# Rethinking Automotive Software Development: Exploring Software Defined Vehicle and its potential

**Candidate:** Lorenzo Sciara
**Supervisors:** prof. Danilo Bazzanella
dott.sa Piera Limonet

## 1 Introduction and Motivation

The *Software Defined Vehicle* (SDV) paradigm is an innovative technology in the automotive industry. It enables software to become a fundamental element of vehicle design by connecting the vehicle to cloud services and separating software development from supporting hardware systems. The resulting benefits of the innovation are the increased safety of automobiles that can be remotely updated throughout their entire lifecycle, and the increased efficiency in the production of automotive software, reducing the waste of economic resources and development time. Additionally, this technology enables the activation or deactivation of vehicle features after the initial purchase, increasing the value of the vehicles and the company's revenue.

### 1.1 Context

In the automotive industry, until recently, software produced for vehicles was always considered secondary to the production of the vehicles themselves. The focus on the mechanical parts of the vehicle resulted in software being very dependent on the hardware for which it was implemented.The software production cycle for a new vehicle consisted of three phases: development, testing, and distribution on the final hardware system. If the system responded correctly, the software remained more or less unchanged for the entire life of the car during the distribution phase. Any modifications had to be made physically, resulting in significant resource waste. If the software did not respond correctly, the development and testing process had to be restarted on new hardware, resulting in additional costs and production delays.

The introduction of the SDV completely changes the paradigm. The vehicle's systems are no longer coordinated by special-purpose devices related to the system itself, but rather by easily reprogrammable general-purpose processors.Additionally, the vehicle is directly connected to the cloud for data analysis and has the potential to receive *Over The Air* (OTA) updates for any of its systems. At this point, the vehicle becomes a controllable device that can be easily updated based on the data and information it generates, similar to more common devices like smartphones or laptops. This increases both the production efficiency of the software and the safety of the vehicle, since any system vulnerability can be promptly identified, corrected and resolved through updates. Safety is a crucial aspect in the production of vehicle software, so much so that it is considered and classified by the *International Organization for Standardization* (ISO) as a safety-critical device for human life.

In the last few years, several companies have come together to collaborate on the development of this technology. The *Scalable Open Architecture for the Embedded Edge* (SOAFEE) project was born from the cooperation of leading companies in the sector, including Amazon and Arm. Its goal is to create a software architectural standard based on Arm devices that can provide a cloud-native open-source environment for mixed-criticality automotive applications. The project is currently under development.

In view of the previous considerations, this thesis, in collaboration with Storm Reply, aims to provide an overview of the essential elements for creating the SDV and presents a practical project that illustrates its operation with the support of a Raspberry Pi board. The text begins with an overview of the state of the art technologies and approaches currently available, then moves to cloud computing and the advantages offered by *Amazon Web Services* (AWS), and follows with an exploration of the use of AWS services in the context of the project. Finally, this document analyzes the implementation of the project in detail, describing all the development phases and illustrating the support tools used, such as *Hawkbit* for deploying updates to the device and *Grafana* for data analysis. Special attention is given to the security aspect, both in the analysis of AWS services and in project implementation.

## 2 Contributions of the thesis

The thesis work's main contributions include defining a *Telematics Control Unit* (TCU) simulator that can emulate the collection of data from vehicle subsystems. Additionally, a basic cloud infrastructure was built using AWS services to receive the telemetry data generated by the simulator and manage the development and deployment of updates on the device. The deployment server was implemented using *Hawkbit*, a tool provided by *Eclipse* that is specifically designed for deploying software to IoT devices. Additionally, a *Grafana* server was implemented for data analysis. The proposed solution incorporates state-

of-the-art elements to advance the industry field.

To make this solution effective, simplifications were made to the overall SDV concept to reduce complexity. All design and implementation decisions were made while considering the optimal trade-off between available resources and fidelity to the real use case.

The remainder of this section will present the contributions related to the construction phase of the cloud infrastructure – Cloud Infrastructure – as the simulator for collecting subsystem telemetry – TCU Device Simulator – as the data analysis server – Grafana Server –.

## 2.1 Cloud Infrastructure

The fundamental element of the proposed approach is the cloud infrastructure built through the use of the services offered by AWS. The architecture can be represented as two distinct workflows: one dedicated to the collection of telemetry data coming from the device, and the other for managing the management of the development and deployment of device update updates.

Regarding the data analysis flow illustrated in the figure 1, a specific service for managing IoT devices is used, which is the *AWS IoT Core* to collect data via a channel using the *Message Queuing Telemetry Transport* (MQTT) protocol. The *IoT Core* service is also responsible for generating certificates to authenticate device connections. The data is then sent to a Kinesis stream, filtered by attributes and sent via a Lambda function to a Timestream database, which is in charge of temporarily storing the data.


**Figure 1:** High-level representation of AWS services for data management

In terms of update management and deployment flow, the figure 2 shows the use of other services. First in chronological order, an *Elastic Compute Cloud* (EC2) instance is created, on which the *Hawkbit* server is installed. The server parameters, such as IP address and user information, are then saved in the *System Manager*'s *Parameter Store*. Subsequently, a pipeline is created via the *Codepipeline* service, which consists of a *Source* stage, a *Build* stage, and a *Deployment* stage divided into three *Lambda* functions. The source stage uses a *Codecommit* repository, which takes commit events to trigger subsequent actions in the pipeline. When an update of a compiled script written in C code (as in the project) is required, the repository containing the update source code is used as input by the build stage. This stage generates the executable file from a specially created image placed in an *Elastic Container Registry* (ECR) registry, which is ready for deployment. At this point, the executable goes through the three *Lambda* functions that, by contacting the Hawkbit server with the information stored in the *Parameter Store*, respectively create the *Distribution Set* and the *Software Module* containing the update files on the server, create the *Roll Out*, and assign the *Distribution Set* to the device (or fleet of devices) to be updated, again using the API exposed by the *Hawkbit* server.

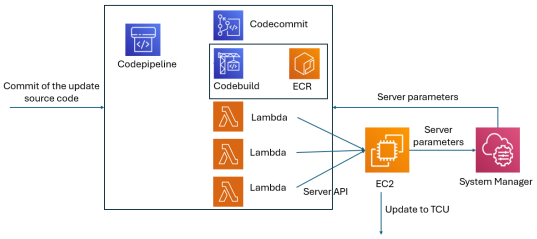


**Figure 2:** High-level representation of AWS services for the update management and deployment

These two execution flows in the cloud infrastructure appear to be separate, but they are linked by the device. The device receives updates and modifies performance and data produced, providing the necessary information to create new updates and continue this cycle throughout the device's life (and the potential vehicle's life). This passage demonstrates how the cloud services offered by AWS are a valid solution for the functions required for the development of the SDV.

## 2.2 TCU Device Simulator

The second essential element for the development of the thesis project is the *TCU Device Simulator*. It is a system that can collect and transmit data from simulated subsystems and integrate updates from external sources. The system is composed of four main elements:

1. *Data sender*, the component responsible for connecting to the cloud and sending the collected data;
2. *Update server connector*, the component responsible for connecting with the server that manages the deployment of updates;
3. *Data generator*, the component that collects data from the subsystems;
4. *Update agent*, the component that manages the activation of the update on the device.

The four components, working together, enable the device simulator to generate, collect, and send data to the cloud infrastructure, as depicted in the image 3. Additionally, the device remains in a constant state of readiness for any updates.

This system enables the creation of a device simulator that can interact with the AWS services for data collection and storage, as well as with the Hawkibt server for deploying updates. The purpose of this simulator is to represent, in a simplified manner, the actions that would occur on a real vehicle in the context of SDV.

**Figure 3:** Snapshot of the TCU device simulator in action

### 2.3 Grafana Server

The final component being analyzed is the data visualization element using the *Grafana* server. Specifically, a *Docker* image is utilized on a virtual machine to connect to the *AWS Timestream* database, retrieve real-time data, and display it to the user in easy to understand and intuitive dashboards. This is done to provide a concrete representation of the generated data, making it easier to view and highlighting the update performed on the device simulator. In practice, the update in this example creates a variation in the data generated, similar to what would happen in a real case where an update on the vehicle's performance modifies its characteristics and the resulting data.

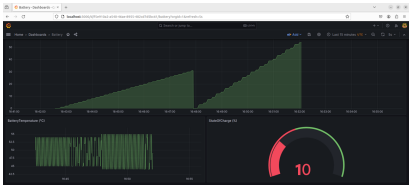

**Figure 4:** Snapshot of the Grafana battery dashboard of the device simulator in action

Image 4 displays a dashboard produced, highlighting the update on regenerative braking performance. The energy accumulated in the battery after the update is greater than before. However, the update also causes an increase in battery temperature peaks due to the increased performance and involvement of the subsystem in relation to the overall activities of the vehicle.

## 3 PoC: Implementation and Validation

The working *Proof of Concept* (PoC) was created to demonstrate the effective synergy between the various systems in practice. Each component element underwent a different development process based on work requirements.

To develop the cloud infrastructure, an introductory phase was initially adopted in which the various services were implemented using graphical or command-line console interfaces. Subsequently, a development approach was adopted using the *Cloud Development Kit* (CDK) and *Software Development Kit* (SDK) provided directly by AWS, and integrated with the Python language. The result is a script that uses the functions of these libraries to manage the entire cloud infrastructure with a single command. It is simple, fast, and optimized. The *TCU Simulator* was developed using a special approach. Python code was used for the connection to the cloud and the creation of the update agent. AWS SDK functions were used to manage the connection to the cloud, while the update agent was built from scratch specifically for this project. To manage updates through the *Hawkbit* server, the device simulator provided by *Hawkbit* was used, and it was adapted to fit the requirements of the PoC. Regarding the data generation and collection system, two different scripts were created for the *TCU Simulator*. One script was written in Python to ensure portability across different architectures, while the other was written in C language for updates compiled via the cloud pipeline of *AWS Codepipeline*. To implement dashboards in *Grafana*, a docker image was created on a virtual machine server. The dashboards were designed to display all the necessary information to understand incoming data related to a specific topic. The dashboards were connected to the cloud infrastructure and the *AWS Timestream* service.

Finally, the PoC was tested on a *Raspberry Pi* board, based on *Arm* architecture, to simulate an SDV use case as closely as possible. The validation demo reported excellent results with all tests carried out, demonstrating how the board was able to interact correctly with the cloud infrastructure and the data visualization server. After an automatic update was deployed, the system was able to download it, detect it, and activate it in the simulator execution, regardless of whether or not it had been compiled for the architecture.

## 4 Conclusions and Future Work

This thesis demonstrates the benefits of introducing the *Software Defined Vehicle* concept in the automotive sector, both for the industry itself and for the end user. It also explains how this paradigm can be implemented using the resources currently available in the sector, especially by taking advantage of the services offered by AWS, which guarantees a high level of attention to the security of its systems. The *Proof of Concept* achieved its objectives, but certain simplifications were made. In future work, it will be necessary to apply the concept to actual production in order to make it a tangible product. To achieve this goal, three main points must be developed. Firstly, the cloud infrastructure should be applied to a real vehicle to ensure compatibility with all subsystems in the real case. Secondly, it is important to conduct a more thorough analysis of the open software used for development, particularly the Hawkbit server, in order to fully customize it. Finally, it is essential to integrate the concept of SDV with the other information technologies in the vehicle, such as the cockpit system or machine learning functions. Several automotive companies are researching these technologies to achieve a more secure and efficient future in the industry.