IoT-event-analytics and Vehicle-Edge projects

> November 2021 - the development of these frameworks was halted. Suggestions made to the Bosch developers to consider building on top of existing frameworks such as DAPR, that might provide similar functionality.

Git repositories (unmaintained, see above): iot-event-analytics, vehicle-edge.

Announcement about this significant open source project creation initiated by Bosch can be found in this blog post

What this page contains:

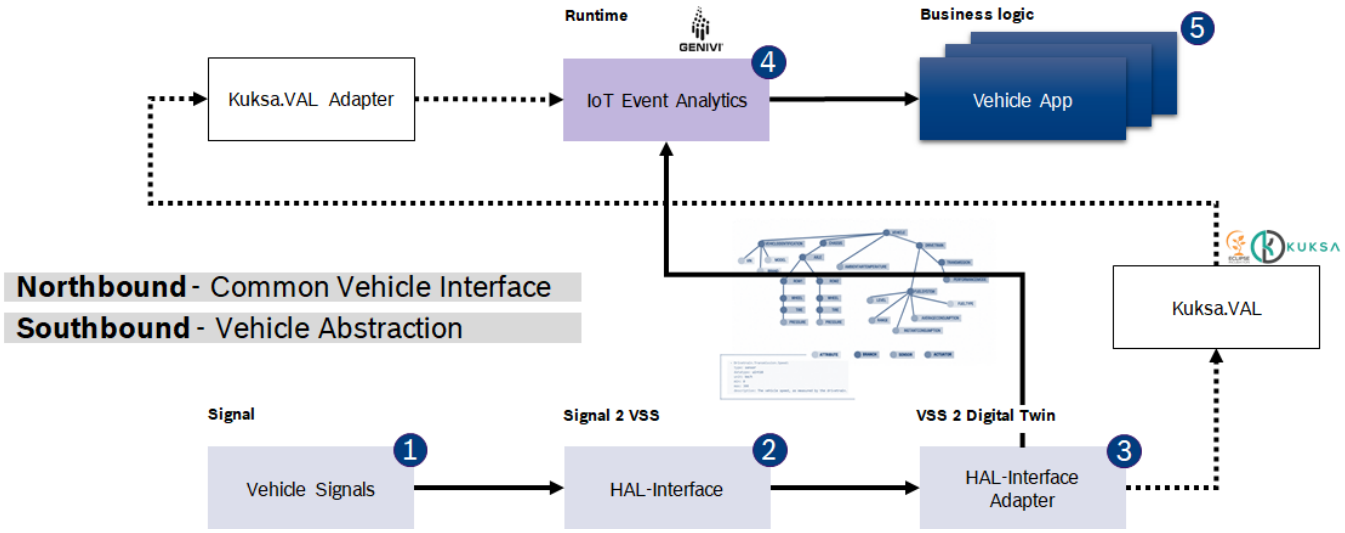
Collect detailed information about the IoT-event-analytics and Vehicle-Edge projects to explain the different components they are made up of.

Purpose:

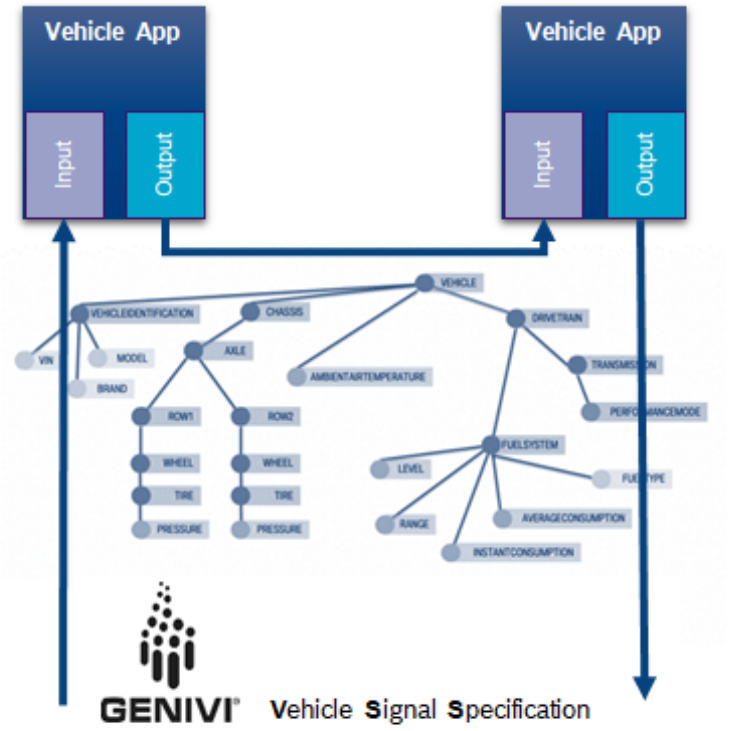
After understanding these projects in detail we can build a full plan for the desired CVII Technology Stack, using the components of these projects as a foundation and combined with other components.

Overview:

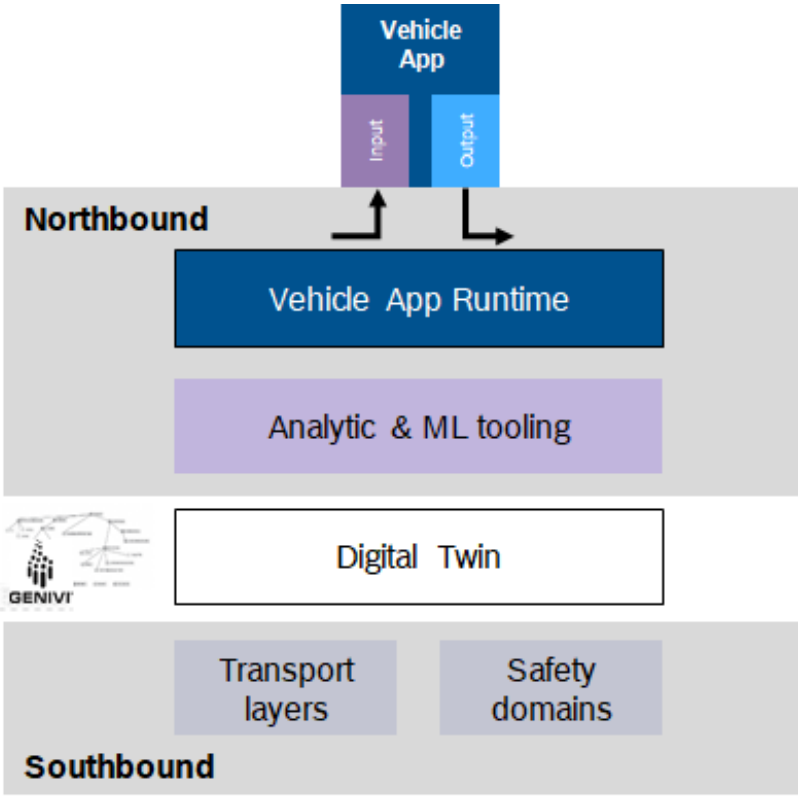
The projects implement a high-level vehicle data abstraction strategy as follows:



The framework's main purpose is to allow vehicle applications to communicate with each other, and with vehicle platforms, using a common data model, here implemented with the VSS.



The abstraction layers of app to platform in more detail:



The platform is useful for processing data before it is transferred to cloud infrastructure, thus being part of an edge-processing strategy. The vehicle is then considered an edge device.

IoT-event-analytics

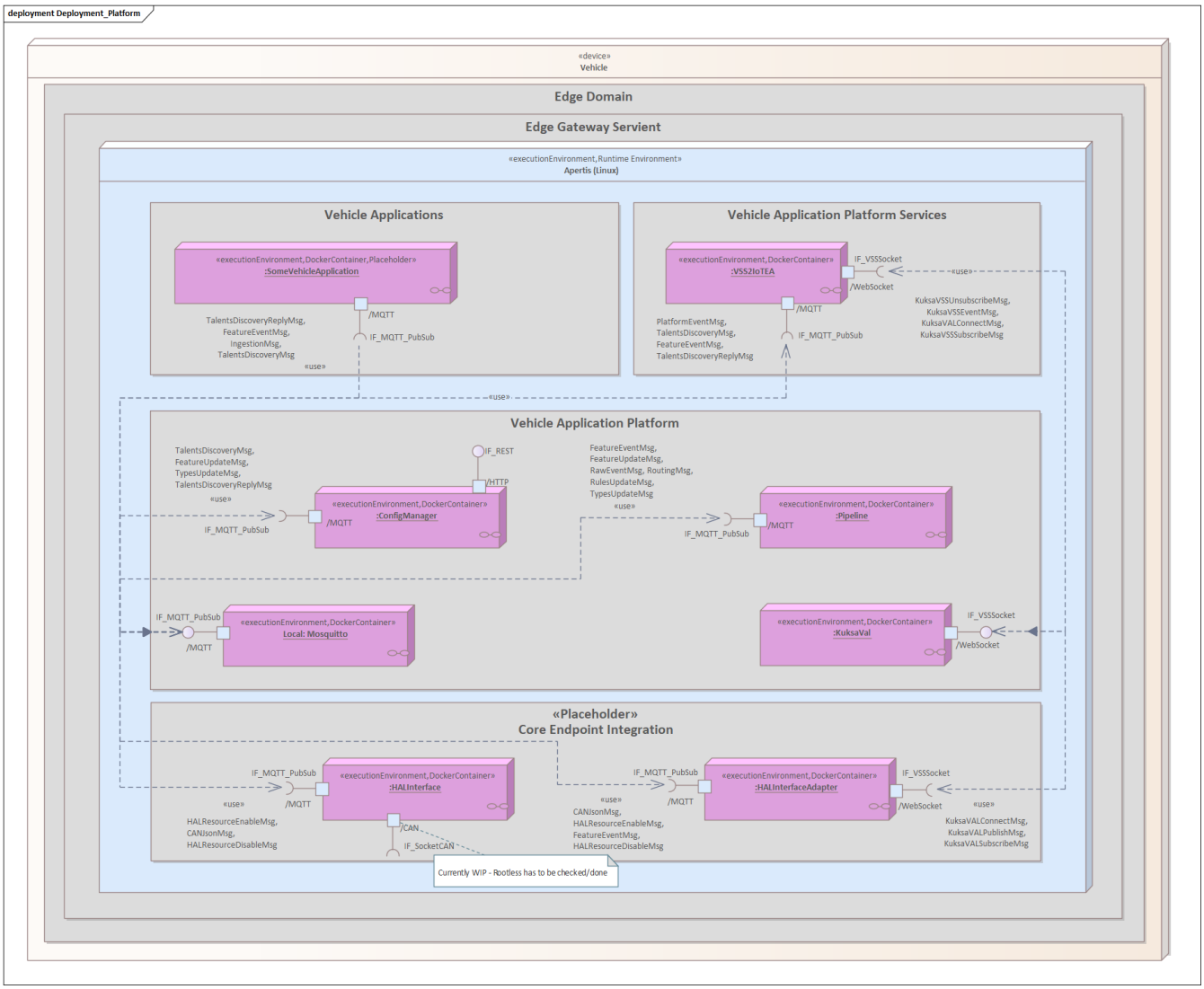
IoT-event-analytics (a.k.a. IOTea) implements an event processing and agent network platform. It orchestrates the flow of events between communicating agents (Talents) where the business logic of that interaction can be programmed using several different languages.

Vehicle-Edge

Vehicle Edge sets up a complete software stack around IOT-event-analytics by adding services such as an MQTT broker, and other components. Vehicle-edge uses containers to set up a network of communicating parts.

Details / analysis

UML model overview (provided by Bosch)



The docker compose file of vehicle edge shows the following dependencies (which order containers must be started)

blocked URL

Containers overview and quick references

Container name	Purpose ⚠ These descriptions need to be checked and updated by Bosch	Container image / Dockerfile	Original source code
Hal-interface	This typically collects signals from a low level vehicle-specific signal source such as CAN bus. For each technology, a different hal interface can be provided.		
Hal-interface-adapter	This implements the necessary translation from the specific signal interface to the standard one used in this framework (VSS). (In CCS terms these have been referred to as "vss-feeder" components, something that feeds the system with VSS formatted data, from some source, that might not be VSS formatted, i.e. a translation occurs)		
test-talent	Shows an example of a "talent" i.e. a cooperating agent in the system. A talent is....		
pipeline	Main component of IOT-event-analytics? - details TBD	built from pipeline /Dockerfile.amd64	
configmanager	Setting up the system together with pipelines? - details TBD Written in		
kuksa-val	Keeps the current value store and sits between VSS2IOtea and the hardware-specific parts. Kuksa.VAL exposes data using the VISS protocol (v1, websocket) , but also provides other protocols like MQTT.	amd64/kuksa imported from Jenkins build server e.g. kuksa-val-<hash>-amd64.tar.xz	
mosquitto-remote mosquitto-local	MQTT brokers (well-known implementation and not unique to this project).	built from mosquitto /Dockerfile.amd64	
platform = ?	Only an adapter between VISS and IoTea.	built from platform /Dockerfile.amd64	
vss2iotea		built from kuksa.val2iotea /Dockerfile.amd64 ?	

Software Components (not always 1-to-1 mapped to a container)

Software Component name	Provided interface	Link to source code	Programming language	Other info about framework/environment
(these names might need update)	*(is it a library with a header file, a process exposing a socket, a REST Web API, another well defined protocol)*			*(important libraries/frameworks/technologies used beyond the choice of programming language)*
Hal-interface				
Hal-interface-adapter				
test-talent				
pipeline				
configmanager		(source code in iot-event-analytics)	Javascript (Node JS)	
kuksa-val		(git-repository) (releases)		
mosquitto		(git repository)	C	
platform = ?				
vss2iotea			Javascript (Node JS)	

File/storage mappings

For convenience, each container maps a config file or directory to a location in the host file system, so that the configuration can be conveniently edited. With few exceptions, each container only sees its own config location and do not share any storage:

A starting example of all configurations is provided.

blocked URL

Interfaces and behavior

From the fact that these components are executing in different container namespaces and do not share storage, we can derive(?) that interaction between the parts are all using network protocols.

More analysis is required on what APIs are provided by each and how communication occurs:

blocked URL

Kuksa.VAL exposes the VISS protocol and this is the main "API" for others to interact with Kuksa.VAL

TODO: Determine the (network?) protocols. Specifically, what functionality does each component provide and consume? And which component communicates with which other one?

Referring to the UML diagram above:

Most components in the framework communicate internally using MQTT taking advantage of the pub/sub capability so that every component has access to the information it needs. To fully track the actual communication flow it would be required to see which topics are published and subscribed by each component.

VISS is an protocol that is provided to external clients by KUKSA.VAL.

Vehicle applications are (in UML overview) described as communicating with the framework using MQTT.

Questions

This needs to be installed from tar-file? IOTEA_JS_SDK=boschio.iotea-0.2.1.tgz