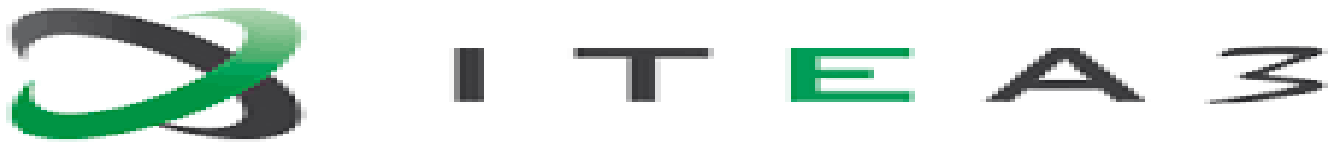


# Kuksa User Manual.

---



SI No.	Author	Version	Date	verified by	Remark
1.	Kirubel	0.1	28.03.2019	Nill	No remarks

# Contents

---

## Introduction

1. Introduction To Kuksa.....	5
-------------------------------	---

## 2. Eclipse Kuksa Ecosystem

2.1 Eclipse Kuksa.....	6
------------------------	---

## 3. Architecture

3.1 In-Vehicle platform.....	8
3.1.1 Core Layer.....	8
3.1.2 API / Binding Layer.....	8
3.1.3 Application Layer.....	8
3.2 In-vehicle connectivity.....	9
3.2.1 Protocols.....	9

## 4. Architectural Overview

4.1 Ex-vehicle connectivity concept.....	11
4.2 App Runtime concept.....	13
4.3 Automotive API concept.....	13
4.3.1 AUTOSAR.....	14
4.4 Apps concept.....	15
4.5 Device Management Client concept.....	15
4.6 Operating System (OS) concept.....	17

## 5. The 5G-Infrastructure

5.1 Evolved Packet Core (EPC) concept.....	20
5.1.1 eNodeB (Baseband Unit) (BBU) concept.....	21
5.1.2 eNodeB (Remote Radio Units) (RRU) concept.....	21

## 6. Cloud back-end details

6.1 Message Gateway concept.....	22
6.2 Device Management Backend concept.....	23

6.3 Report Generation concept.....	23
6.4 Marketplace Backend concept.....	24
6.5 Visualization concept.....	25
6.6 Marketplace Frontend concept.....	26
6.7 Data Management concept.....	27
6.8 Identity Management concept.....	28
6.9 Device Representation concept.....	29
6.10 Big Data Analysis concept.....	29
6.11 Core Services concept.....	30
6.12 Domain-specific Services concept.....	30

## 7. Kuksa-IDE Building & Deploying

7.1 Version.....	32
7.2 IDE.....	32
7.2.1 kuksa.apps.....	34
7.2.2 kuksa appstore.....	34
7.3 Getting started with the In-Vehicle platform.....	34
7.3.1 Kuksa App example.....	34

## 8. Cloud back-end

8.1 Kuksa Cloud Deployment.....	36
8.2 Getting started with Kuksa Appstore.....	36

## 9. Kuksa In-Vehicle

# 1. Introduction To Kuksa

Because today's software-intensive automotive systems are still developed in silos by each car manufacturer or OEM in-house, long-term challenges in the industry are yet unresolved. Establishing a standard for car-to-cloud scenarios significantly improves comprehensive domain-related development activities and opens the market to external applications, service provider, and the use of open source software wherever possible without compromising security. Connectivity, OTA maintenance, automated driving, electric mobility, and related approaches increasingly demand technical innovations applicable across automotive players.

The open and secure Eclipse Kuksa project will contain a cloud platform that interconnects a wide range of vehicles to the cloud via in-car and internet connections. This platform will be supported by an integrated open source software development environment including technologies to cope especially with software challenges for vehicles designed in the IoT, Cloud, and digital era.

This ecosystem will provide a comprehensive environment across various frameworks and technologies for *the in-vehicle platform, the cloud platform, and an app development IDE* - that is, the complete tooling stack for the connected vehicle domain see [Figure 1](#) below. Essential to this environment will be the capabilities for collecting, storing, and analysing vehicle data in the cloud as well as the transmission of diverse information such as cloud calculation results (e.g. improved routing), software maintenance updates or even complete new applications. While many IoT solutions exist in the Eclipse IoT ecosystem, Eclipse Kuksa combines the necessary existing technologies and fills the gaps for the specific requirements of the connected embedded real-time nature of the automotive domain.[see here for more details](#).



(Source: <https://www.eclipse.org/kuksa/about/EKuksa.png> )

Figure 1. Kuksa in-vehicle platform, cloud platform, and app development IDE.

## 2. Eclipse Kuksa Ecosystem

### 2.1 Eclipse Kuksa

#### Kuksa IDE

Based on Eclipse Che

Allows Cloud and In-Vehicle Application development

Platform independent

Shared workspaces

Almost configuration free

Docker-based: VPN planned to allow remote / network independent cross compilation

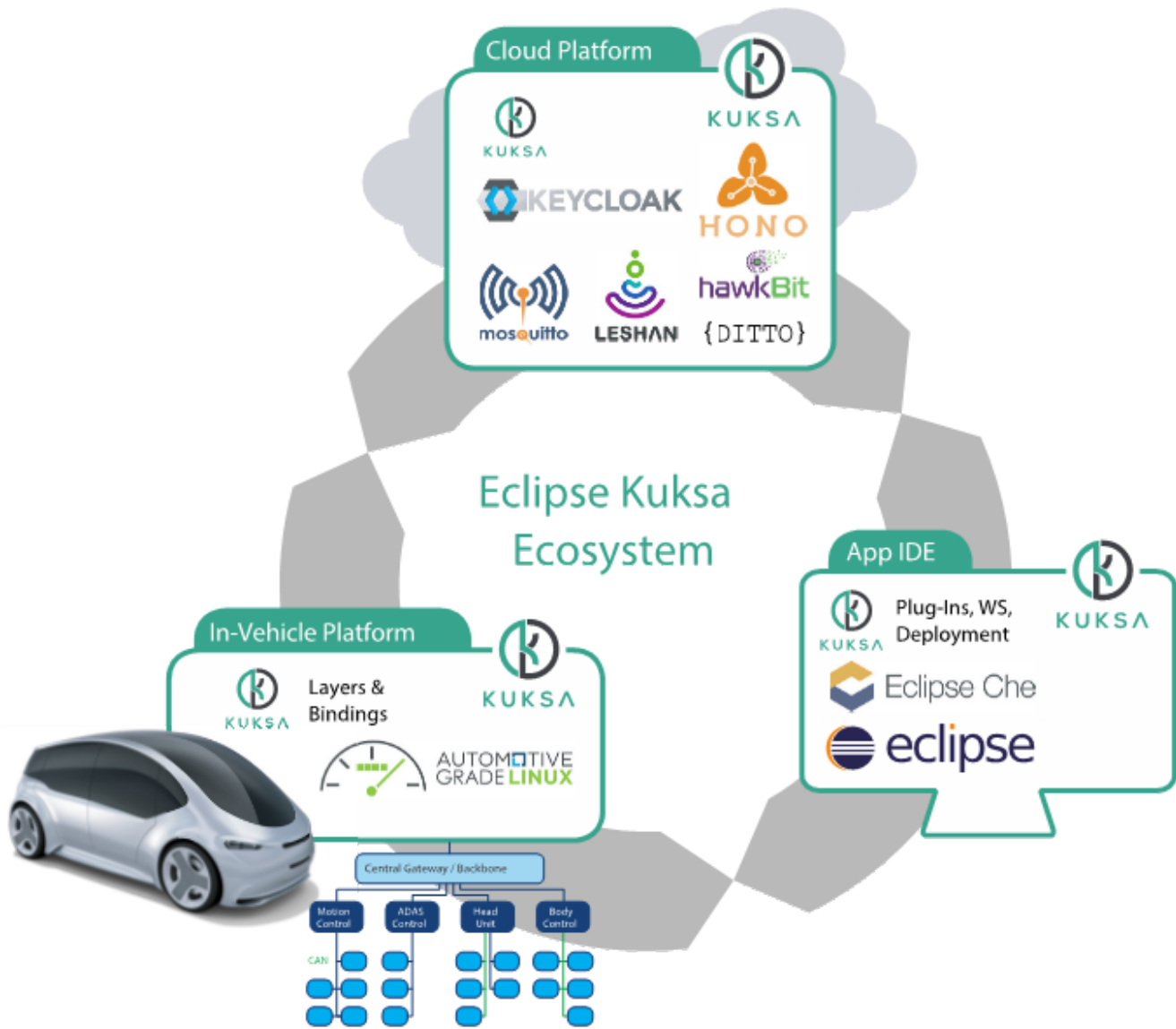


Figure 1. Eclipse Kuksa Ecosystem.

(source:[https://www.researchgate.net/profile/Marco\\_Wagner2/publication/330281127\\_Innovation\\_through\\_Openness\\_-\\_The\\_Open\\_Source\\_Connected\\_Vehicle\\_Framework\\_Eclipse\\_Kuksa/links/5c371b5892851c22a3691df8/Innovation-through-Openness-The-Open-Source-Connected-Vehicle-Framework-Eclipse-Kuksa.pdf?origin=publication\\_detail](https://www.researchgate.net/profile/Marco_Wagner2/publication/330281127_Innovation_through_Openness_-_The_Open_Source_Connected_Vehicle_Framework_Eclipse_Kuksa/links/5c371b5892851c22a3691df8/Innovation-through-Openness-The-Open-Source-Connected-Vehicle-Framework-Eclipse-Kuksa.pdf?origin=publication_detail))

### 3. Architecture

The overall platform of the security-relevant aspects of the APPSTACLE environment architecture is divided into three building blocks:

1. In-vehicle platform,
2. 5G Infrastructure, and
3. Cloud back-end.

The above mentioned building blocks comprises a set of components which communicates between components of the same building block and/or components of different blocks.

## 3. 1. In-Vehicle Platform

The APPSTACLE environment is created to provide add-on services to the connected vehicles. This provides a complete functionality of the vehicles through deploying the Apps on the in-vehicle platform. Therefore, three layers of components are required to enable this purpose:

**3.1.1 Core Layer:** Contains the in-vehicle platform components, such as operating system and application runtime. It, furthermore, allows the vehicle owner to interact with the vehicle, e.g., via smartphone access. In addition, it provides an interface to the 5G infrastructure similar to the core layer of the cloud back-end.

**3.1.2 API / Binding Layer:** consists of relevant APIs and components for internal and external communication.

**3.1.3 Application Layer:** It represents the arrangement of all Apps that are running within the in-vehicle platform. Some of the Eclipse base Open source solutions to inreach kuksa components are:

- Automotive Grade Linux (AGL)
- Eclipse hawkBit
- Eclipse Hono
- Eclipse Ditto
- Eclipse Che
- Keycloak
- ...



Figure 2. Eclipse base Open source solutions to inreach kuksa components

(source:[https://www.researchgate.net/profile/Marco\\_Wagner2/publication/330281127\\_Innovation\\_through\\_Openness\\_-\\_The\\_Open\\_Source\\_Connected\\_Vehicle\\_Framework\\_Eclipse\\_Kuksa/links/5c371b5892851c22a3691df8/Innovation-through-Openness-The-Open-Source-Connected-Vehicle-Framework-Eclipse-Kuksa.pdf?origin=publication\\_detail](https://www.researchgate.net/profile/Marco_Wagner2/publication/330281127_Innovation_through_Openness_-_The_Open_Source_Connected_Vehicle_Framework_Eclipse_Kuksa/links/5c371b5892851c22a3691df8/Innovation-through-Openness-The-Open-Source-Connected-Vehicle-Framework-Eclipse-Kuksa.pdf?origin=publication_detail))

The in-vehicle platform additionally provides means for retrieving telemetry data collected by the vehicle itself as well as a human machine interface (HMI ) for user interaction.

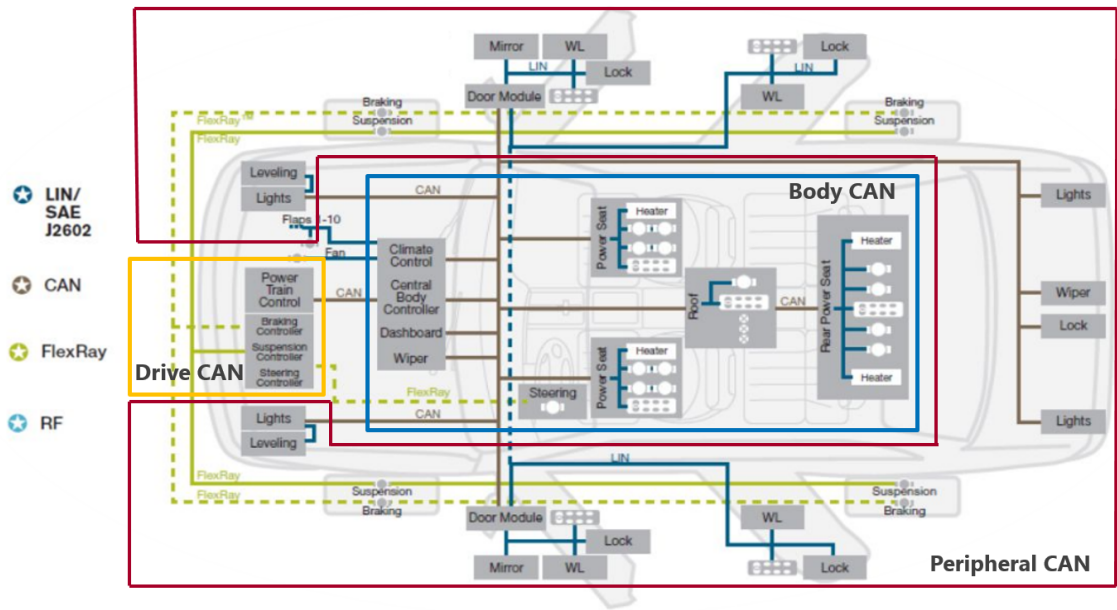
## 3.2 In-vehicle connectivity :



This section provides an overview of the communication protocols that are currently used in the existing automotive architectures as well as their interconnections in the Electrical / Electronic (E/E) in-vehicle architecture. The scope of these protocols defines the in-vehicle communication interfaces for the APPSTACLE platform.

### 3.2.1 Protocols

Automotive protocols are classified by the Society of Automotive Engineers (SAE) into four categories according to the transmission rate and their role in the automotive architecture. Specifically, Class A defines the protocols that are used for convenience systems (e.g. lighting, windows, seatcontrols) and require inexpensive, low-speed communication. Class B defines the protocols supporting instrument cluster or vehicle speed communication and require medium-speed communication. Furthermore, Class C is defined for real-time control ECUs such as the engine, braking and steer-by-wire and require high-speed communication. Finally, telematics systems usually require higher communication speed for multimedia (audio / video) and navigation, and therefore SAE defined the additional Class D communications. All four protocol Class categories are illustrated in [Table 1](#) along with the protocols that belong to each category and are used for in-vehicle communication in terms of their characteristics.



Firugre 3. Automotive Network.

Table 1.: Characteristics of the communication protocols

Bus	LIN	CAN	CAN FD	FlexRay	MOST	Automotive Ethernet
-----	-----	-----	--------	---------	------	---------------------

Bus	LIN	CAN	CAN FD	FlexRay	MOST	Automotive Ethernet
Used in						
Application domains				Hard real-time	Multimedia	Multimedia
Message transmission	Subnets	Soft real-time		Chassis, Powertrain	Multimedia and	Telematics and active
Access control	Body Soft Synchronous Polling 20 kbps A	Powertrain, Chassis Asynchronous CSMA/CA 1 Mbps BC	Soft real-time a a CSMA/CA 10 Mbps D	Synchronous and Asynchronous TDMA 10 Mbps D	Telematics Synchronous and Asynchronous CSMA/CA 24Mbps D	safety Synchronous and Asynchronous CSMA/CD 100Mbps D
Maximum Data Rate						
Protocol Class						

## 4. Architectural Overview

Modern automotive embedded systems consist of several subsystems, which are comprised of one or several Electronic Control Units (ECUs). In turn, the ECUs are made up of a micro-controller and a set of sensors and actuators. They are able to communicate through the transmission of electronic or optical signals through a dedicated communication unit. The subsystems that rely on network communication in automotive systems are divided into five main categories: power train, chassis, body, HMI, and telematics (illustrated in [Figure 3](#)). Each subsystem uses a different protocol to communicate, which is selected based on the architectural requirements and the subsystem functionality. Specifically, the powertrain domain is related to the systems that participate in the longitudinal propulsion of the vehicle, including engine, transmission and all subsidiary components. This domain is supported by a dedicated subsystem called Drive CAN using the Controller Area Network (CAN) for data exchange. The chassis domain refers to the four wheels and their relative position and movement; in this domain the systems are mainly steering and braking. In this subsystem category we find two protocols that are used for high-critical communication, namely CAN and FlexRay, as well as the Local Interconnect Network (LIN) for the lower critical functionalities (e.g. door locking, window raising / lowering). According to the EAST-EEA 1 project definition the body domain includes the entities that do not belong to the vehicle dynamics (i.e., being those that support the car’s user) such as airbags, wipers, lighting, etc. Today’s cars sometimes use two CAN buses (peripheral CAN and body CAN) which interconnect the ECUs of the comfort domain. The telematics domain includes the equipment allowing information exchange between electronic systems and the driver (displays and switches). Such interactions are possible through the infotainment subsystem that is supported by the MOST protocol. Finally additional peripheral systems (e.g., cameras) allow the in-vehicle system to monitor and extract information from its physical environment through the use of Automotive Ethernet technologies. All the aforementioned systems are able to exchange data through a central gateway ([Figure 3](#)) that is able to map (through packet encapsulation) or forward messages from one subsystem to another.

ID	Property
1.	It enables communication with the infotainment unit or the ECUs internal to the vehicle
2.	<ul style="list-style-type: none"><li>• Telemetry data or data specified in the Cloud platform related to internal vehicle functionalities.</li><li>• System metadata (which components are present, heartbeats etc)</li></ul>

ID	Property
3.	<ul style="list-style-type: none"> <li>Interacts with the in-vehicle HW for data gathering and with the ex-vehicle connectivity for data forwarding</li> </ul>
4.	<ul style="list-style-type: none"> <li>One component per vehicle with different interfaces according to the employed protocols.</li> </ul>
5.	<ul style="list-style-type: none"> <li>Information about the protocols used in the internal vehicle architecture</li> <li>Depending on the chosen protocols: what components are communicating what information</li> <li>Development of software modules for handling data for each protocol</li> </ul>

## 4.1 Ex-vehicle connectivity concept:

The technology evolution in the automotive vehicles contributed to the demands for smarter mobility solutions. These solutions are focused on several types of V2X communication:

- Vehicle-2-Vehicle (V2V)
- Vehicle-2-Infrastructure/Infrastructure-2-Vehicle (V2I,I2V)
- Vehicle-2-Pedestrian (V2P) / Pedestrian-2-Vehicle (P2V)
- Vehicle-2-Network (V2N) / Network-2-Vehicle (N2V), 5) Infrastructure-2-Network (I2N) / Network-2-Infrastructure (N2I).

These types along with their interactions are demonstrated in [Figure 4](#).

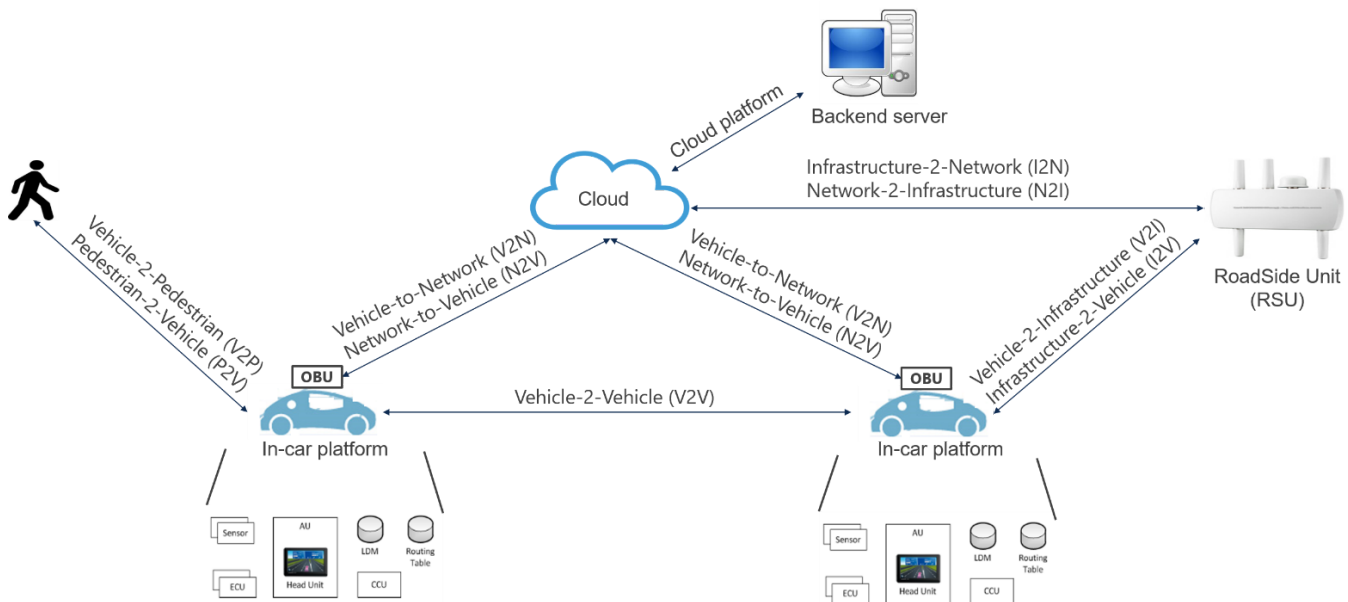


Figure 4. V2X communication types

The units supporting V2X communication are:

- **RoadSide unit (RSU):** It is connected to road sensors (e.g. induction loops, cameras) and a local control center, such that it performs actions or exchanges critical information other vehicles or servers about road or traffic management.

- *OnBoard unit (OBU)*: The on-board unit (OBU) is a radio built-in vehicle device mounted on each vehicle that transmits vehicle data (i.e. identification and location) to a transponder. The OBU itself is a transponder, that is, a data exchange takes place automatically and only on request of one of the participating devices. It allows Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I, I2V) communications with other OBUs or RSUs.
- *Backend server* : It is composed by a PKI, traffic management and roadside unit management servers, all accessible via the RSU's or cellular base stations. In order to facilitate this evolution a couple of solutions were defined that are split into 3 main categories:
- *5G radio access technologies*: This technology provides wide area, broadband access. The 5G technology is currently in the process of conceptual development and standardization by the World Radiocommunication Conference (WRC). The 5G technology is expected to have a specific V2X aspect of the 5G technology in a practical scale after 2020. However, in this document we are leveraging the limited standardization to illustrate conceptually its main scope and architectural view.
- *Pre-5G radio access technologies*: Multiple cellular technologies were identified by the ETSI 3rd Generation Partnership Project (3GPP), LoRa Alliance and other organizations, such as Narrowband IoT , Long Term Evolution for Machines (LTE-M), LoRA are considered. Even though these technologies are already used in V2P/P2V, the main challenge when adopting them in other V2X communication types are reliability and safety, which are currently not addressed in the scope of Low-Power Wide Area Networks (LPWAN).
- *Non-cellular technologies providing wireless access*: IEEE has defined different standards for wireless communication, such as 802.11ac and 802.11p, however only 802.11p is flexible in terms of throughput and offers higher reliability, even though its maximal throughput is more limited than 802.11ac (from 3 to 27 Mbps raw data rate). The reason behind this is that 802.11p was designed particularly for for safety-related Vehicular Ad-hoc NETWORKS (VANET), including the V2V and V2I/I2V concepts. IEEE 802.11p technology is currently fully specified and already deployed in different locations. The following paragraphs start with a description of the scenarios supported by 802.11p communication and cellular communication. This is followed by a description on both the 802.11p and 5G technologies. In the scope of this section we focus on these two technologies, because, to the best of our knowledge, they are considered as the leading candidates for V2X communication.

 For detail information

ID	Property
1.	It enables outward and inward communication between the vehicle and the external entity
2.	<ul style="list-style-type: none"> <li>• No data processing as such.</li> <li>• Re-Packs the data received from bus to appropriate format for the external entity and vice versa.</li> </ul>
3.	<ul style="list-style-type: none"> <li>• It has the direct connection to the BUS and no direct user interaction.</li> </ul>
4.	<ul style="list-style-type: none"> <li>• There will be multiple instances in the minimum two cases.</li> </ul>
5.	<ul style="list-style-type: none"> <li>• Driver for hardware component</li> <li>• (Since Development Stage) Need manual configuration at the moment for 5G mm Radio.</li> </ul>

## 4.2 App Runtime concept:

ID	Property
----	----------

ID	Property
1.	The APP Runtime provides the environment for executing APPs and starts / stops APPs. It has to provide and control resources for the APP, enforce access control (permissions), and isolate APPs from each other.
2.	APPs, configuration data (permissions, options, ...), APP data
3.	The APP Runtime permits or denies communication between APPs, or APP and backend (depending on "the policy"). The APP Runtime obtains APPs from the marketplace. It can be configured by the OEM and/or the vehicle owner (via backend and/or an in-vehicle user interface; probably also by devices [with authorization]).
4.	There is one APP Runtime per in-vehicle platform. It could be part of the operating system.
5.	Initialisation / start up: The APP Runtime is started during the (secure) boot process. It can be configured by the OEM and the vehicle owner (details are left open in this document), e.g., to configure permissions ("the policy").

### 4.3 Automotive API concept:

The Application Programming Interface (API) for vehicles are introduced and discussed in here. The automotive API's try to achieve (a) merging the potentially very complex device and network structure of a car into a single virtual device and (b) hiding the differences between manufacturers, models and makes behind a common interface. On the other hand these interfaces strongly differ in their scope (data-subset or use-case), technological approach and creators.

#### 4.3.1 AUTOSAR

AUTomotive Open System ARchitecture (AUTOSAR) is a cooperation between car manufacturers, OEMs and tool manufacturers and defines a software development paradigm for Electronic Control Units (ECUs) in the automotive domain. In order to separate the development process of application software from the chosen ECU hardware platform, AUTOSAR is introducing a layer model with the three layers Application Software, Runtime Environment and Basic Software (illustrated IN [Figure 5](#)).

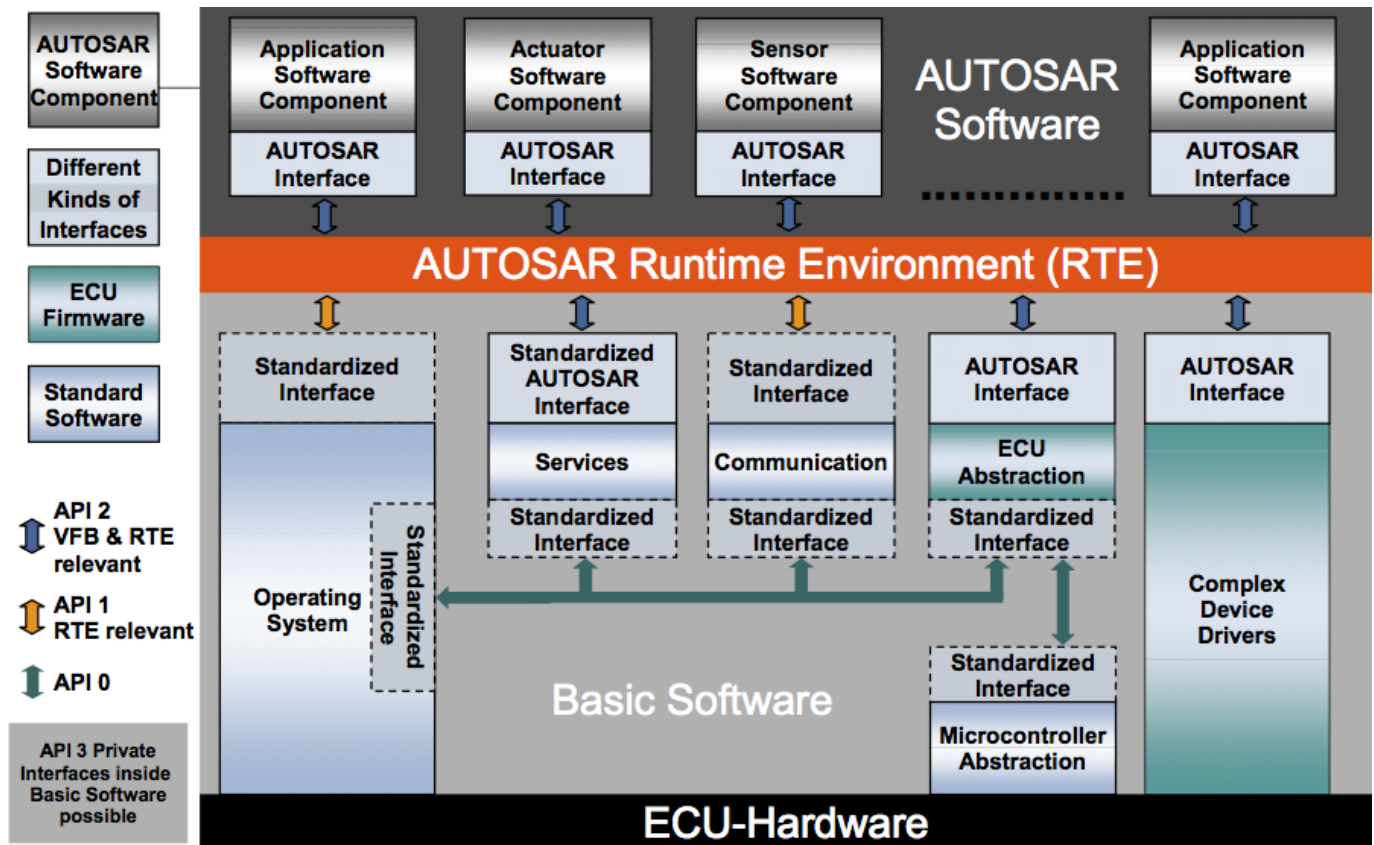



Figure 4. Autosar Layers

The top layer is formed by the application software. It is divided into software components, each of which realizes a part of the application and can consume and provide data via so-called ports. Any communication that does not take place via port connections is forbidden. A port is classified via a port interface (here referred to as interface). Two ports can only be connected to each other if both ports use compatible interfaces. Two important communication paradigms, that are selected by interfaces, are client-server and sender-receiver communication. For client-server communication, a server component provides functions (C, C++) which can be called by clients. A 1:n communication is also possible (i. e. a server can provide its functionality to several clients). In sender-receiver communication, a sender provides data that can be consumed by receiver components. Both 1:n and m:1 communication is possible here (i.e. a date can be consumed by several components or several senders provide a date for one receiver concurrently). Many-to-many communication is not provided.

The lower layer consists of the basic software and contains the hardware drivers, the operating system and the communication stack. The communication stack handles communication from and to other ECUs that are connected via network interfaces like CAN, LIN, Flexray, automotive Ethernet, etc.

All communication, whether between software components on the upper level or between software components and basic software on the lower level, is realized via the runtime environment (RTE), which forms the middle layer. The RTE specification document defines a schema for API functions (C, C++), which are usually generated by code generators of the AUTOSAR modeling tools according to the modeled communication between software components and basic software. All communication must take place via the (generated) API functions. Other communication is not permitted. Likewise, all communication interfaces must be defined at the time of development, which makes it impossible to dynamically extend the software architecture at runtime.

 For detail information

## ID Property

**ID    Property**

- 
1. The Automotive API provides an interface to in-vehicle data for APPs (and cloud services? other entities??). An (operating system) service ("Automotive API server") implements the Automotive API, for instance similar to the "Vehicle Information Service" specified by the W3C.

---

  2. Vehicle data (sensor data, diagnosis information, configuration of vehicle components, vehicle status information, ...)

---

  3. It can be used by APPs (and cloud services, etc.) to retrieve information from the vehicle, to send data to in-vehicle components, and to write (vehicle) configuration data.

---

  4. There is one Automotive API per in-vehicle platform.

---

  5. Initialisation / start up: The "Automotive API" service is started during the (secure) boot process.

---

## 4.4 Apps concept:

**ID    Property**

- 
1. (In-vehicle) APPs are programs that provide new features to the vehicle.

---

  2. App data (depends on APP / use case)

---

  3. APPs are executed and controlled by the APP Runtime and can access in-vehicle data via the Automotive API. They can communicate with other entities via connectivity APIs (cf. architecture picture). If permitted (by "the policy"), they might communicate with cloud services (backend providers) and other APPs, and they could interact with the driver via a GUI (if available). APPs can access resources via the APP Runtime (subject to "the policy"). A user can install APPs from the Marketplace in the vehicle.

---

  4. There can be multiple APPs per in-vehicle platform.

---

  5. Initialisation / start up: APPs are started by the APP Runtime. Configuration data depends on the APP / use case.

---

## 4.5 Device Management Client concept:

**ID    Property**

- 
1. The DM is responsible for keeping the devices compliant to the whole system landscape. This starts with building up a base in communication and process protocols. It is also providing features for securely enrolling new devices, governing and configuring them while being out in the field, monitoring and debugging their behavior remotely and maintaining the devices with software updates. Four subjects can be distinguished:
 

---

    - Enrollment includes provisioning and authentication for bringing new devices into an IoT landscape. The authentication assures that only trust-worthy devices are added to the network and connected to cloud services. Also only authorized users should be able to bring in new devices and gain access according to their roles granted.

---

    - Governing contains features for controlling and configuring devices. As IT systems are often under a constant development, some parts change and so do some of their constants, for example network addresses or ports.

---

ID	Property
	<ul style="list-style-type: none"> <li>Monitoring keeps an eye on all components of the system and their status. Reports and alerts for incidents are raised and logged. An on-time awareness of system issues is enabled. For diagnosis and solving software bugs it is also imperative to load log dumps remotely from devices.</li> </ul>
	<ul style="list-style-type: none"> <li>Maintenance with the ability to distribute and apply software updates is the fourth subject. The device management assures that the update is delivered and applied to the device according to the present constraints. Feedback mechanisms answer back to the cloud for a detailed status of the update.</li> </ul>
2.	<b>Process</b>
	<ul style="list-style-type: none"> <li>System states</li> </ul>
	<ul style="list-style-type: none"> <li>Application states</li> </ul>
	<ul style="list-style-type: none"> <li>Update states</li> </ul>
	<ul style="list-style-type: none"> <li>Management Calls</li> </ul>
	<ul style="list-style-type: none"> <li>Push Calls</li> </ul>
	<ul style="list-style-type: none"> <li>Alarms</li> </ul>
	<ul style="list-style-type: none"> <li>Enrollment policies</li> </ul>
	<ul style="list-style-type: none"> <li>Updates, Update scheduling</li> </ul>
	<ul style="list-style-type: none"> <li>Inventory Hardware/Software listings</li> </ul>
	<ul style="list-style-type: none"> <li>Communication requests</li> </ul>
	<b>Provide</b>
	<ul style="list-style-type: none"> <li>Access to resources through OMA-DM Management Objects (<a href="http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html">http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html</a>)</li> </ul>
	– Management access through LwM2M)
	<ul style="list-style-type: none"> <li>Push Service (for notifying in-vehicle Applications form the cloud)</li> </ul>
	<ul style="list-style-type: none"> <li>Monitoring Service</li> </ul>
	<ul style="list-style-type: none"> <li>Update Service (Maintanance)</li> </ul>
	<ul style="list-style-type: none"> <li>Keep-Alive Signal Service</li> </ul>
	<ul style="list-style-type: none"> <li>Inventory Hardware/Software</li> </ul>
	<ul style="list-style-type: none"> <li>Enrollment Service</li> </ul>
	<ul style="list-style-type: none"> <li>Control Service (e.g. shutdown certain components)</li> </ul>
3.	<ul style="list-style-type: none"> <li>Apps, OS: (Re-)Configure, Update</li> </ul>
	<ul style="list-style-type: none"> <li>ECU: Flashing/Updating ECUs</li> </ul>
	<ul style="list-style-type: none"> <li>Apps: Notify/Wake Up through push</li> </ul>
	<ul style="list-style-type: none"> <li>HW/SW: Read Inventory</li> </ul>



ID	Property
	<ul style="list-style-type: none"> <li>• HW/SW: access on resource (if Management Object is defined)</li> </ul>
	<ul style="list-style-type: none"> <li>• User: Update scheduling</li> </ul>
4.	The DM can only exist once per vehicle. But it is split into its functional groups (subjects). A hierarchical design of distributed DMs in a vehicle might be the subject of another design.
5.	<ul style="list-style-type: none"> <li>• Authentication Set (MAC, UUID, Public Key)</li> </ul>
	<ul style="list-style-type: none"> <li>• Cloud contact</li> </ul>
	<ul style="list-style-type: none"> <li>– Address of DM at the desired cloud</li> </ul>
	<ul style="list-style-type: none"> <li>– Certificate</li> </ul>

## 4.6 Operating System (OS) concept:

ID	Property
1.	The Operating System is the backbone any platform where it operates including In-vehicle platform. We also consider drivers as being part of the Operating system.
	<i>Operating System</i>
	<ul style="list-style-type: none"> <li>• consists of kernel and user space components</li> </ul>
	<ul style="list-style-type: none"> <li>• provides security</li> </ul>
	<ul style="list-style-type: none"> <li>• I/O and networking services to applications running in user space</li> </ul>
	<ul style="list-style-type: none"> <li>• runs on bare metal or hypervisor</li> </ul>
	<ul style="list-style-type: none"> <li>• OS kernel provides: <ul style="list-style-type: none"> <li>– processes/thread management including scheduling</li> <li>– inter process communication</li> <li>– memory management</li> <li>– access to underlying HW</li> <li>– networking stacks</li> <li>– I/O stacks</li> <li>– security subsystem include: <ul style="list-style-type: none"> <li>* access control (e.g. MAC)</li> <li>* crypto and key management</li> <li>* integrity measurement</li> <li>* entropy pools and gathering entropy from various sources It is also important to highlight that the security hardening of an OS is crucial to the platform and application security</li> </ul> </li> </ul> </li> </ul>
2.	<b>Process</b>

ID	Property
	<ul style="list-style-type: none"> <li>• data to/from underlying HW</li> </ul>
	<ul style="list-style-type: none"> <li>• data to/from networking</li> </ul>
	<ul style="list-style-type: none"> <li>• data to/from file system</li> </ul>
	<ul style="list-style-type: none"> <li>• user input</li> </ul>
	<ul style="list-style-type: none"> <li>• data exchanged between kernel and user space</li> </ul>
	<ul style="list-style-type: none"> <li>• data exchanged between processes</li> </ul>
	<ul style="list-style-type: none"> <li>• configuration data such as security policies, system settings,..</li> </ul>
	<b>Provide</b>
	<ul style="list-style-type: none"> <li>• System state information (running processes, CPU utilization, etc.)</li> </ul>
	<ul style="list-style-type: none"> <li>• log data</li> </ul>
	<ul style="list-style-type: none"> <li>• debug or diagnostics data if certain debug features are enabled</li> </ul>
3.	<ul style="list-style-type: none"> <li>• <i>Access to underlying HW</i> <ul style="list-style-type: none"> <li>– peripherals</li> <li>– networking interfaces</li> <li>– any physical interface</li> <li>– HW based crypto and key management functionality</li> <li>– HW based TRNG (True Random Number Generator)</li> </ul> </li> <li>• <i>App-Runtime (transport layer)</i> <ul style="list-style-type: none"> <li>– Provide transport layer interface for Apps (inter-app communication, app to cloud backend communication)</li> <li>– Provide system level services (file access, etc.)</li> </ul> </li> <li>• <i>Device Management Client (transport layer)</i> <ul style="list-style-type: none"> <li>– Provide transport layer interface for communication with device management (cloud backend)</li> </ul> </li> <li>• <i>5g Infrastructure (data link layer)</i> <ul style="list-style-type: none"> <li>– Connection Management</li> </ul> </li> <li>• <i>Smartphone (data link layer)</i> <ul style="list-style-type: none"> <li>– Pairing</li> <li>– Connection Management</li> <li>– Data Transfer</li> </ul> </li> <li>• <i>Net-IDS</i> <ul style="list-style-type: none"> <li>– Provides Interface to allow the Net-IDS to monitor network traffic.</li> </ul> </li> </ul>

ID	Property
4.	Different subsystems on a vehicle may be running their own OSES
	<ul style="list-style-type: none"> <li>• whether on bare metal or virtualized</li> <li>• whether micro kernel based, unikernel based or rich OS such as Linux</li> </ul>
5.	<ul style="list-style-type: none"> <li>• build and runtime OS configuration <ul style="list-style-type: none"> <li>– for process, resource management, memory management, functionality, security,...</li> <li>– policies (e.g. security)</li> <li>– configuration of different processes</li> <li>– networking configuration</li> </ul> </li> <li>• User credentials</li> </ul>

## 5. The 5G-Infrastructure

The 5G infrastructure enables the communication between the connected vehicles and the cloud back-end. Based on the 5G standard description, two component layers are structured as *control plane* and *user plane* which represents communication with cloud back-end and communication with the connected vehicle respectively.

5G is the next generation of mobile communication technology. It is expected to be defined by the end of this decade and to be widely deployed in the early years of the next decade. As opposed to earlier 3G and 4G technologies, 3GPP conceptualized 5G to be more than another mobile broadband connectivity, covering a variety of use-cases and industries. 5G was initially based on the conceptual composition as well as evolution of cellular technologies. This is because the different technologies have communication requirements, that are focused in a local (LAN) or wide area network (WAN) communication.

In particular, IEEE's 802.11p has been developed to support different types of wireless communications (e.g. PAN, LAN), but since it is based on CSMA/CA its performance degrades quickly as network load increases. This happens because a high number of transmitting stations will increase the number of collisions on the communication medium. Additionally, since it was designed for short-range transmissions (transmission range up to 1km), many vendors introduce a multi-hop functionality to increase the transmission range. An example of multi-hop functionality is introduced in the European Cooperative Intelligent Transport System (C-ITS) protocol stack named as GeoNetworking. Furthermore, another technology that could not support as standalone broadband (e.g. WAN) communications is LTE. The main drawback of this technology is that every transmitted packet must traverse the infrastructure, meaning that each infrastructure failure will have a strong impact on connectivity of the entire network. Furthermore, even though broadband connections can be supported in LTE, scenarios where the infrastructure is not available due to out-of-coverage are also quite probable. Finally, since LTE was designed to use radio resources in order to allow broadband communication, its extension in V2X connectivity where smaller data packets and higher bandwidth are required is suboptimal in terms of consumed resources still remains a great challenge.

### 5.1 Evolved Packet Core (EPC) concept:

ID	Property
----	----------

ID	Property
1.	EPC includes a number of components such as MME (Mobility Management Entity), SGW (Serving Gateway), PGW (Packet Data Network Gateway), HSS (HomeSubscriber Server). Each of these components have their own purpose
	<ul style="list-style-type: none"> <li>• MME (Mobility Management Entity): located in the control plane and handles authentication, roaming and other management functions</li> </ul>
	<ul style="list-style-type: none"> <li>• SGW (Serving Gateway): handles routing user data packets</li> </ul>
	<ul style="list-style-type: none"> <li>• PGW (Packet Data Network Gateway): handles data connectivity with external networks. This is also where packet processing and lawful interception can take place</li> </ul>
	<ul style="list-style-type: none"> <li>• HSS (Home Subscriber Server): handles user subscription information</li> </ul>
2.	<ul style="list-style-type: none"> <li>• MME (Mobility Management Entity): authentication data, roaming data</li> </ul>
	<ul style="list-style-type: none"> <li>• SGW (Serving Gateway): processes user data packets</li> </ul>
	<ul style="list-style-type: none"> <li>• PGW (Packet Data Network Gateway): policy enforcement, packet processing and filtering.</li> </ul>
	<ul style="list-style-type: none"> <li>• HSS (Home Subscriber Server): processes and stores user ubscription data</li> </ul>
3.	<ul style="list-style-type: none"> <li>• MME interacts with eNodeB, SGW and HSS</li> </ul>
	<ul style="list-style-type: none"> <li>• SGW interacts with MME and PGW</li> </ul>
	<ul style="list-style-type: none"> <li>• PGW interacts with SGW and external network</li> </ul>
	<ul style="list-style-type: none"> <li>• HSS interacts with MME</li> </ul>
4.	There can be multiple instances
5.	Depends on the vendors and the products

### 5.1.1 eNodeB (Baseband Unit) (BBU) concept:

ID	Property
1.	BBU (Baseband Unit) is responsible for processing baseband signals received from RRUs and further communicating data with EPC.
2.	Processes baseband signals received from RRUs (Remote Radio Units) and other data from EPC
3.	Interacts with RRUs and EPC
4.	There can be multiple instances
5.	Depends on the vendors and the products

### 5.1.2 eNodeB (Remote Radio Units) (RRU) concept:

ID	Property
1.	Interacts with user equipment by using a signaling protocol and relays baseband signals to BBUs for further processing

ID	Property
2.	Signaling data
3.	Interacts with user equipment and BBUs
4.	Multiple instances
5.	Depends on the vendors and the products

## 6. Cloud back-end

The cloud back-end provides service components to the connected vehicle by making sure a reliable and safe functionality according to the pre-defined operation. Just like the In-vehicle platform, the cloud back-end is composed of multiple layers: **Core Layer**: the core layer functions as

- central back-end applications
  - messaging infrastructure
  - control the message flow from and to the connected vehicles, and
  - device update management and data management. **Data Analytic & Visualization Layer**: this layer analyzes and visualizes the data transmitted by the vehicle based on the functionalities of the core layer.
- Application Layer**: In general, applications that are deployed on in-vehicle platforms require a cloud back-end counterpart. Often, applications in the back-end are provided by third parties allowing vehicle owners to acquire certain functionalities. The market place is also accessible for developers and service providers to register and upload applications. The original equipment manufacturers (OEMs) has the access and controls on contents of the market place.

### 6.1 Message Gateway concept:

ID	Property
1.	<ul style="list-style-type: none"> <li>• Serves as central messaging hub</li> <li>• Provides a set of micro services that transport messages from the car to the other components within the APPSTACLE backend and vice versa</li> <li>• Implements routing and brokering mechanisms</li> </ul>
2.	<b>Input and Output</b>
	<ul style="list-style-type: none"> <li>• Telemetry data</li> <li>• Event messages</li> <li>• Command &amp; control messages</li> <li>• Software updates</li> <li>• Each message type may have a different quality of service level, depending on the configuration of the system</li> </ul>
	<b>Messaging Protocols</b>
	<ul style="list-style-type: none"> <li>• <i>MQTT</i></li> </ul>

ID	Property
	– Publish/subscribe model
	– Implements own security model
	• <i>HTTP (REST-based)</i>
	– GET, SET, PUT, DELETE
	• <i>Other protocols</i>
3.	• Receiving and routing messages to the correct recipient
	• Providing or leveraging identity management for connected devices
4.	• There exists only a single instance within the cloud backend
5.	• Deployment configuration regarding micro services:
	– External IP addresses
	– Ports
	– Connection to other services, e.g, identity management, service bus, etc.

## 6.2 Device Management Backend concept:

ID	Property
1.	• Managing the software stack on the individual devices
	• Responsible for distributing the software artifacts to the devices
	• Steering software rollout campaigns for large sets of devices
2.	• It provides arbitrary software and data artifacts that are stored in one or multiple repositories
3.	<b>Input</b>
	• It interacts either with an app store or a backend service that is provided by the OEM
	• It receives commands that initiate the change of software stack of one or multiple devices
	<b>Output</b>
	• It sends software and data artifacts to one or multiple devices
	• The outbound communication is ideally conducted via the messaging gateway, managing the packaging according to the according protocol
	– HTTP
	– LWM2M
	– OMA-DM
	– Other
	• The outbound communication may also be realised using direct communication to the devices

ID	Property
4.	<ul style="list-style-type: none"> <li>• There exists only a single service within the cloud backend</li> </ul>
	<ul style="list-style-type: none"> <li>• The may exist multiple instances behind the service interface</li> </ul>
5.	<ul style="list-style-type: none"> <li>• The configuration for initialization is provided within deployment scripts</li> </ul>
	<ul style="list-style-type: none"> <li>• The device management resources are expected to be deployed into an existing cluster</li> </ul>

## 6.3 Report Generation concept:

ID	Property
1.	Report Generation is a component to generate business reports from the data collected by the cloud platform.
2.	data collected by the cloud platform and, in particular, results from data analytics; output is a "report" (document)
3.	The Report Generation component takes data from the Data Management (e.g., results of the data analytics component) and creates reports, including visualizations, lists, documents, charts, etc.
	It is implemented based on Eclipse BIRT.
	Q: How is report generation triggered - periodically? by Apps? what interfaces exist?
	Q: Who can request request reports? End-users (e.g., vehicle owner)? Service providers? OEMs?
	Remark: It must be ensured that report generation cannot be misused by malicious users to violate data protection rules or to obtain confidential information without authorization.
4.	Usually there is one logical report generation component per cloud instance, but potentially, there could be different report generation components for different kinds of reports.
5.	<ul style="list-style-type: none"> <li>• The configuration for initialization is provided within deployment scripts</li> </ul>
	<ul style="list-style-type: none"> <li>• The device management resources are expected to be deployed into an existing cluster</li> </ul>

## 6.4 Marketplace Backend concept:

ID	Property
1.	<ul style="list-style-type: none"> <li>• Stores and manages the software and data artifacts related to the apps that are installed on the in-vehicle platform</li> </ul>
	<ul style="list-style-type: none"> <li>• Initiates and controls the communication with the app provider backends</li> </ul>
	<ul style="list-style-type: none"> <li>• Interacts with potential payment providers, depending on the payment methods applied (Credit/Debit Card, SMS)</li> </ul>
	<ul style="list-style-type: none"> <li>• Stores the data related to registered users (vehicle owners and app developers)</li> </ul>
	<ul style="list-style-type: none"> <li>• Central hub for the interaction with app developers</li> </ul>
	<ul style="list-style-type: none"> <li>• Provides app data to the in-vehicle platforms via the device management back-end</li> </ul>
	<ul style="list-style-type: none"> <li>• Scan app software and data artifacts regarding vulnerabilities and malware</li> </ul>

ID	Property
	<ul style="list-style-type: none"> <li>• Question: Should in-app transactions be possible?</li> </ul>
2.	<b>Process</b>
	<ul style="list-style-type: none"> <li>• User input via the Marketplace Frontend</li> </ul>
	<ul style="list-style-type: none"> <li>– Vehicle owner core data (Name, Address, Supported payment methods, User vehicle mappings (1 User, N Vehicles))</li> </ul>
	<ul style="list-style-type: none"> <li>– App developer core data (Name, Company, Address, Supported payment methods, Apps provided)</li> </ul>
	<ul style="list-style-type: none"> <li>– Transactional data regarding buying and returning apps (Transaction ID, Transaction status, Payment method applied, Vehicle Identification Number, Version of the app transferred)</li> </ul>
	<b>Provide</b>
	<ul style="list-style-type: none"> <li>• Software and data artifacts that are provided by the app developers</li> </ul>
	<ul style="list-style-type: none"> <li>– Software (Compiled source code)</li> </ul>
	<ul style="list-style-type: none"> <li>– Data (Licenses, Documentation, Version history, Supported in-vehicle software platforms, Supported in-vehicle hardware platforms)</li> </ul>
	<ul style="list-style-type: none"> <li>– API/Frontend for app developer interaction (Transferring software and data artifacts to the backend, Receiving account and app related information, e.g. number of downloads or total revenue per app)</li> </ul>
3.	<b>Input</b>
	<ul style="list-style-type: none"> <li>• Marketplace frontend (see informations listed above)</li> </ul>
	<ul style="list-style-type: none"> <li>• Payment provider backends (Handling of payment transactions)</li> </ul>
	<ul style="list-style-type: none"> <li>• App developer backends (Receiving data and software artifacts) Output</li> </ul>
	<ul style="list-style-type: none"> <li>• Marketplace frontend (Deliver data requested by the frontend, e.g., results for search queries)</li> </ul>
	<ul style="list-style-type: none"> <li>• In-vehicle platforms, via Device Management component (Software and data artifacts regarding specific apps)</li> </ul>
	<ul style="list-style-type: none"> <li>• App developer backends (Transaction and evaluation data)</li> </ul>
	<ul style="list-style-type: none"> <li>• Identity Management (Requests for user authentication and authorisation)</li> </ul>
4.	<ul style="list-style-type: none"> <li>• There should be only a single instance within the cloud backend</li> </ul>
5.	<ul style="list-style-type: none"> <li>• Deployment configuration regarding micro services</li> </ul>
	<ul style="list-style-type: none"> <li>– External IP addresses</li> </ul>
	<ul style="list-style-type: none"> <li>– Ports</li> </ul>
	<ul style="list-style-type: none"> <li>– Connection to other services, e.g, Identity Management</li> </ul>

## 6.5 Visualization concept:

ID	Property
----	----------



ID	Property
1.	The Visualization component creates graphical representations of time series data retrieved via the Data Management component. For example, the Visualization component can generate graph like representations of the evolution of telemetry data over time.
2.	<b>Process</b>
	<ul style="list-style-type: none"> <li>• Time series data (e.g., telemetry data, performance/monitoring data)</li> </ul>
	<b>Provide</b>
	<ul style="list-style-type: none"> <li>• Graphical representations that can be utilized for report generation</li> </ul>
3.	<ul style="list-style-type: none"> <li>• Data mangement component</li> </ul>
	<ul style="list-style-type: none"> <li>– retrieving time series data</li> </ul>
	<ul style="list-style-type: none"> <li>– sending representation for storage and use in report generation</li> </ul>
4.	There should be at least one Visualization instance per Data Management instance because of scalability aspects. Multiple instances per data management instance are possible. <b>Not sure here</b>
5.	<ul style="list-style-type: none"> <li>• Data source</li> </ul>
	<ul style="list-style-type: none"> <li>• Dashboard information</li> </ul>
	<ul style="list-style-type: none"> <li>– Metrics</li> </ul>
	<ul style="list-style-type: none"> <li>– Representation style</li> </ul>
	<ul style="list-style-type: none"> <li>– Layout information</li> </ul>

## 6.6 Marketplace Frontend concept:

ID	Property
1.	<ul style="list-style-type: none"> <li>• Provides visual interface that enable the interaction with the Marketplace Backend</li> </ul>
	<ul style="list-style-type: none"> <li>• Vehicle owners can search and order apps as well as initiate their transfer to the in-vehicle platform</li> </ul>
	<ul style="list-style-type: none"> <li>• Allows app developers to access app storage and monitoring data</li> </ul>
	<ul style="list-style-type: none"> <li>• Allows vechicle owners and app developers to edit their core data / profiles</li> </ul>
2.	<b>Process</b>
	<ul style="list-style-type: none"> <li>• Core data (vehicle owner and app developer) entered via the visual interface</li> </ul>
	<ul style="list-style-type: none"> <li>• Query database regarding apps available for the specific in-vehicle platform</li> </ul>
	<ul style="list-style-type: none"> <li>• Software and data artifacts send by the app developers</li> </ul>
	<ul style="list-style-type: none"> <li>• Data referring to payment transaction, e.g., data exchange with payment providers</li> </ul>
	<b>Provide</b>
	<ul style="list-style-type: none"> <li>• Visual interface depicts information regarding</li> </ul>
	<ul style="list-style-type: none"> <li>– Vehicle owner and app developer core data</li> </ul>

ID	Property
	– App transactions
	– App search query results
3.	<b>Input</b>
	• Vehicle owner
	– Enter and update core data
	– Fill out order forms
	– Search apps via query masks
	• App developer
	– Enter and update core data
	– Upload apps
	– Retrieve monitoring data
	<b>Output</b>
	• Vehicle owner
	– Display core data
	– Display app search queries
	• App developer
	– Display core data
	– Display monitoring data
4.	• There exists only a single services within the backend
	• There might be several container instances behind this service
5.	• Deployment configuration regarding micro services
	– External IP addresses
	– Ports
	– Connection to other services, e.g, Marketplace Backend

## 6.7 Data Management concept:

ID	Property
1.	• Manages the data that is transferred from the in-vehicle platforms to the cloud backend
	<b>Historical Data</b>
	• Provides storage mechanisms to persist historical sensor data
	• Is independent of the storage technologies applied

ID	Property
	– Time series vs. transactional vs. analytical data bases
	– Relational vs. object data bases
	<b>Streaming Data</b>
	• It assumed that only portions of the data transferred can actually be stored, due to the large volume
	• Instead of storing the full set of data, it may be analysed on-the-fly
	• Therefore data must be rerouted to corresponding systems
2.	• Sensor data that is collected by the in-vehicle platform and transmitted by the message gateway
3.	<b>Input</b>
	• Message Gateway
	– Sends sensor data to the Data Management component
	– Data Management component acts as a consumer
	<b>Output</b>
	• Data bases
	– Receive subset of data that is transferred via the Message Gateway
	• Stream processing engines
	– Receive full set of data that is transferred via the Message Gateway
4.	• There exists of a single vertically scaled instance within the cloud backend
	• It may be connected to multiple data bases and stream processing engines
5.	• Deployment configuration regarding micro services
	– External IP addresses
	– Ports
	– Connection to other services, e.g, Message Gateway and Big Data Analysis

## 6.8 Identity Management concept:

ID	Property
1.	• Managing the identities of the entities within the connected car environment, such as:
	– Vehicles
	– Backend services within the following layers:
	* Core Layer
	* Data Analytic & Visualization Layer
	* Application Layer

ID	Property
	– Marketplace users
	• Authorization of vehicle access by marketplace users
2.	• Login data of the marketplace users
	• Authentication data of backend services as well as vehicles
	• Authorization data of vehicles, backend services and marketplace users
3.	• Exchange of login data, authorization data and authentication data
4.	• There exists only a single instance within the cloud environment.
5.	• The setup of the identity management component requires the awareness regarding the individual backend services.
	• Regarding the authentication and authorization of vehicles and marketplace users, specific mechanisms have to be selected and configured.

## 6.9 Device Representation concept:

ID	Property
1.	• Providing an abstraction between the physical devices, the vehicles, and their digital representations (digital twin)
	• Managing the state of the vehicles attached to the backend infrastructure (reported vs. desired vs. current)
	• Providing a unified interface for accessing the information provided by the vehicles
	• Organizing the digital twins for all vehicles attached to the backend infrastructure
2.	• Information provided by the physical devices, in particular sensor data
3.	• Synchronizing data between the physical devices and their digital representations
	• Processing data provided by the physical devices
4.	• There exists only a single instance managing all digital twins referring to all cars within the connected car environment
5.	• The Device Representation component has to be connected to the message gateway
	• A digital twin is created for each connected vehicle added to the environment

## 6.10 Big Data Analysis concept:

ID	Property
1.	• Vehicles provide large amounts of data that are is by the Data Management component.
	• Big data analysis provides insights, e.g., the properties and usage of individual vehicles as well as their interaction.

ID	Property
2.	<ul style="list-style-type: none"> <li>• It consumes data collected by the vehicles that is persisted by the Data Management component.</li> </ul>
	<ul style="list-style-type: none"> <li>• This data serves as input for the analytical tasks, depending on the individual objectives.</li> </ul>
	<ul style="list-style-type: none"> <li>• The analytical results are provided to Domain-specific services.</li> </ul>
3.	<ul style="list-style-type: none"> <li>• Data is retrieved from the Data Management component.</li> </ul>
	<ul style="list-style-type: none"> <li>• Analytical tasks are initiated by Domain-specific Services.</li> </ul>
4.	<ul style="list-style-type: none"> <li>• There may be multiple Big Data Analysis components within the environment.</li> </ul>
5.	<ul style="list-style-type: none"> <li>• The Big Data Analysis component(s) have to be connected to the Data Management component as well as the Domain-specific Services.</li> </ul>
	<ul style="list-style-type: none"> <li>• This includes deployment configuration data, such as: <ul style="list-style-type: none"> <li>– External IP addresses</li> <li>– Ports</li> </ul> </li> </ul>

## 6.11 Core Services concept:

ID	Property
1.	<ul style="list-style-type: none"> <li>• Provide an abstraction layer regarding the access of the connected vehicles within in the environment.</li> </ul>
	<ul style="list-style-type: none"> <li>• The focus is especially on the sending messages to and receiving messages from the connected vehicles.</li> </ul>
	<ul style="list-style-type: none"> <li>• Therefore, it relies on the functionality provided by the Message Gateway component.</li> </ul>
	<ul style="list-style-type: none"> <li>• Potentially restrict the access of Domain-specific Services.</li> </ul>
2.	<ul style="list-style-type: none"> <li>• It provides access to the information provided by the connected vehicles to the Domain-specific Services.</li> </ul>
	<ul style="list-style-type: none"> <li>• It allows to send messages from Domain-specific Services to the connected vehicles.</li> </ul>
3.	<ul style="list-style-type: none"> <li>• Consume data from the Message Gateway component and route it to the Domain-specific Services.</li> </ul>
	<ul style="list-style-type: none"> <li>• Consume messages from the Domain-specific Services and route them to the Message Gateway component.</li> </ul>
4.	<ul style="list-style-type: none"> <li>• There exist different services but for each service a single instance.</li> </ul>
5.	<ul style="list-style-type: none"> <li>• The Core Services have to be connected to the Message Gateway component as well as the Domain-specific services.</li> </ul>
	<ul style="list-style-type: none"> <li>• This includes deployment configuration data, such as: <ul style="list-style-type: none"> <li>– External IP addresses</li> <li>– Ports</li> </ul> </li> </ul>

## 6.12 Domain-specific Services concept:

ID	Property
1.	<ul style="list-style-type: none"> <li>• The Domain-specific Services provide added value to the users of the APPSTACLE connected car environment.</li> </ul>
2.	<ul style="list-style-type: none"> <li>• The provide selected functionalities that leverage the data as well as the communication provided by the connected vehicles.</li> <li>• It consumes data that is provided the Core Services component.</li> <li>• It accesses and synchronizes the state of the connected vehicles via the Device Representation component.</li> <li>• It consumes aggregated data that is provided by the Big Data Analysis component.</li> </ul>
3.	<ul style="list-style-type: none"> <li>• Accessing and consuming data that is provided by the components attached.</li> <li>• Sending messages to the connected vehicles.</li> </ul>
4.	<ul style="list-style-type: none"> <li>• Domain-specific Services may be provided by third-party developers.</li> <li>• Therefore, there may exist a set of individual services which do not necessarily depend on each other.</li> </ul>
5.	<ul style="list-style-type: none"> <li>• Depending on the use case, the individual services have to be connected to the Device Representation component, the Big Data Analysis component as well as the Core Services component.</li> <li>• This configuration data includes: <ul style="list-style-type: none"> <li>– External IP addresses</li> <li>– Ports</li> </ul> </li> </ul>

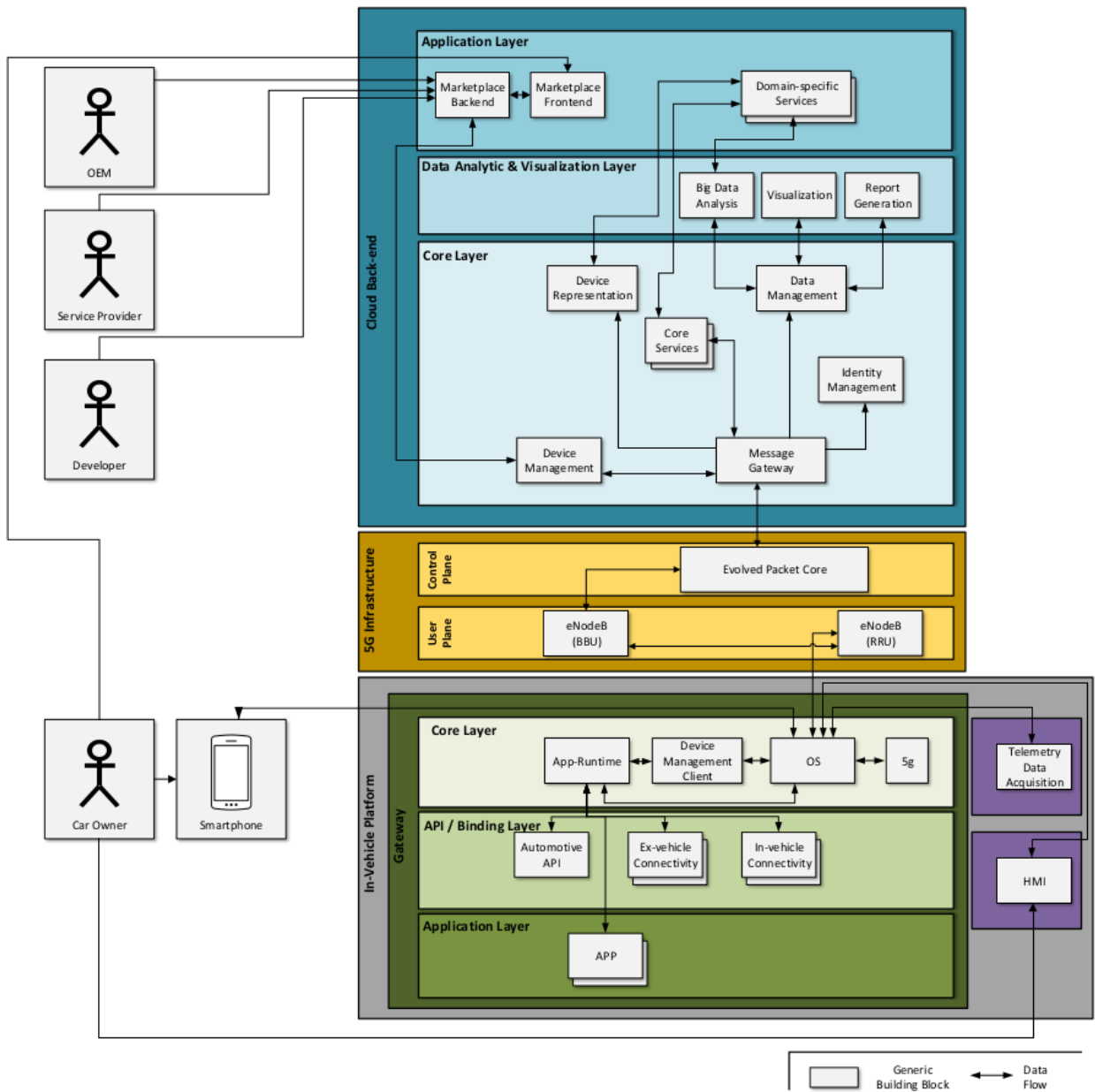


Figure 5. The above picture summarizes the the architectural concepts discussed so far.

## 7. Kuksa-IDE Building & Deploying

---

A documentation repository containing implementation to setup Eclipse Kuksa che instance is available in [here](#). Which also contains the Automotive Grade Linux (AGL) stack with Yocto support. AGL represents an automotive specific Linux distribution specifically designed as open software stack for connected car scenarios. An example on how to use the Kuksa-IDE for developing AGL applications and services running on a Rover can be found in [here](#).

### 7.1 Version

*Table 6.1 Eclipse Che Kuksa instance version*

Available	Version number
Current	6.10
Plan	7.0

The Kuksa-IDE Developers guidance and the neccessary steps for building and running Eclipse Che kuksa instance are explained in this [link](#).

For the neccessary [prerequisites](#) and [Eclipse Che Kuksa setup](#), please visit [here](#).

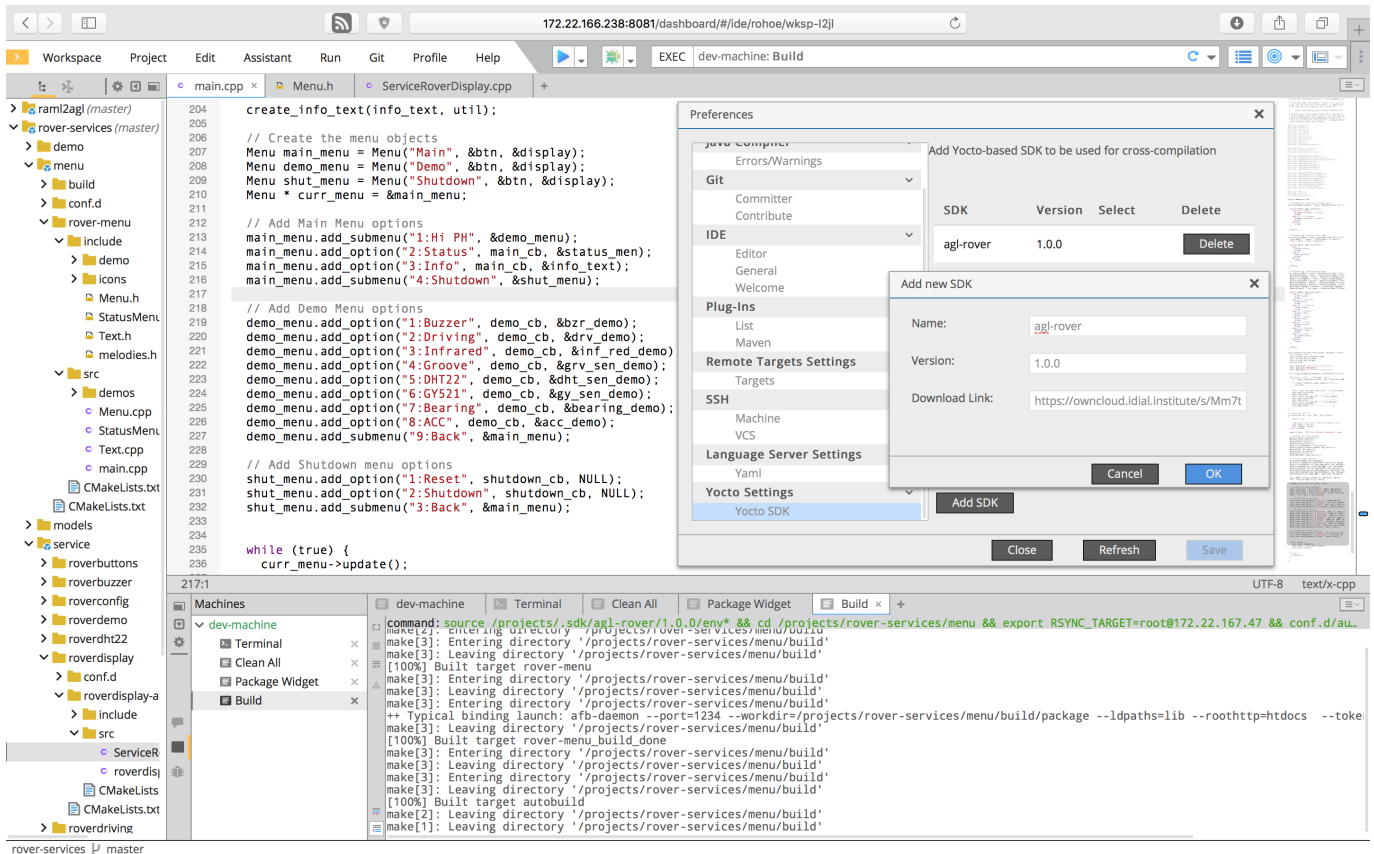
### 7.2 IDE

The IDE provided under Eclipse Kuksa not only support the development of applications for the vehicle component, but also the creation of applications for the cloud. Users should be able to choose between two different workspaces and technology stacks that contain the preconfigured and embedded APIs as well as software libraries of the respective applications to be developed. This allows the car to be equipped with new functions and new services to be deployed in the cloud.

Kuksa offers various APIs for implementing vehicle applications, a project template for cloud services, and wizards for easily providing vehicle applications in the App Store via the IDE. The extensive provision of the various APIs and libraries in the IDE enables accessing existing communication interfaces for the secure data transmission, storage, management, and authentication without having to take separate measurements for processing or interpreting the data.

Kuksa also supports the simplified deployment of new applications for both the cloud and vehicle components. This is provided by a pre-configured Eclipse Che stack, to which only the address of a target platform must be specified. Configuration, building and deployment can be done at the push of a button without further configuration or processing. Depending on the application, different development tools (e.g. Logging, Debugging, Tracing,...) can be included. Of course, syntax highlighting, code completion, and other necessary IDE functions are supported. For instance, the in-vehicle Eclipse Kuksa Che stack for AGL development activities features including Yocto based SDKs in order to support target specific programming shown in the screenshot below. After compiling and building software, specifying a target IP allows also the deployment process.





In order to make new applications applicable to a greater amount of vehicles, applications need to be centrally checked, managed, and organized with regard to various in-vehicle derivatives and variants in such a way that only vehicle-appropriate applications are accessible. Similar to a Smartphone App Store, it has to be possible to add new functions and applications to their vehicle or perform updates or upgrades. Therefore, standardized interfaces of the in-vehicle and cloud platforms are required and they must offer the most diverse and yet simple infrastructure for vehicle owners. Authentication methods, security concepts, variant management, and suitable data transmission technologies in combination with the publicly accessible ecosystem form mandatory components as well as the difference to existing solutions. [Click here to read more.](#)

### 7.2.1 Here are repository folders for [kuksa.apps](#)

The prerequisite to [set up the Kuksa IDE](#) are:

- A running docker instance on a Linux host system (tested with Ubuntu 18.04).
- Maven

[Kuksa IDE -Developers Guide](#) consists of build the assembly for system configuration and how to start kuksa IDE. In addition, a [sample project](#) provides custome sample project that can be used as kuksa App example.

### 7.2.2 Getting started with [Kuksa appstore](#).

## 7.3 Getting started with the In-Vehicle platform

[Kuksa IDE - Developers Guide](#) contains the neccessary guides such as:- *Quick start, Kuksa IDE Custom Assembly, Plugins, Stacks and Sample Projects.*

### 7.3.1 Kuksa App example

The eclipse Che doesn't provide a standard mechanism to add custom sample projects during build time. Therefore, Kuksa IDE provides an easy and straight forward mechanism to append them to ones provided by Eclipse Che during build time. [Sample Projects](#) can be found from this link.

## 8. Cloud back-end

The cloud back-end is the counterpart to the services provided by the in-vehicle platform. It offers basic services regarding connectivity, authentication, authorization, device update and data management. These services are realized by open source [Eclipse IoT](#) technologies, which are tailored to the requirements of a connected vehicle platform. An abstract overview of the cloud components is given in the Figure below.

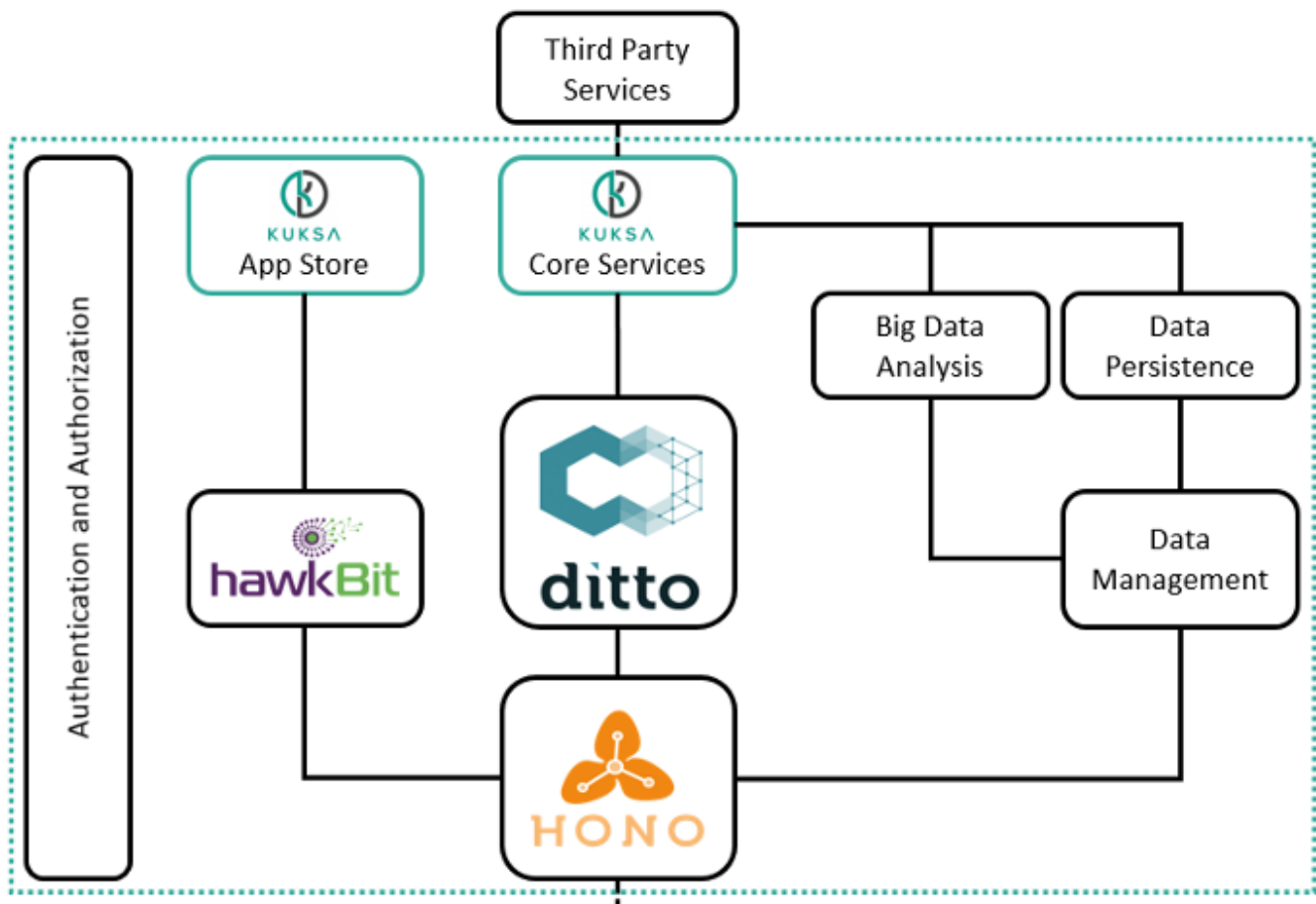


Figure 8. Cloud component overview.

Central functionalities rely on a unified and secure communication between the back-end and the in-vehicle platform. In this regard, the connectivity for a large number of vehicles is realized using [Eclipse Hono](#), which receives telemetry as well as event data and allows to transmit information to the vehicles. Open communication protocols such as AMQP and MQTT are accompanied by vehicle identity management.

The large amount of collected data can either be processed by big data analysis applications via streaming or persisted using database management systems. In addition, the state on the individual vehicles is managed using [Eclipse Ditto](#), a digital twin implementation, which allows the synchronization of their physical and virtual representations.

A device management component takes care of provisioning software updates to the vehicle, both in terms of the core system as well as adding or modifying vehicle functionality through apps. The former comprises so-called rollout management, which controls the distribution of the software to a large number of vehicles. This component is represented by [Eclipse hawkBit](#). Applications augmenting the functionality of the vehicles are provisioned by the vehicle manufacturers or external developers via an app store.

A set of core services provide the access to the vehicles managed within the cloud back-end. In this regard, an authentication and authorization component ensures that malicious access by third-party services is prohibited. This involves all major components of the back-end infrastructure. To read more [click here](#)

## 8.1 Kuksa Cloud Deployment

The scripts in this directory and its subdirectories help to setup a deployment of the Kuksa cloud. These scripts assume a running Kubernetes cluster which can be configured using *kubectf*. More information regarding the parameters of the scripts can be found within the respective script file [here](#).

### Structure

The deployment scripts are divided into the following parts:

1. Eclipse hawkBit enables the deployment of the corresponding software update components, in particular the update server. Note that this step requires the installation of the command line tool kompose. [Here](#) is the installation instructions.
2. Eclipse Hono enables the deployment of a messaging infrastructure.
3. Kubernetes provides functions for the Kubernetes deployment of the Kuksa cloud.
4. Utils scripts that are included by other parts of the deployment infrastructure (e.g. handling static IP-addresses for the services). It is possible to set static IP-addresses and DNS entries for deployed services. For more details on that configuration see the [Readme.md](#) file in the [utils](#) directory.

## 8.2 Getting started with Kuksa Appstore

The necessary infrastructures, prerequisites, key features and deployment setups are discussed in this [repository](#).

## 9. Kuksa In-Vehicle

[Eclipse Kuksa](#) includes an open and secure cloud platform that interconnects a wide range of vehicles to the cloud via open in-car and Internet connection and is supported by an integrated open source software development ecosystem. The Eclipse Kuksa project contains a set of repositories and this repo is one among those that contains in-vehicle platform code and also contains required layers and bindings to build a Kuksa adapted AGL (Automotive Grade Linux) distribution. The in-vehicle platform is primarily designed to work with AGL. However the individual components found in [this repo](#) could be used on other platforms as well.

Kuksa is a wrapper project around Automotive Grade Linux (AGL). From its side, AGL uses Yocto/Bitbake building system to build an automotive domain specific Linux distribution. Therefore, this project provides a building system that adds Kuksa's specific Bitbake layers on top of the original AGL. The scripts in this project help ease the process of building an AGL image by simply using a few commands. This project includes the yocto recipes found in meta-kuksa project.

Therefore, to get started with In-Vehicle platform, AGL KUKSA Build and Run on Raspberry Pi 3 / Compute Module 3 (Lite) can be found from the link [here](#).

In order to Build the Image/SDK with cmake scripts, the required system configuration both (hardware and software) are:

```
Need Ubuntu 16

Fast Internet connection.

Minimum of 100 GB memory.

Some patience as it takes about 8 hours the first time.
```

To build the Image/SDK, run;

```
cd mkdir build cd build cmake .. make
```

Where can be;

```
agl-kuksa-sdk: AGL kuksa image and SDK
agl-kuksa: AGL kuksa Image only
Other Targets to follow.
```

The output images can be seen at /build/images and the SDKs at /build/sdk.

To set up and build the Image using yocto/bitbake, the neccessary prerequisites are:

```
Need Ubuntu 16

Fast Internet connection.

Minimum of 100 GB memory.

Some patience as it takes about 8 hours the first time.
```

## Steps

### *Setup the machine*

Execute

```
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat libstdl1.2-dev xterm cpio curl
```

This will install the necessary packages.

Execute

```
export AGL_TOP=$HOME/workspace_agl; mkdir -p $AGL_TOP
```

Execute

```
mkdir -p ~/bin ; export PATH=~/bin:$PATH ; curl https://storage.googleapis.com/git-repo-downloads/repo >
~/bin/repo; chmod a+x ~/bin/repo
```

This will set up the repo tool. Repo tool is used to download the recipes for AGL image.

Execute

```
cd $AGL_TOP ; repo init -b flounder -m flounder_6.0.1.xml -u https://gerrit.automotivelinux.org/gerrit/AGL/AGL-
repo ; repo sync
```

This will download the Funky Flounder version of AGL. This version has been tested and is recommended.

## Start Building

Execute

```
source meta-agl/scripts/aglsetup.sh -m raspberrypi3 agl-demo agl-netboot agl-appfw-smack ; bitbake agl-demo-
platform
```

This will start the build system and would take about 7 hours to complete if you are running for the first time, so you could take a nap 😴. The Yocto/bitbake build system has a caching mechanism and hence from the next time on, this would only take a few minutes.

## Adding Kuksa layers

Go to \$HOME/workspace\_agl/build/conf folder and open bblayers.conf file.

Append the following lines to the end of the file.

```
BBLAYERS += "
${METADIR}/meta-kuksa
${METADIR}/meta-kuksa/meta-kuksa-bsp
${METADIR}/meta-virtualization
"
```

Now copy the meta-kuksa folder (Link : <https://github.com/eclipse/kuksa.invehicle/agl-kuksa>) into the \$HOME/workspace\_agl directory.

## Building for the Raspberry Pi Compute Module 3 (Lite)

To build for the Raspberry Pi CM3 (Lite) platform, go to \$HOME/workspace\_agl/build/conf folder and open local.conf file.

Append the following lines to the end of the file.

```
KERNEL_IMAGETYPE = "zImage"
```

## Configure meta-kuksa layer

The kuksa layer contains recipes for the APIs and Apps contained in Eclipse kuksa Invehicle repo.

The AGL image with meta-kuksa layer adds w3c-visserver-api and elm327-visdatafeeder as systemd services. It will install the datalogger apps in the respective locations /usr/bin/datalogger-

## Set up wifi

--- Ignore this step if wifi is not required ---

With meta-kuksa layer the wifi connection could be set up while building an Image so that the target device connects to the specified wifi, which make it easier to ssh into the device. The wifi settings could be configured by modifying the meta-kuksa/recipes-devtools/wifi-conf/files/wifi\_default.config file. Update the "Name" and the "Passphrase" of the wifi you want the device to connect to. More more secured wifi connection please refer to the link

## configure Bluetooth connection with ELM 327 bluetooth adapter

The elm327-datafeeder service connects to an ELM327 Bluetooth adapter to retrieve data from the vehicle. Hence the bluetooth connection with the ELM327 adapter needs to be established before the service starts. The BT connection can be configured by Updating the MAC-Address of the adapter along with its pairing PIN. The MAC-Addr and PIN can be updated in file meta-kuksa/recipes-elm327-visdatafeeder/elm327-visdatafeeder/files/bt\_setup.sh

```
#!/bin/bash

# Enter the MAC-ADDR of your bluetooth elm327 adapter here.
MACID='00:11:03:01:04:35'
# Enter the pair key of your elm327 adapter here.
PAIRID='6789'
```

Update the fields

Alt text

Now Execute the below line to build image with Kuksa layers

```
source meta-agl/scripts/aglsetup.sh -m raspberrypi3 agl-demo agl-netboot agl-appfw-smack ; bitbake agl-demo-platform
```

This would take a few minutes to execute and at the end of the process the bootable image for RaspberryPi 3 will be found in the below location

```
$HOME/workspace_agl/build/tmp/deploy/images/raspberrypi3
```