data.

```
C/C++
struct USFC_iso_header_S
{
    f32 timestamp_s;
    struct USFC_iso_serving_sensors_S serving_sensors;
    uint16_t interface_version_id[3];
    enum USFC_data_qualifier_E data_qualifier;
};
```

Field Name	Type	Description
timestamp_s	f32	Timestamp of the data acquisition in seconds. This typically represents the time when the sensor data was captured or generated.
serving_sensors	USFC_iso_serving_sensors_S struct	Information about the sensors that contributed to this data interface.

Field Name	Type	Description
interface_version_id[3]	uint16_t	Version of the interface definition [major, minor, patch]. This helps in managing compatibility between different software versions. /* major, minor, patch */
data_qualifier	enum USFC_data_qualifier_E	Qualifier indicating the validity and usability of the data in this interface.

- **USFC_ultrasonic_detection_entity_S**
Represents a single ultrasonic detection.

```
C/C++
struct USFC_ultrasonic_detection_entity_S
{
    u8 existence_probability;
    f32 distance_m;
    f32 distance_error_m;
    f32 extent_height_m;
    f32 extent_height_error_m;
};
```

Field Name	Type	Description
existence_probability	u8	Probability of the detection's existence, typically in percentage (0-100) or a scaled value.

Field Name	Type	Description
distance_m	f32	Measured distance to the detected object in meters.
distance_error_m	f32	Estimated error or uncertainty in the measured distance, in meters.
extent_height_m	f32	Estimated extent (e.g., height or width) of the detected object in meters
extent_height_error_m	f32	Estimated error or uncertainty in the extent measurement, in meters.

- **USFC_ultrasonic_detection_entity_list_S**
Represents a list of ultrasonic detection entities.
This structure holds an array of individual detections and metadata about the list.

```
C/C++
struct USFC_ultrasonic_detection_entity_list_S
{
    struct USFC_ultrasonic_detection_entity_S
    detections[USFC_DET_MAX_NUM_DET_PER_SGW];
    u8 num_detections;
    u8 max_num_detections;
};
```

Field Name	Type	Description
detections[USFC_DET_MAX_NUM_DET_PER_SGW]	struct USFC_ultrasonic_detection_entity_S	Array of ultrasonic detection entities. Each element provides details about a specific detection.
num_detections	u8	The current number of valid detection entities in the `detections` array.
max_num_detections	u8	The maximum number of detection entities that can be stored in the `detections` array. This corresponds to the compile-time size of the array.

- **USFC_iso_ultrasonic_detection_interface_S**
Combines the header and a list of detection entities.
This structure represents a complete set of detection data from one or more sensors, conforming to a specific interface standard.

```
C/C++
struct USFC_iso_ultrasonic_detection_interface_S
{
    struct USFC_iso_header_S header;
    struct USFC_ultrasonic_detection_entity_list_S detection_list;
};
```

Field Name	Type	Description
header	struct USFC_iso_header_S	Header information for this ultrasonic detection interface, containing metadata like timestamp and sensor IDs.
detection_list	struct USFC_ultrasonic_detection_entity_list_S	List of ultrasonic detection entities associated with this interface.

- **USFC_det_itf_list_S**

Represents a list of ultrasonic detection interfaces.

This structure aggregates multiple

`USFC\_iso\_ultrasonic\_detection\_interface\_S` instances,

typically corresponding to different signal words or processing cycles.

```
C/C++
struct USFC_det_itf_list_S
{
    struct USFC_iso_ultrasonic_detection_interface_S
iso_uls_detections[USFC_DET_MAX_NUM_SGWS_PER_CYC];
    u8 num_iso_uls_detections;
    u8 max_num_iso_uls_detections;
};
```

Field Name	Type	Description
iso_uls_detections[USFC_DET_MAX_NUM_SGWS_PER_CYC]	struct USFC_iso_ultrasonic_detection_interface_S	Array of ISO ultrasonic detection interfaces.
num_iso_uls_detections	u8	The number of valid ISO ultrasonic detection interfaces.
max_num_iso_uls_detections	u8	The maximum number of ISO ultrasonic detection interfaces that can be stored in the `iso_uls_detections` array. This corresponds to the compile-time size of the array.

- **USFC_uls_det_itf_object_S**

Structure for decoded sensor data, primarily containing a list of ISO ultrasonic detection interfaces.

This structure encapsulates the processed data from ultrasonic sensors, formatted according to the USFC (Ultrasonic Sensor Fusion Component) ISO detection interface standard. It includes a list of detection interfaces, along with metadata like a live counter and an availability flag.

```
C/C++
struct USFC_uls_det_itf_object_S
{
    struct USFC_det_itf_list_S uls_det_itf_list;
    u8 live_cnt;
    bool is_available;
};
```

Field Name	Type	Description
uls_det_itf_list	struct USFC_det_itf_list_S	List of ISO detection interfaces for all signal ways processed in the last cycle.
live_cnt	u8	Live counter incremented every time data is available.
is_available	bool	Flag indicating if the data in this structure is new or has been updated in the current processing step. `true` if data is fresh/updated, `false` otherwise.

5.1.5.3. Valeo Examples

Valeo examples' template for ADAS application (ADAS_app.cpp) is available as a reference for developing custom applications. Refer to [§5.1.5 ADAS Application](#)

Application development requires including the DFTRceiver.h header file and linking with the libDFTRceiver.so library.

Note: If the build configuration changes or a new library is added to the application, the adjacent CMake file must be updated accordingly.

The **ADAS_app.cpp** file is the reference ADAS application, demonstrating how to receive and validate sensor data from the simulation environment. It uses the Google Test framework to execute test cases that verify specific aspects of the system's functionality.

Below is a detailed explanation of each test case and instructions on how to run them.

The following test cases are included in the ADAS_app.cpp example:

1. ADAS_TIMESTAMP_001

- **Name:** GenericTool_TimestampValidation.ADAS_TIMESTAMP_001
- **Description:** This test validates that the timestamp (timestamp_s) in the received sensor data packets is strictly increasing over time. It collects several data frames at set intervals and compares the timestamp of each frame to the previous one.
- **Purpose:** To perform a basic health check on the data pipeline, ensuring that data packets are processed in the correct temporal order and that the simulation's time source is advancing as expected.
- **Input:** The test processes USFC_uls_det_itf_object_S data structures obtained by repeatedly calling the receiveSensorData method of the dftReceiverAPP class. It specifically analyzes the timestamp_s field within the nested USFC_iso_header_S struct of each received object. library.
- **Pass/Fail Criteria:**
 - **Pass:** Each timestamp_s value extracted from consecutive USFC_uls_det_itf_object_S objects is strictly greater than the previous one.
 - **Fail:** A timestamp_s value is less than or equal to the previous one, or if an insufficient number of data packets are received to perform the comparison.

2. ADAS_LIVECNT_001

- **Name:** GenericTool_LiveCounterValidation.ADAS_LIVECNT_001
- **Description:** This test validates that the live_cnt field in the sensor data increments correctly by one for each consecutive frame. It also correctly handles the wrap-around case where the counter resets from 255 to 0.
- **Purpose:** To verify data continuity and detect potential packet loss between the sensor model and the ADAS application. A continuous live_cnt sequence indicates a stable data stream.
- **Input:** The test processes a sequence of USFC_uls_det_itf_object_S data structures obtained by calling the receiveSensorData method of the dftReceiverAPP class. It specifically validates the live_cnt member of this struct.
- **Pass/Fail Criteria:**
 - **Pass:** For each consecutive USFC_uls_det_itf_object_S object received, the live_cnt value is equal to $(\text{previous_live_cnt} + 1) \% 256$.
 - **Fail:** The live_cnt sequence is broken (i.e., it does not follow the expected increment or wrap-around logic).

3. ADAS_ESMINI_STATIC_001_12_SENSORS

- **Name:**
ESMINI_Tool_Validation.ADAS_ESMINI_STATIC_001_12_SENSORS
- **Description:** This test is designed for a static Esmini scenario (StaticObjects.xosc) where the ego vehicle is surrounded by objects at a fixed distance. It verifies that all **12** ultrasonic sensors consistently report a distance of 1.0 meter, within a tolerance of +/- 0.1 meters, over a series of 50 frames.

- **Purpose:** To verify the baseline accuracy and stability of all ultrasonic sensors in a controlled, non-moving Esmini environment.
- **Input:** The test processes USFC_uls_det_itf_object_S data structures obtained via the recieveSensorData method from the dftRecieverAPP class. It analyzes the distance_m from the USFC_ultrasonic_detection_entity_S struct for each of the 12 sensors.
- **Pass/Fail Criteria:**
 - **Pass:** For all 50 validated frames, the reported distance_m for each of the 12 active sensors is within the range of 1.0m +/- 0.1m.
 - **Fail:** Any sensor reports a distance_m outside the specified tolerance at any point during the test.

4. ADAS_ESMINI_STATIC_001_8_SENSORS

- **Name:**
ESMINI_Tool_Validation.ADAS_ESMINI_STATIC_001_8_SENSOR
- **Description:** This test is designed for a static Esmini scenario (StaticObjects.xosc) where the ego vehicle is surrounded by objects at a fixed distance. It verifies that all **8** ultrasonic sensors consistently report a distance of 1.0 meter, within a tolerance of +/- 0.1 meters, over a series of 50 frames.
- **Purpose:** To verify the baseline accuracy and stability of all ultrasonic sensors in a controlled, non-moving Esmini environment.
- **Input:** The test processes USFC_uls_det_itf_object_S data structures obtained via the recieveSensorData method from the dftRecieverAPP class. It analyzes the distance_m from the USFC_ultrasonic_detection_entity_S struct for each of the 8 sensors.

- **Pass/Fail Criteria:**

- **Pass:** For all 50 validated frames, the reported distance_m for each of the 8 active sensors is within the range of 1.0m +/- 0.1m.
- **Fail:** Any sensor reports a distance_m outside the specified tolerance at any point during the test.

5. ADAS_ESMINI_DYNAMIC_001_12_SENSORS

- **Name:**

ESMINI_Tool_Validation.ADAS_ESMINI_DYNAMIC_001_12_SENSORS

- **Description:** This test validates sensor behavior in a dynamic Esmini scenario (MovingEgo.xosc) where the ego vehicle moves past objects. It checks the behavior of different sensor groups of the **12** sensors across five distinct phases of the scenario, validating patterns like "pass-by" (distance decreases then increases) and "approach-and-hold" (distance decreases and then stays constant).
- **Purpose:** To verify the system's ability to correctly track objects and manage sensor hand-offs as the vehicle moves. It validates the dynamic response of the sensors according to a predefined scenario sequence.
- **Input:** Sensor data generated by an Esmini simulation running the **MovingEgo.xosc** scenario. The test processes a continuous stream of USFC_uls_det_itf_object_S structures obtained via the receiveSensorData method. It analyzes the distance_m values over time for all 12 sensors to validate their behavior against expected patterns.

- **Pass/Fail Criteria:**
 - **Pass:** The sequence of distance_m readings from all sensor groups (front, rear, side, outer) matches their expected behavioral patterns (e.g., pass-by, approach-and-hold) for each of the five phases of the scenario.
 - **Fail:** Any sensor group's readings deviate from the expected pattern in any phase

6. ADAS_ESMINI_DYNAMIC_001_8_SENSORS

- **Name:**
ESMINI_Tool_Validation.ADAS_ESMINI_DYNAMIC_001_8_SENSORS
- **Description:** This test validates sensor behavior in a dynamic Esmini scenario (**MovingEgo.xosc**) where the ego vehicle moves past objects. It checks the behavior of different sensor groups of the **8** sensors across five distinct phases of the scenario, validating patterns like "pass-by" (distance decreases then increases) and "approach-and-hold" (distance decreases and then stays constant).
- **Purpose:** To verify the system's ability to correctly track objects and manage sensor hand-offs as the vehicle moves. It validates the dynamic response of the sensors according to a predefined scenario sequence.
- **Input:** Sensor data generated by an Esmini simulation running the **MovingEgo.xosc** scenario. The test processes a continuous stream of USFC_uls_det_itf_object_S structures obtained via the receiveSensorData method. It analyzes the distance_m values over time for all 8 sensors to validate their behavior against expected patterns.

- **Pass/Fail Criteria:**
 - **Pass:** The sequence of distance_m readings from all relevant sensor groups (front, side, outer) matches their expected behavioral patterns for each phase of the scenario.
 - **Fail:** Any sensor group's readings deviate from the expected pattern in any phase.

5.2. HW-Virtualization-lab

This directory is located at **/home/ESMINI_VIRTUALIZED_HW_LAB_V*/HW-Virtualization-lab** and contains the core components of the virtualized solution as shown in [Figure \[18\]](#).

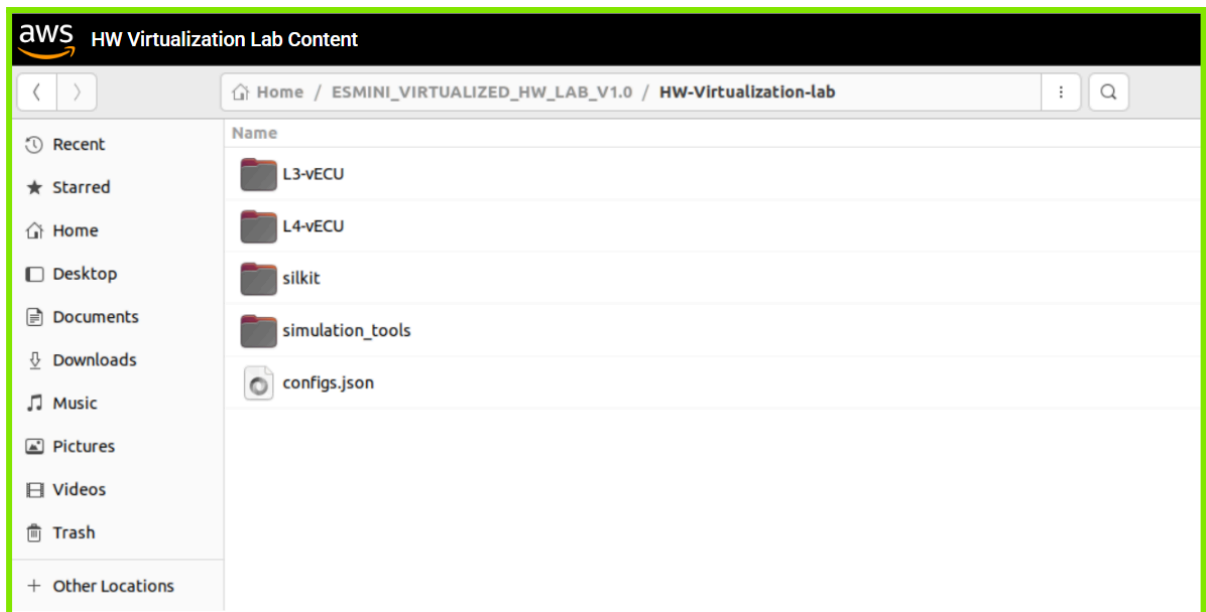


Figure [18]: HW Virtualization lab Content

5.2.1. Silkit

The SilKit communication framework components.

5.2.2. simulation_tools

Contains the simulation tool server that will be used and its required libraries.

5.2.3. L4_vECU

Contains the solution's Level 4 virtual ECU using Qemu and Silkit Qemu Adapter.

5.2.4. HW Lab Configuration Files

The Config.json file is crucial for initializing and running the simulation, as it defines the basic paths for package folders and files. This configuration file acts as a parameter for the initialization and running script.

Attribute	Value	Description
pythonVersion_Linux	3.7	Defines the default value of the used python
Silkit_path	silkit	The default path for the silkit component, relative to the HW-Virtualization-lab directory.
Silkit_Adapter_path	L4-vECU/Qemu_Participant/sil-kit-adapters-qemu	The default path for the sil-kit-adapters-qemu component, relative to the HW-Virtualization-lab directory.

Attribute	Value	Description
QEMU_path	L4-vECU/Qemu_Disk_Image	The default path for the L4-vECU component, relative to the HW-Virtualization-lab directory.
Simulation_Tools_Path	simulation_tools	The default path for the ESMINI Tool component, relative to the HW-Virtualization-lab directory.
Simulator_config_path	../User_workspace/simulator_config	The default path for the ESMINI tool configuration file, relative to the HW-Virtualization-lab directory.
TemplatesAndLibraries_path	../User_workspace/Templates_and_Libraries	The default path for the Templates_and_Libraries directory, relative to the HW-Virtualization-lab directory.
FMU_Participant_path	L3-vECU/FMUParticipant	The default path for the Ultrasonic SM FMU directory, relative to the HW-Virtualization-lab directory.

5.2.5. Ultrasonics SM FMU

This directory contains Valeo's ultrasonic sensor model. This model is essential for the solution's operation. To ensure system stability and correct functionality, modifications within this directory should be restricted to the config.json file only. Altering other files is not advised, as it may lead to performance issues or unexpected behavior

5.2.6. Ultrasonics SM FMU Configuration File

Location: This configuration file (**config.json**) is located at **/home/ESMINI_VIRTUALIZED_HW_LAB_V*/HW-Virtualization-lab/L3-vECU/FMUParticipant**

Content: Configuration for the ultrasonic sensors distributed on the vehicle and this config contains these parameters:

```
JSON
{
  "SensorModel_Variables": {
    "VerboseMode_bool":      true
  },
  "SensorModel_Parameters": {
    "SensorCount_int":      12,

    "Sensors[0].Name_string":  "FSL",
    "Sensors[0].Pos.X_m":      3.411,
    "Sensors[0].Pos.Y_m":      0.942,
    "Sensors[0].Pos.Z_m":      0.54,
    "Sensors[0].Rot.Roll_deg":  0.0,
    "Sensors[0].Rot.Pitch_deg": 0.0,
    "Sensors[0].Rot.Yaw_deg":   95.799496,

    "Sensors[1].Name_string":  "FOL",
    "Sensors[1].Pos.X_m":      3.619,
    "Sensors[1].Pos.Y_m":      0.795,
    "Sensors[1].Pos.Z_m":      0.57,
    "Sensors[1].Rot.Roll_deg":  0.0,
    "Sensors[1].Rot.Pitch_deg": 0.0,
    "Sensors[1].Rot.Yaw_deg":   40.199588,

    "Sensors[2].Name_string":  "FIL",
    "Sensors[2].Pos.X_m":      3.845,
```

```

"Sensors[2].Pos.Y_m":      0.364,
"Sensors[2].Pos.Z_m":      0.36,
"Sensors[2].Rot.Roll_deg": 0.0,
"Sensors[2].Rot.Pitch_deg": 0.0,
"Sensors[2].Rot.Yaw_deg":  8.498054,

"Sensors[3].Name_string":  "FIR",
"Sensors[3].Pos.X_m":      3.845,
"Sensors[3].Pos.Y_m":      -0.364,
"Sensors[3].Pos.Z_m":      0.36,
"Sensors[3].Rot.Roll_deg": 0.0,
"Sensors[3].Rot.Pitch_deg": 0.0,
"Sensors[3].Rot.Yaw_deg":  -8.498054,

"Sensors[4].Name_string":  "FOR",
"Sensors[4].Pos.X_m":      3.619,
"Sensors[4].Pos.Y_m":      -0.795,
"Sensors[4].Pos.Z_m":      0.57,
"Sensors[4].Rot.Roll_deg": 0.0,
"Sensors[4].Rot.Pitch_deg": 0.0,
"Sensors[4].Rot.Yaw_deg":  -40.199588,

"Sensors[5].Name_string":  "FSR",
"Sensors[5].Pos.X_m":      3.411,
"Sensors[5].Pos.Y_m":      -0.942,
"Sensors[5].Pos.Z_m":      0.54,
"Sensors[5].Rot.Roll_deg": 0.0,
"Sensors[5].Rot.Pitch_deg": 0.0,
"Sensors[5].Rot.Yaw_deg":  -95.799496,

"Sensors[6].Name_string":  "RSR",
"Sensors[6].Pos.X_m":      -0.494,
"Sensors[6].Pos.Y_m":      -0.934,
"Sensors[6].Pos.Z_m":      0.56,
"Sensors[6].Rot.Roll_deg": 0.0,
"Sensors[6].Rot.Pitch_deg": 0.0,
"Sensors[6].Rot.Yaw_deg":  -70.102082,

"Sensors[7].Name_string":  "ROR",
"Sensors[7].Pos.X_m":      -0.88,
"Sensors[7].Pos.Y_m":      -0.64,
"Sensors[7].Pos.Z_m":      0.54,
"Sensors[7].Rot.Roll_deg": 0.0,
"Sensors[7].Rot.Pitch_deg": 0.0,
"Sensors[7].Rot.Yaw_deg":  -149.899748,

```

```

"Sensors[8].Name_string": "RIR",
"Sensors[8].Pos.X_m": -0.979,
"Sensors[8].Pos.Y_m": -0.325,
"Sensors[8].Pos.Z_m": 0.57,
"Sensors[8].Rot.Roll_deg": 0.0,
"Sensors[8].Rot.Pitch_deg": 0.0,
"Sensors[8].Rot.Yaw_deg": -170.801556,

"Sensors[9].Name_string": "RIL",
"Sensors[9].Pos.X_m": -0.979,
"Sensors[9].Pos.Y_m": 0.325,
"Sensors[9].Pos.Z_m": 0.57,
"Sensors[9].Rot.Roll_deg": 0.0,
"Sensors[9].Rot.Pitch_deg": 0.0,
"Sensors[9].Rot.Yaw_deg": 170.801556,

"Sensors[10].Name_string": "ROL",
"Sensors[10].Pos.X_m": -0.88,
"Sensors[10].Pos.Y_m": 0.64,
"Sensors[10].Pos.Z_m": 0.54,
"Sensors[10].Rot.Roll_deg": 0.0,
"Sensors[10].Rot.Pitch_deg": 0.0,
"Sensors[10].Rot.Yaw_deg": 149.899748,

"Sensors[11].Name_string": "RSL",
"Sensors[11].Pos.X_m": -0.494,
"Sensors[11].Pos.Y_m": 0.934,
"Sensors[11].Pos.Z_m": 0.56,
"Sensors[11].Rot.Roll_deg": 0.0,
"Sensors[11].Rot.Pitch_deg": 0.0,
"Sensors[11].Rot.Yaw_deg": 70.102082
}
}

```

Attribute	Description
SensorModel_Variables.VerboseMode_bool	Enabling/disabling Printing log data to stdout.
SensorModel_Parameters.SensorCount_int	Number of ULS sensors can be 2, 4, 8, or 12
SensorModel_Parameters.Sensors[NUMBER].Name_string	Variable defines the sensor name
SensorModel_Parameters.Sensors[NUMBER].Pos.X_m	Variable defines the x position [in meters] of the sensor relative to Ego vehicle rear axis.
SensorModel_Parameters.Sensors[NUMBER].Pos.Y_m	Variable defines the y position [in meters] of the sensor relative to Ego vehicle rear axis.
SensorModel_Parameters.Sensors[NUMBER].Pos.Z_m	Variable defines the z position [in meters] of the sensor relative to Ego vehicle rear axis.
SensorModel_Parameters.Sensors[NUMBER].Rot.Roll_deg	Variable defines the rotation value of the sensor relative to Ego vehicle rear axis.

Attribute	Description
SensorModel_Parameters.Sensors [NUMBER].Rot.Pitch_deg	Variable defines the pitch value of the sensor relative to Ego vehicle rear axis.
SensorModel_Parameters.Sensors [NUMBER].Rot.Yaw_deg	Variable defines the yaw value of the sensor relative to Ego vehicle rear axis.

6. Environment Setup

6.1. Virtualized Lab GUI

The Virtualized Lab serves as a user interface application, designed to simplify the overall solution usage.

The §5.2.4 HW Lab Configuration Files of Virtualized Lab is parsed from the defined location in the package.

6.1.1. Accessing the Virtualized Lab

To start the Virtualized Lab, two methods are available for user to do so:

- **Desktop Icon:** There will be a desktop icon deployed on our instances that opens the Virtualized Lab directly to the user. as shown on [Figure \[19\]](#).

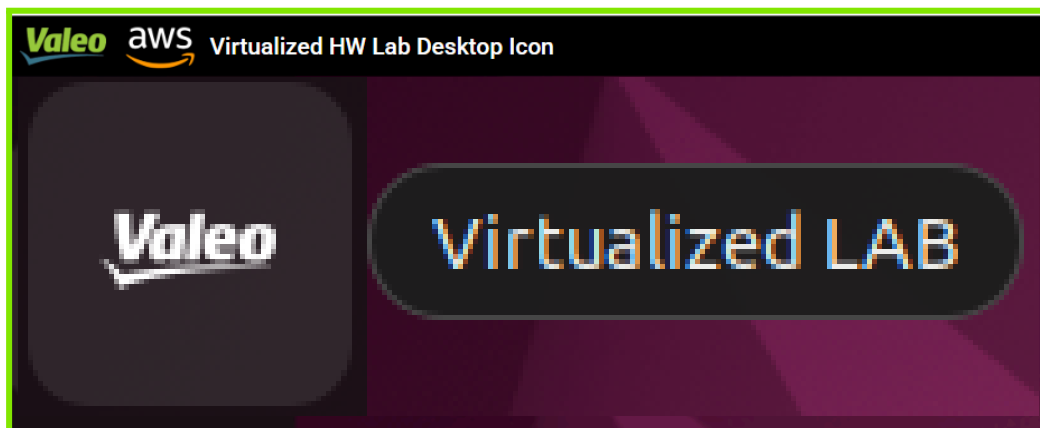


Figure [19]: Virtualized HW Lab Desktop Icon

- **Terminal alias command:** User writing command “Valeo_Virtualized_HW_Lab” on any terminal will open Valeo's User Interface.

For better Virtualized Lab experience it is recommended to go to the display settings on your local machine and adjust the scale to 100%.

6.1.2. Virtualized Lab Main Window

This main window shown on [Figure \[20\]](#) is the central control hub, divided into two primary sections:

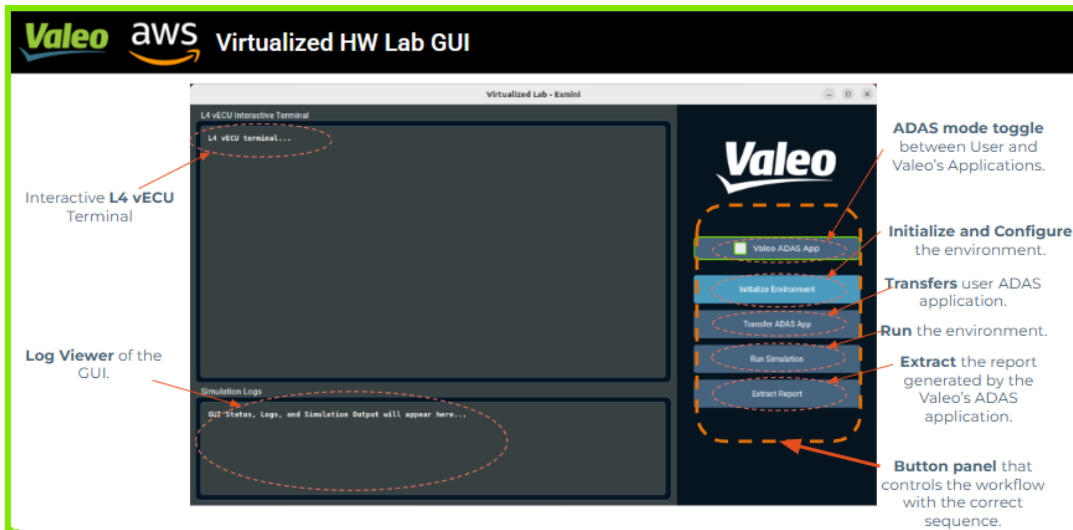


Figure [20]: Virtualized HW Lab

6.1.2.1. Interactive L4 Terminal

Provides a direct, interactive SSH shell to the L4 vECU for manual commands, debugging, and file system exploration. It becomes active after the "Initialize Environment" step is successfully completed.

6.1.2.2. Log Viewer

Displays real-time status messages from the Virtualized Lab and output from background scripts. Messages are color-coded for readability.

6.1.2.3. Button Panel

This area contains the primary controls for guiding the user through the simulation workflow.

Buttons are dynamically enabled and disabled to enforce the correct order of operations.

Important Note: The Virtualized Lab opens and operates based on the simulation tool delivered with the package of the bought AMI.

6.1.3. Virtualized Lab Usage Workflow

The "Valeo ADAS App" checkbox significantly alters the GUI's behavior, offering two distinct workflows based on the user's need for using the Valeo ADAS application or his own ADAS application. Here are the steps a user would take for each mode:

6.1.3.1. Valeo ADAS App Workflow

- Check the **"Valeo ADAS App"** checkbox.
- **Click "Initialize Environment":** The Configuration window will open, presenting a dropdown of predefined, validated test scenarios. Select the desired scenario. It also requires an input of the Ego vehicle's dimensions and center to rear axle shift. Should also state the number of sensors placed on the Ego vehicle and each sensor name, position with respect to rear axle center and rotation.
- You can load either a previously saved configuration file or the default configuration. These files are located in the "User_workspace/sensor_config" directory.
- The Configuration window for enabled "Valeo ADAS App" is shown in [Figure \[21\]](#).

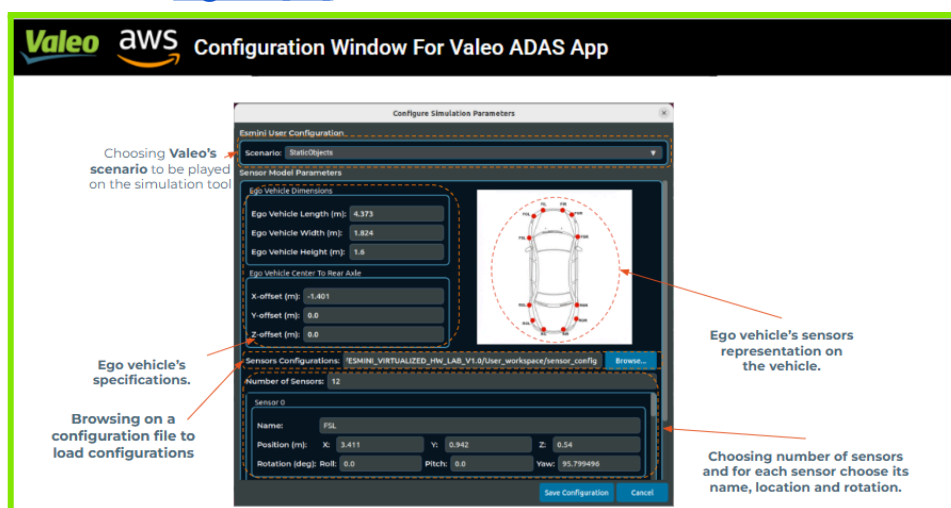


Figure [21]: Configuration Window For Valeo ADAS App

- This will set up the virtual environment. Once complete, the "Run Simulation" button will become enabled and the L4 vECU terminal will become interactive to users.

- **Click "Run Simulation":** The run test window will open. This window will ask for the desired test suite and test case to be running on the L4 vECU from the test examples (refer to [§7.1.4.4 Valeo Examples](#)) provided in the ADAS application template delivered. As Shown in [Figure \[22\]](#).

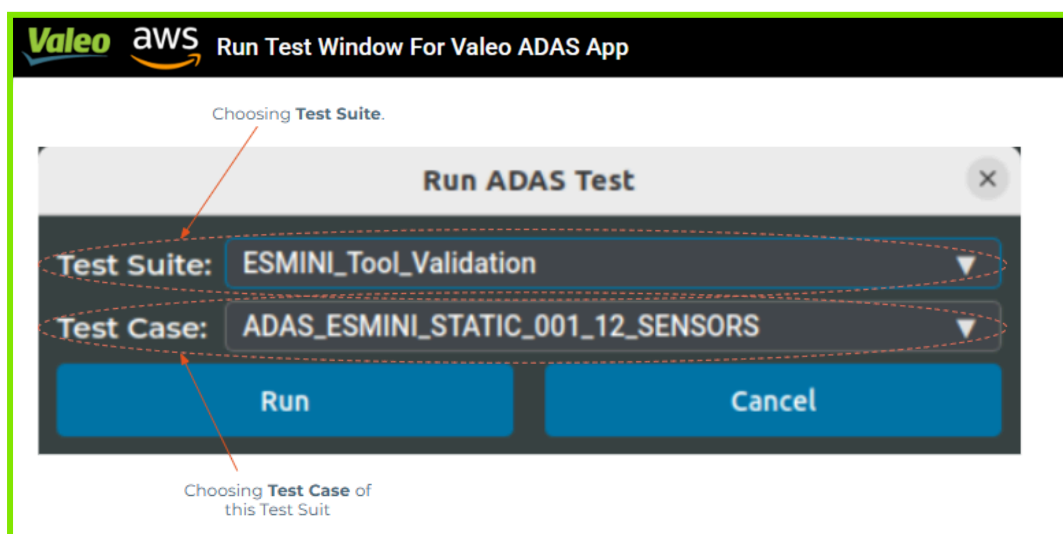


Figure [22]: Run Test Window For Valeo ADAS App

- **Select Test Suite and Test Case:** Choose the specific Google Test you wish to execute.
The selected test will run as a command within the interactive SSH terminal. The "Extract Valeo ADAS Report" button will become enabled.
- **Click "Extract Valeo ADAS Report":** Downloads the uniquely named XML report file generated by the last ADAS test run from the vECU and generates HTML report in the directory "User_workspace/ValidationReports".

6.1.3.2. User ADAS App Workflow

- Ensure the **"Valeo ADAS App"** checkbox is **UNCHECKED**.
- **Click "Initialize Environment":** The ConfigurationDialog will open, allowing you to browse for any scenario file. Select your desired scenario.
It also requires an input of the Ego vehicle's dimensions and center to rear axle shift. Should also state the number of sensors placed on the Ego vehicle and each sensor name, position with respect to rear axle center and rotation.
- You can load either a previously saved configuration file or the default configuration. These files are located in the "User_workspace/sensor_config" directory.
- The Configuration window for disabled "Valeo ADAS App" is shown in [Figure \[23\]](#).

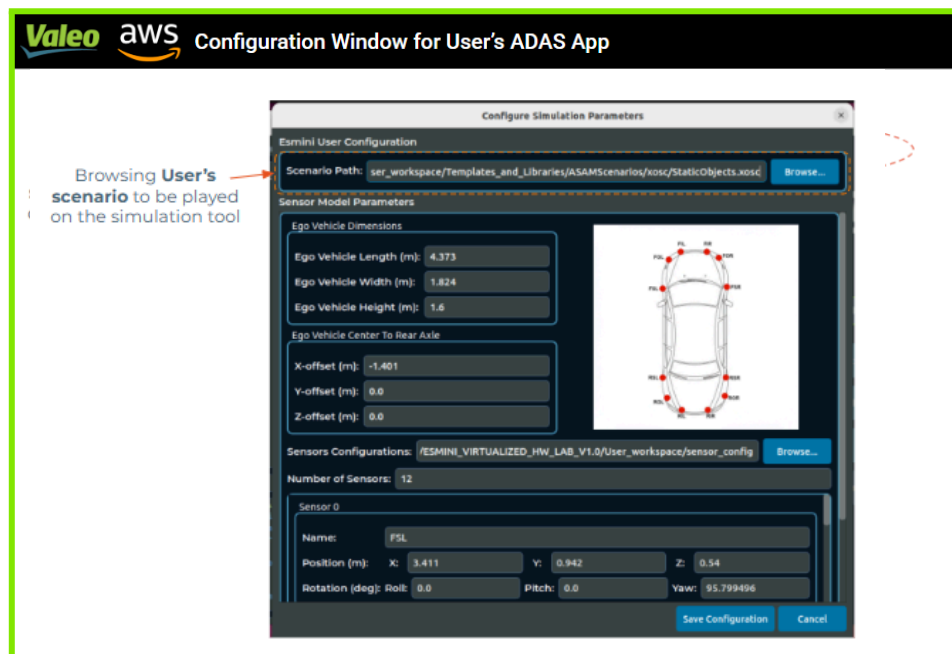


Figure [23]: Configuration Window for User's ADAS App

- This will set up the virtual environment. Once complete, the "Run Simulation" and "Transfer ADAS App" buttons will become enabled. The L4 vECU terminal will become interactive for users to run their ADAS application after transferring it.

- **Click "Transfer ADAS App" (Optional):** This opens a full-featured file transfer dialog, as shown in [Figure \[24\]](#).
 - The dialog presents a dual-pane interface for managing files between the local AWS instance (left pane) and the remote L4 vECU (right pane).
 - Navigation through the directories is possible on both systems. New folders can be created, and files and folders can be renamed or deleted in both windows using the right-click context menu.
 - To transfer your ADAS application, navigate to its location on the local system. Select the application files/folders and use the ">" button or drag-and-drop them into the desired directory on the remote L4 vECU.

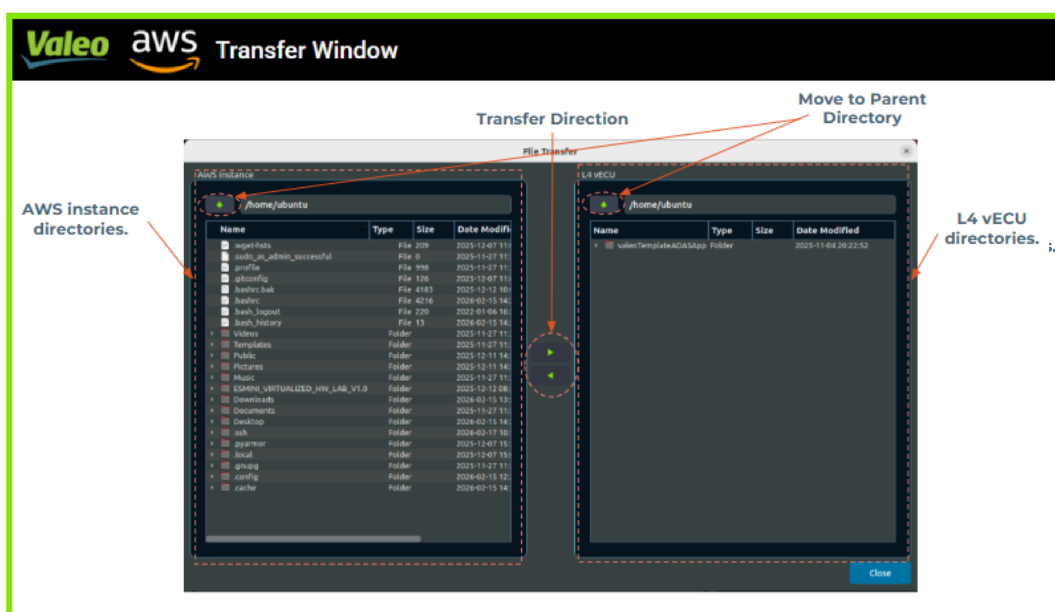


Figure [24]: Transfer Window

- **Run your ADAS application** with the command of your choice in the L4 vECU terminal.
- **Click "Run Simulation":** This will directly execute the `run_virtual_env.sh` script, starting the simulation client components.
- **Extract Valeo ADAS Report Button:** This button remains disabled in this mode.

6.2. Manual Setup

6.2.1. Environment Initialization

6.2.1.1. Initialization Setup Process

To set up the solution environment, execute the script named **"init_virtual_env.sh"**.

The execution can be done using one of the following commands:

1. Without Parameters:

In this case, the initialization script will refer to the default **"config.json"** in [§5.2.4 HW-Virtualization-lab](#).

None

```
./init_virtual_env.sh
```

2. With Configuration File Path:

If using a custom configuration, the initialization script will refer to the "config.json" provided within the command. Ensure this file includes the same parameters as the default configuration located in `"/home/ESMINI_VIRTUALIZED_HW_LAB_V*/HW-Virtualization-lab"`, respecting case sensitivity as illustrated in [§5.2.4 HW-Virtualization-lab](#). Subsequently, add the correct paths for each configuration, whether supported or custom.

None

```
./init_virtual_env.sh "FULL_PATH_TO_CUSTOM_CONFIG"
```

- The script ensures that all components, including the communication layer and the virtual ECUs, are set up with the correct configurations as defined in this file.
- The initialization process involves setting up the following:
 - **SilKit communication framework:** which facilitates Ethernet communication between the L4 vECU and the L3 vECU.

- ### 6.2.1.2. Environment Verification

None

```

.....
Simulation intialized successfully
.....

```

Additionally, the interactive SSH shell to the L4 vECU becomes active, allowing for manual commands and verification that the virtual environment is running correctly.

6.2.2. Environment Running

6.2.2.1. Running Setup Process

- Running the environment can be done through the automation script, using one of the following commands:

1. Without Parameters:

In this case, the initialization script will refer to the default “**config.json**” in [§5.2.4 HW-Virtualization-lab](#).

None

```
./run_virtual_env.sh
```

2. With Configuration File Path:

If using a custom configuration, the initialization script will refer to the "config.json" provided within the command. Ensure this file includes the same parameters as the default configuration located in

"/home/ESMINI_VIRTUALIZED_HW_LAB_V*/HW-Virtualization-lab", respecting case sensitivity as illustrated in [§5.2.4 HW-Virtualization-lab](#). Subsequently, add the correct paths for each configuration, whether supported or custom.

None

```
./run_virtual_env.sh "FULL_PATH_TO_CUSTOM_CONFIG"
```

- The script's primary function is to connect the various simulation participants to the SIL Kit registry and start the main scenario driver.
- Before running the environment you need to refer to the Simulation Tool, to complete the whole solution cycle as this part differs depending on the simulation tool used, please refer to the simulation tool manual.

6.2.2.2. ADAS Application Procedures

1. To build the application, The "**L4-VECU-templates**" folder must be copied from the developer workspace inside the AMI to the L4 vECU using SCP as following:

None

```
scp -P 3232 -r  
/home/ubuntu/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/Templates_and_L  
ibraries/L4-VECU-templates/ ubuntu@localhost:<desired location on the  
vECU>
```

2. After copying, to generate the executable, simply run the "**ADAS_app_builder.sh**" script on the L4 vECU. This script uses cmake to build the application, and the resulting executable will be mounted in the same folder, requiring no further steps.

None

```
cd <Location on the vECU>/L4-VECU-templates/ADAS_APP_Template
```

```
./ADAS_app_builder.sh
```

- **Note: Alternatively, the environment can be set up and built on any ARM EC2 instance, provided the necessary libraries are installed. The required libraries are *cmake*, *make*, and *libgtest-dev* and can be installed using the following command**

None

```
sudo apt install cmake  
  
sudo apt install make  
  
sudo apt install libgtest-dev
```

3. Next, use the following command to SSH from the AMI to Valeo's vECU:

None

```
ssh -p 3232 ubuntu@localhost
```

4. Navigate to the location that has the application and make sure that the libraries are reachable to the application using the following commands:

None

```
cd <Location on the vECU>/L4-VECU-templates/ADAS_APP_Template  
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/DFT_lib/lib
```

5. Then, run the ADAS application

None

```
./ADAS_app_exe
```

6. To run a specific test the command should be:

None

```
./ADAS_app_exe --gtest_filter=<Test to run>
```

7. To extract a xml test report the run command should look like this:

None

```
./ADAS_app_exe --gtest_output="xml:<Report_location_in_vECU>/report.xml"
```

8. To send the report to the instance the following command should be used:

None

```
scp -P 3232 ubuntu@localhost:<Report_location_in_vECU>/report.xml  
<Desired Location on Instance>
```

- **Note: If a password is required in any of these steps the default password is “ubuntu”.**

7. Simulation Tool

7.1. Pre-requisite

- Integration with external simulation tools is facilitated through gRPC-protobuf. This approach requires the external simulation environment to implement a gRPC-protobuf method named

"TriggerStep". This can be done in any programming language that supports protobuf code generation.

- Simulation tool is should also support OSI generation

7.2. Integration Process

- **Generate gRPC Files:** Generate the necessary gRPC files from the provided "message.proto" file. This file defines a service externalSync and an RPC method TriggerStep.
- Create a file that defines the Protobuf messages and service for communication between any external simulation tool and Valeo sensor model FMUs. It facilitates triggering simulation steps and exchanging data like sensor views.

None

```
syntax = "proto2";

package FMUSync;

import "osi_sensorview.proto";

service externalSync {
  rpc TriggerStep (externalCmd) returns (externalResponse) {}
}

message externalCmd{
  required float stepSize_ms =1;
}

message externalResponse{
  bool status =1;
  osi3.SensorView view =2;
}
```

- **Service externalSync:** Service definition for external synchronization with the FMU.
 - **TriggerStep:** RPC method to trigger a simulation step in the FMU.
- **Message externalCmd:** Message sent to the FMU to trigger a simulation step.

Attribute	Type	Description
stepSize_ms	float	The desired simulation step size in milliseconds.

- **Message externalResponse:** Response received from the FMU after completing the simulation step.

Attribute	Type	Description
status	bool	(Required) Indicates the status of executing the simulation step. True if successful, false otherwise.
view	osi3.SensorView	Optional sensor data from the simulation, using the OSI SensorView format.

- **Develop a gRPC Application:** The first step is to create an application for running scenarios with the desired external simulator.
 - **Implement TriggerStep Method:** The generated files must be included in the external simulator, and an implementation for the TriggerStep function must be added.
- This function should:
 - Contain the simulation stepping function of the simulation tool.
 - Return an "externalReponse" message that includes a status confirmation and, if available, an OSI 3 sensor view.

- **Establish Connection:** The integration connects to an insecure gRPC server at the localhost address on port 50000.
- The following is a pseudo code showing the correct implementation of “TriggerStep”:

None

```
Triggerstep() // gRPC function called by the FMU
begin
  SimulationToolStep() // Execute one step of the simulation tool

  if SensorView is available then
    response.status = true
    response.sensorView = available SensorView
  else
    response.status = false
    response.sensorView = NULL
  endif

  return response
end
```

- **Execution:** Examples of the TriggerStep implementation are provided in the document for both Python and C++ simulation environments.

■ Python external simulation environment

Python

```
import grpc
import message_pb2
import message_pb2_grpc
import osi3

class
```

```

ExternalSyncServicer(message_py_pb2_grpc.externalSyncServicer):

    def TriggerStep(self, request, context):
        print("TriggerStep called")
        time.sleep(0.0000001) # Simulate processing time

        # Create the response
        response = message_py_pb2.externalReponse()
        try:
            response.view = get_current_osi_view() # Function to be
            implemented by the user to return the osi 3 view of the current
            scene
            response.status = True
        except IndexError:
            response.status = False

        Do_step() # User's function to step the simulation

        return response

def run():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))

    message_py_pb2_grpc.add_externalSyncServicer_to_server(ExternalSyncS
    ervicer(), server)
    server.add_insecure_port('[::]:50000')
    server.start()

if __name__ == '__main__':
    run()

```

■ CPP external simulation environment

C/C++

```

#include <grpcpp/grpcpp.h>
#include "message.grpc.pb.h"
#include "message.pb.h"
#include <chrono>

```

```
#include <thread>

class ExternalSyncServiceImpl final : public ExternalSync::Service {
public:
    ::grpc::Status TriggerStep(::grpc::ServerContext* context, const
    ::Empty* request, ::externalReponse* response) override {
        std::cout << "TriggerStep called" << std::endl;

        // Simulate processing time
        std::this_thread::sleep_for(std::chrono::nanoseconds(100));

        bool status = false;
        status = get_currentViewStatus(); // Function implemented by
        the user to get the status of the current scene for the scene to
        proceed or not
        if (status){
            response->set_view(get_currentView()); // get_currentView()
            is a function the user needs to implement to get the osi3 view of
            the current scene
            response->set_status(true);
        }

        Do_step(); //User's function to step the simulation
        return ::grpc::Status::OK;
    }
};

void RunServer() {
    std::string server_address("[::]:50000");
    ExternalSyncServiceImpl service;

    ::grpc::ServerBuilder builder;
    builder.AddListeningPort(server_address,
    grpc::InsecureServerCredentials());
    builder.RegisterService(&service);
    std::unique_ptr<::grpc::Server> server(builder.BuildAndStart());
    std::cout << "Server listening on " << server_address <<
    std::endl;

    server->Wait();
}

int main(int argc, char argv) {
    RunServer();
    return 0;
}
```

- Following these steps the external simulation tool is now successfully integrated with Valeo's virtualized hardware lab achieving the connection shown in [Figure \[25\]](#). Running the

environment with the external simulation tool is similar to the method described above, the only difference is while calling the “./init_virtual_env.sh” and the “./run_virtual_env.sh” you must add “--external”.

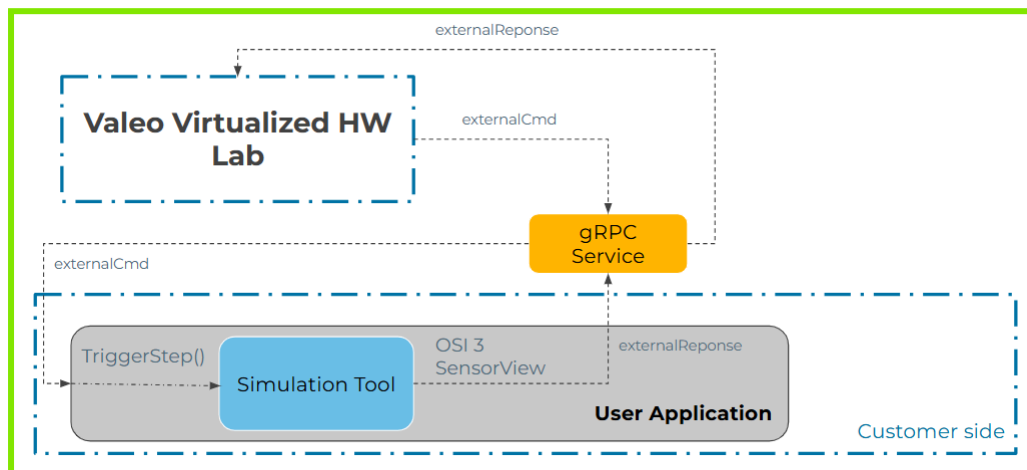


Figure [25]: External Simulation Tool Environment

7.3. ESMINI Simulation Tool

7.3.1. Tool Overview

The Valeo solution is already integrated with the ESMINI tool.

ESMINI Simulator User Interface as shown in [Figure \[26\]](#).



Figure [26]: ESMINI Simulator

The solution with ESMINI tool as shown in [Figure \[27\]](#) currently supports the following architectures:

- Linux x86-64
- Linux ARM-64

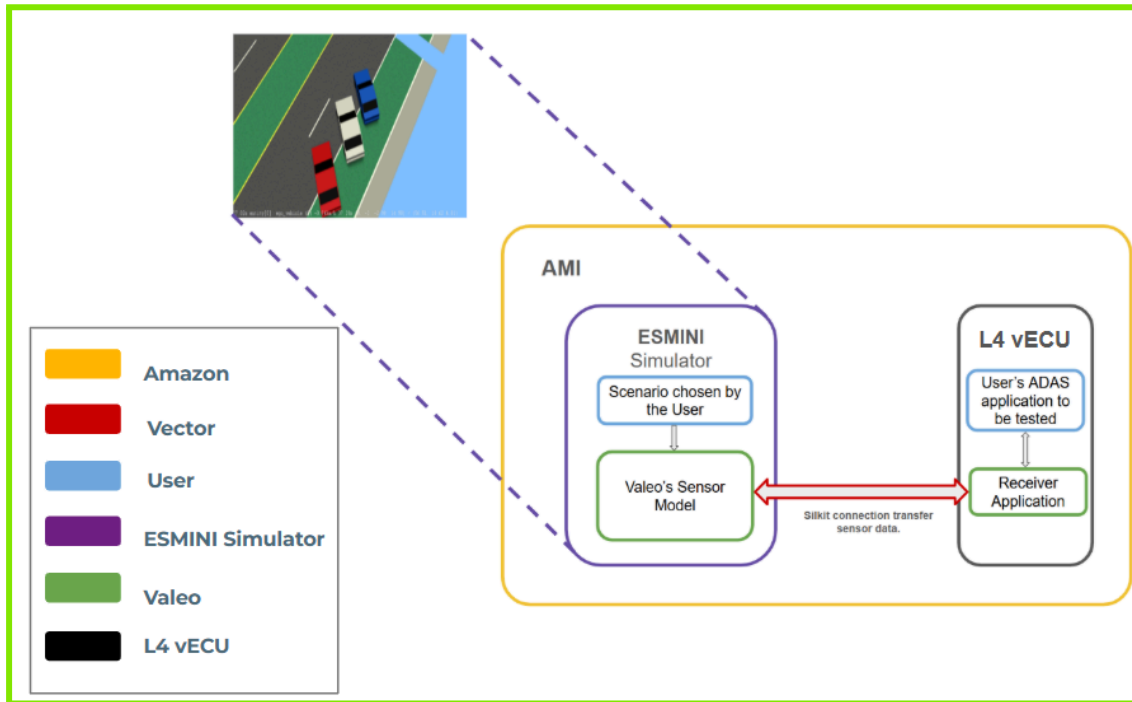


Figure [27]: Solution Environment With ESMINI

7.3.2. Pre-Simulation Setup

Before running a simulation, the environment must be configured by editing the **EsminiUserConfig.json** file. Refer to [§5.1.2 Simulation Configuration File](#).

7.3.3. Running Options

7.3.3.1. Running Using GUI

For information on the running environment, please consult [§6.1 Virtualized Lab GUI](#).

7.3.3.2. Running Using Execution Commands

For information on the running environment, please consult [§6.2 Manual Setup](#).

7.3.4. Running Components

Executing the `run_virtual_env.sh` script initiates several components that communicate with each other to create the complete simulation environment:

- **SIL Kit Communication:** The backbone for inter-component communication.
- **vECU SIL Kit Participant:** Enables the QEMU virtual environment to communicate via Ethernet using SIL Kit. This is achieved using a modified `qemu-adapter` from Vector to allow synchronized simulation and then integrated with the Valeo environment for this specific usage.
- **ESMINI Executable:** The core simulation tool that runs the specified scenario.
- **Valeo's SM:** The integrated Valeo sensor technology that provides data to the simulation.

This communication architecture allows for a synchronized and realistic simulation of the entire system.

The running components communication can be seen in [Figure \[28\]](#).

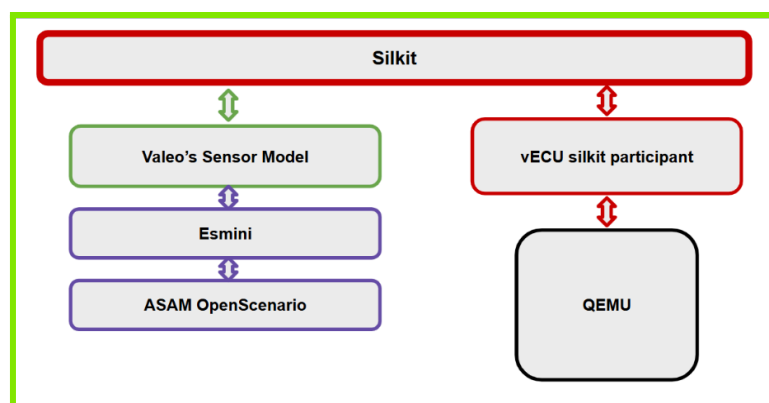


Figure [28]: Silkit Communication With ESMINI

7.3.5. Managing Scenarios

This section explains how to use the provided scenarios and how to create new ones.

7.3.5.1. Using an Existing Scenario

To run one of the pre-built scenarios, simply update the **ScenarioName** parameter in the **EsminiUserConfig.json** file to point to the desired .xosc file located in the [ASAM scenarios directory](#). Refer to [§7.3.2 Pre-simulation Setup](#)

7.3.5.2. Creating a Custom Scenario

To add a custom ASAM OpenSCENARIO, follow these steps:

- **Navigate to**
“/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/Templates_and_Libraries/ASAMScenarios”
- **Create Scenario Files:** Create two files:
 - **<Scenario name>.xosc:**
 - **Location:**
/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/Templates_and_Libraries/ASAMScenarios/xosc
 - **Description:** An XML file that defines the story of the scenario (e.g., vehicle maneuvers, events, conditions). The scenario must follow the ASAM OpenSCENARIO standard.
 - **<Scenario name>.xodr:**
 - **Location:**
/home/ESMINI_VIRTUALIZED_HW_LAB_V1.0/User_workspace/Templates_and_Libraries/ASAMScenarios/xodr
 - **Description:** An OpenDRIVE file that defines the road network, lane markings, and other static elements of the environment's GUI.
- **Update Configuration:** After creating the files, update the ScenarioName in the EsminiUserConfig.json file to point to the new <Scenario name> file.

7.4. Integration With External Tool

This section explains how to integrate Valeo's vECU solution with external simulation tools not supported by Valeo.

7.4.1. Pre-requisites

For details on integrating with gRPC, please refer to the Integration Process section [§7.2 Integration Process](#).

7.4.2. Integrated Environment Initialization

- The execution phase is managed by the “**init_virtual_env.sh**” script, the same script in [§6.2.1 Environment Initialization](#)
- To initialize the solution environment with external simulation tool, run the “init_virtual_env.sh” script as follow:

None

```
./init_virtual_env.sh --external
```

- Alternatively, initialize it as follows to use an alternative configuration file, as illustrated in [§6.2.1.1 Initialization Setup Process](#).

None

```
./init_virtual_env.sh "FULL_PATH_TO_CUSTOM_CONFIG" --external
```

- The “**--external**” option allows the script to bypass the initialization of Valeo's supported simulation tools.

7.4.3. Integrated Environment Running

- The execution phase is managed by the “**run_virtual_env.sh**” script. The same script in [§6.2.2 Environment Running](#)
- The integration with external tools is controlled via the **--external** command-line argument, with different use cases:
 - **Case 1:** External Tool with a Default Script:
 - Navigate to
/home/ESMINI_VIRTUALIZED_HW_LAB_V*/HW-Virtualization-lab/simulation_tools/external
 - Open “**ExternalToolScript.sh**”
 - Edit the script with the necessary commands that needed to initialize and run the external tool.
 - To run an external tool using a default script name (ExternalToolScript.sh), the **--external** flag is used without a specific script path.

None

```
./run_virtual_env.sh --external
```

- The script will look for ExternalToolScript.sh and execute it. This is useful for a primary or standard external tool.
- **Case 2:** External Tool with a Specific Script Name:
 - To run a specific external tool script located in the default external tools directory:

None

```
./run_virtual_env.sh --external my_custom_tool.sh
```

- The script will look for my_custom_tool.sh inside the default external tools directory

(`/home/ESMINI_VIRTUALIZED_HW_LAB_V*/HW-Virtualization-lab/simulation_tools/external`) and execute it. This allows for switching between different pre-defined external tool scripts.

- **Case 3:** External Tool with a Full Script Path:
 - To run an external tool script from any location on the filesystem, provide the full or relative path to the script.

None

```
./run_virtual_env.sh --external /path/to/another/tool.sh
```

- To run an external tool script from any location on the filesystem, provide the full or relative path to the script.
 - The script will resolve the provided path and execute the script from its specified location. This offers maximum flexibility for running tools that are not part of the standard project structure.
- In all cases involving **--external**, the **run_virtual_env.sh** script will start the specified external tool script as the main scenario driver instead of supported tools by Valeo.

Important Note: *Responsibility lies with the individual to close external tool processes, as scripts will reopen them upon rerunning.*

- Document the integration capabilities with various simulation tools

8. Troubleshooting Guide

This section addresses common issues that may arise during the initialization, configuration, and operation of the Valeo Virtualized HW Lab environment.

8.1. AMI Issues

An access issue may occur for users who are not fully aware of the solution's prerequisites.

8.1.1. Describe Instance

8.1.1.1. Issue Description

- Users might encounter multiple warning messages indicating the closure of an instance, which was generated from Valeo solution's AMI. This occurs because a necessary prerequisite—the attachment of required policies that check for encrypted volumes—was not fulfilled.

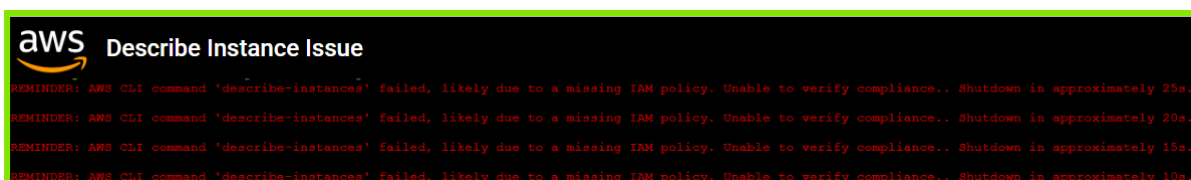


Figure [29]: Describe Instance Issue

8.1.1.2. Resolution Steps

- Make sure to follow the required requirements for the AMI and start the instance again. Refer to [S4.2 IAM Role Creation](#)

8.1.2. Metadata Configuration

8.1.2.1. Issue Description

- Users might encounter multiple warning messages indicating the closure of an instance, which was generated from Valeo solution's AMI. This occurs because a necessary prerequisite—Meta data configuration—was not fulfilled.

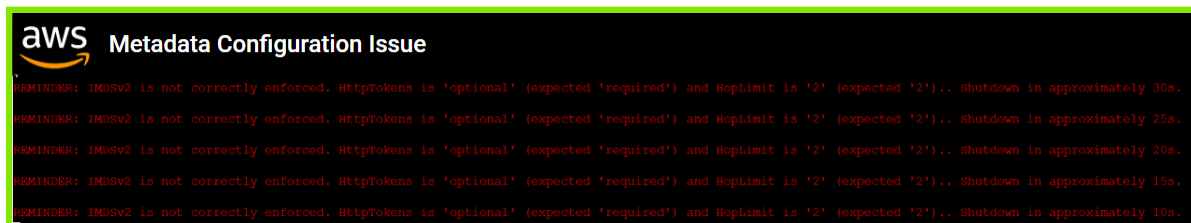


Figure [30]: Metadata configuration Issue

8.1.2.2. Resolution Steps

- Make sure to follow the required requirements for the AMI and start the instance again. Refer to Meta data configuration in [§4.4 Configuration Process](#)

8.2. AWS SSO Issues

An access issue may occur for users who are not fully aware of the solution's prerequisites.

8.2.1. Authentication Issues

8.2.1.1. Issue Description

- After installing the AWS CLI, you might encounter an error where running any aws command fails with a message like: **/usr/bin/aws: No such file or directory.**
- This error typically occurs for one of two reasons:
 - **Stale Command Cache:** The shell has cached the location of a previous aws installation and has not updated it to the new one.

- **Incorrect PATH:** The directory where the new AWS CLI was installed (/usr/local/bin) is not included in your shell's PATH environment variable. The PATH tells the system where to find executable programs.

8.2.1.2. Resolution Steps

Follow these steps to resolve the issue. Start with Step 1, as it is the most common and simplest solution.

- **Step 1: Clear the Shell's Command Cache**

The first step is to clear the shell's cache of command locations.

1. Run the following command in your terminal

```
None  
hash -r
```

2. Try to verify the AWS CLI version again:

```
None  
aws --version
```

If the command now works and displays the correct version, the issue is resolved. If it still fails, proceed to Step 2.

- **Step 2: Add the AWS CLI to Your PATH**

If the command is still not found, its location must be permanently added to the shell's PATH.

1. Add the export command to your shell's startup file. For most users on Linux or WSL, this file is ~/.bashrc. Run the following command to append the necessary line to the end of the file:

```
None  
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bashrc
```

2. Apply the changes to your current terminal session by running the source command. This reloads the startup file without requiring you to log out.

None

```
source ~/.bashrc
```

8.3. Troubleshooting via Execution Commands

This applies to errors encountered when running the **init_virtual_env.sh** script.

This applies to errors encountered when running the **init_virtual_env.sh** script.

8.3.1. Init Troubleshooting

This applies to errors encountered when running the **init_virtual_env.sh** script.

8.3.1.1. Common Issues

- An error message appears on the console.
- The error details are also logged in the `init_logs` directory
- **init_logs** located in `/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/`

8.3.1.2. Resolution Steps

To fix the issue, run the cleanup script using the following command: supports protobuf code generation.

None

```
./clean_env.sh
```

8.3.2. Valeo's vECU Troubleshooting

This section covers issues related to accessing and using Valeo's vECU.

8.3.2.1. Access Issues

- **Problem:** Unable to access Valeo's L4 vECU.
- **Resolution:** Use the same troubleshooting steps as [§8.3.1 Init Troubleshooting](#); run the cleanup script:

```
None  
./clean_env.sh
```

8.3.2.2. Linking Errors

- **Problem:** "Undefined Symbols" error appears in your ADAS application.
- **Resolution:** Ensure you have exported the library path for libDFTReceiver.so and libgtest.so.* before executing the application with the command:

```
None  
LD_LIBRARY_PATH=<libDFTReceiver.so and libgtest.so.* path>:$LD_LIBRARY_PATH
```

- Recheck the application to confirm you are using the correct APIs and that they are implemented correctly.

8.3.3. Running Troubleshooting

This applies to errors that occur while executing the run script (**run_virtual_env.sh**).

8.3.3.1. Runtime Issues

- An error message appears on the console during the execution of the run script.
- The error details are also logged in the run_logs directory.

- **run_logs** located in
/home/ESMINI_VIRTUALIZED_HW_LAB_V*/User_workspace/

8.3.3.2. Recovery Procedures

- First, simply try to execute the run script again.
- If the issue persists, follow the [§8.3.1 Init Troubleshooting](#) steps (i.e., run ./clean_env.sh) and then execute the run script again. simulation tools)

Important Note: If these troubleshooting methods do not resolve your issue, contact Valeo support for further assistance

8.4. Troubleshooting via Virtualized Lab

This applies to errors encountered when using the Virtualized Lab application

8.4.1. Run Simulation Troubleshooting

This applies to errors encountered when starting the simulation with the "Run Simulation" button.

8.4.1.1. Common Issues

- The simulation tool might crash, leaving the run in a hanging state, and an error status will be shown in the status terminal.

8.4.1.2. Resolution Steps

To fix the issue, you'll have to press the "**Initialize Environment**" button and wait for the environment cleaning. Then, press "**Run Simulation**", and this time the issue should be fixed.



Figure [31]: Troubleshooting Runs in the Virtualized Lab

Important Note: If these troubleshooting methods do not resolve your issue, contact Valeo support for further assistance

9. Open-Source Tools and License

The following table shows all open source tools used by Valeo Virtualized Hardware Lab solution, version used, component purpose, and license type of this component.

Used Open Source Tool	Version	Component Purpose	License Type
esmini/esmini	2.46.0	Provides the usage of ESMINI simulation tools by the solution.	MPL-2.0
open-simulation-interface	3.5.0	Open Simulation Interface (OSI) provides a common language for simulated sensors and environments so that tools and models from different vendors can interoperate.	MPL-2.0
Protobuf	3.20.2	Serialization of OSI message while handling the payloads requests and responses between vECUs, services and tools.	BSD-3-Clause
OpenSimulationInterface/osi-sensor-model-packaging	1.5.0	Integrate the FMU with the Open Simulation Interface	MPL-2.0

Used Open Source Tool	Version	Component Purpose	License Type
		(OSI) to enable it to parse OSI frames.	
JSON for Modern C++	3.11.3	Used for parsing every sensor configuration from a JSON file ("config.json"). Using JSON file was preferred as it is a simple yet efficient user-friendly way to configure.	MIT
muflihun/easyloggin gpp	9.97.1	A lightweight C++ logging library is employed to provide straightforward and efficient logging capabilities for the internal components of the L3 vECU.	MIT
spdlog	1.15.2	Leveraging Spdlog, a fast C++ logging library, we ensure high-performance logging essential for robust integration between all solution modules.	MIT
modelica/fmi-stand ard	2.0.4	The FMI standard defines a standard interface for model exchange and co-simulation. It is used to enable the FMU	BSD-2-Clause & Creative Commons

Used Open Source Tool	Version	Component Purpose	License Type
		functionality, allowing the system to be packaged and run as an FMU for simulation in compliant tools. The presence of FMU-related files and model descriptions confirms this usage.	Attribution 4.0 License Text (Can be Neglected as component used as Code and Schema only)
aws-labs/aws-sdk-cpp	1.11.603	This component provides an integration library for AWS License Manager. The solution can call its validation function to check whether customers have a valid license before granting solution to operation, ensuring license compliance.	Apache-2.0
grpc	1.61.3	Provides remote procedure call (RPC) capabilities, used for communication between simulation components to transfer protobuf serialization.	Apache-2.0
jbeder/yaml-cpp	0.7.0	Handles YAML configuration files parsing for sil-kit interface configurations	MIT

Used Open Source Tool	Version	Component Purpose	License Type
NTNU-IHB/FMI4cpp	0.8.3	Provides FMI support for running and interacting with FMUs.	MIT
seladb/PcapPlusPlus	22.05	Create a complete Ethernet frame that encapsulates a SOME/IP message within a UDP/IP packet to send it using the sil-kit.	Unlicense
vectorgrp/sil-kit	4.0.50	- Provides simulation bus and communication infrastructure for sil-kit adapter only. - Provides simulation bus and communication infrastructure for the entire solution except the sil-kit adapter.	MIT
vectorgrp/sil-kit-adapters-qemu	1.2.0	Bridges QEMU-based virtual hardware with the SilKit co-simulation environment.	MIT

Used Open Source Tool	Version	Component Purpose	License Type
jammy-server-cloud img-arm64.img	20251017	QEMU-based virtual hardware for L4-vECU <u>Note: This component used to generate the QEMU image to act as level 4 virtualization so it can be considered as Separate Work and it is also used without code modification</u>	LGPL-3.0-only
Google C++ Testing Framework	1.11.0	gtest is used as a powerful framework to validate the ADAS application's behavior under various simulated conditions, from data integrity checks (like timestamps and live counters) to complex dynamic scenarios.	BSD-3-Clause

Each license type has certain permissions, restrictions, and obligations. They are described in the following table.

License Type	License Permits	License Forbids	License Obligations
MPL-2.0	Commercial use, modification, distribution, Patent Claims, Place Warranty, and Sub-License	Hold Liable , User Trademarks, Patent Retaliation	Including license text
BSD-3-Clause	Commercial use, modification, distribute, Disclose Source, and place Warranty.	Hold Liable , User Trademarks, Place Additional Restrictions	Including license text
MIT	Commercial/private use, modification, distribute, Disclose Source, and Sub License.	Hold Liable, Place Additional Restrictions	Including license text
BSD-2-Clause	Commercial use, modification, Disclosure Source, Place Warranty ,and distribution.	Hold Liable, Place Additional Restrictions	Including license text

License Type	License Permits	License Forbids	License Obligations
Apache-2.0	Commercial/Private use, Modification, Distribution, Patent Claims, Disclosure Source, Place Warranty, Place Additional Restrictions and Sub-License.	Hold Liable, User Trademarks, Patent Retaliation	Including license text
LGPL-3.0-only (Separate Work ONLY without code modification)	Commercial use, modification, distribution , patent Claims, and Place Warranty.	Anti DRM Provision, Sub-License, Fees, Hold Liable, Patent Retaliation	Including license text
Unlicense	Disclose Source	Place Additional Restrictions	Including license text