JANUARY 2024

Driving Forward: Best Practices for Frequent Over-The-Air Vehicle Updates

MongoDB.

Table of Contents

Introduction

Modern automobiles contain complex computing networks of up to 100 controllers running millions of lines of code. Automotive manufacturers strive to gain significant advantages from the ability to efficiently update this software. These updates help avoid costly fleet recalls by solving software and even hardware issues, opening new business models, and extending the general life of a vehicle by upgrading functionalities over time.

Traditionally, the process of updating the car software has been time consuming and frustrating for the vehicle owners who are forced to visit a dealership to receive new software installations.

This manual process is inconvenient for both the owner and the manufacturer for several reasons:

- It takes time and resources to disseminate software updates across a widespread dealer network, especially for OEMs with global operations.
- Dealers maintain a library of multiple software versions, which introduces the potential for errors and complexities.
- Relying on snail mail to notify vehicle owners about critical software updates is never an accurate method. It can result in missed notifications and overlooked updates, especially in the case of resale and change of ownership.
- Customer satisfaction suffers when the owner has to bring the car in every time.

Over-the-air (OTA) updates offer a promising solution to these issues. It also offers the potential for enhanced revenue streams as the OEM can offer add-on services through an OTA update. In fact, the majority of automotive OEMs regard OTA as the most impactful trend that will become a standard requirement, because it is the most visible and beneficial feature for end customers.

However, performing a reliable OTA process has its own unique set of challenges including:

- Reducing the update package size to transmit only essential modifications
- Safeguarding against unauthorized software and firmware changes
- Preventing vehicle owners from compromising security through unauthorized customizations
- Ensuring safe execution of updates in uncontrolled environments
- Preventing misprogramming of controllers for driver safety
- Ensuring consistent internet connectivity and power supply during updates
- Directing specific updates to various vehicle models based on their purchased options or aftermarket equipment installations

In this paper, we look at the requirements for a secure, efficient, and reliable OTA update mechanism and propose an architecture that would speed up the development process for efficient and stable OTA updates for the OEMs, all while increasing customer satisfaction.

Typical OTA Update Flow and Challenges

The traditional methodology for software updates can be represented by the flowchart below.

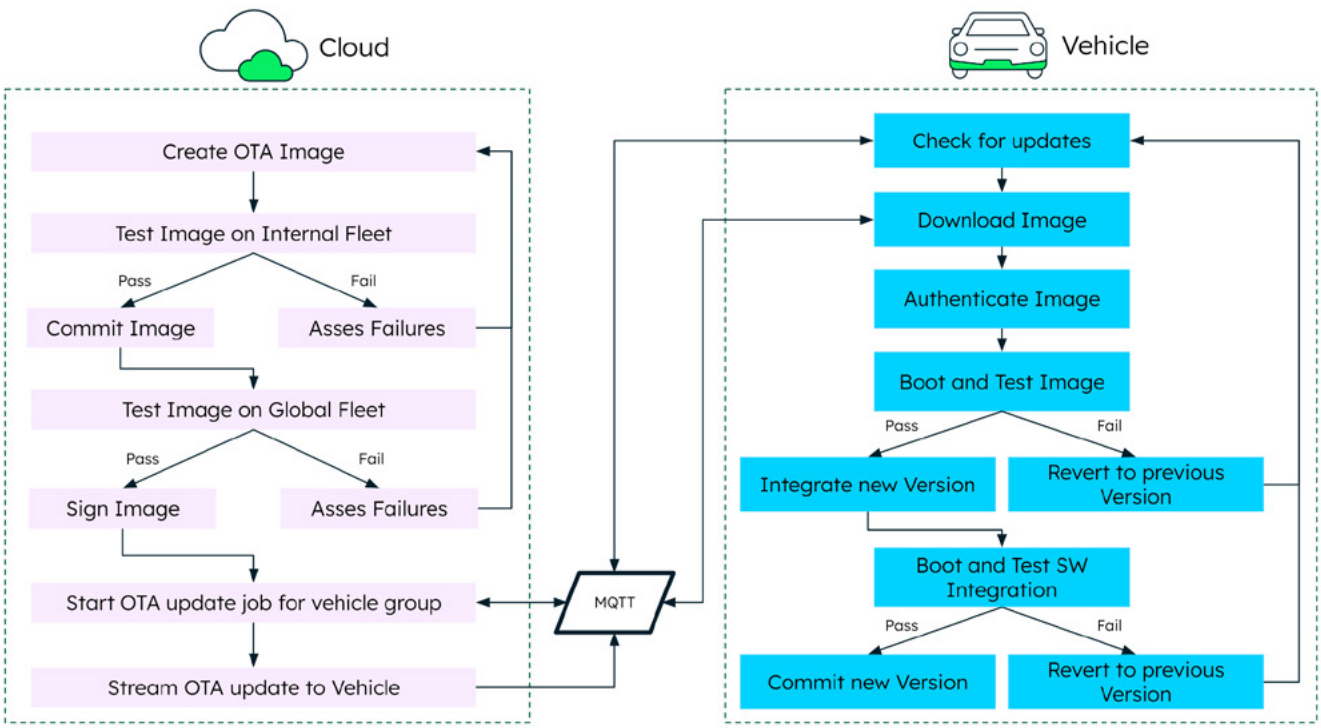


Figure 1. Typical software update flow for vehicles

Even though OTA updates can present many benefits, they also come with their own set of challenges:

- High network costs: Data transfer over the network can become extremely expensive at scale. As an example, an OEM selling 2 million cars per year will have 20 million cars on the road in 10 years. For a 10MB software update, there will be 200,000 TB of data transfer, not counting retries. Assuming a preferred network cost of $10 per GB, that amounts to $2 billion for a software update.
- Bandwidth limitations may affect updates: Some bigger updates may require a few GBs. With slow network connectivity, vehicles could stay blocked for many hours, resulting in a poor user experience.
- Resource intensive backend infrastructure: In terms of operational costs, it takes lots of resources to manage retries, connectivity, encryption, conflict resolution mechanisms, authentication, filtering, scalability, and high availability.
- Heterogeneous vehicle systems: Different configurations and update plans mean that updates are not just one-to-one. Sometimes one ECU has to update, then that triggers another update in another ECU and so on. There is no one-size-fits-all update methodology.

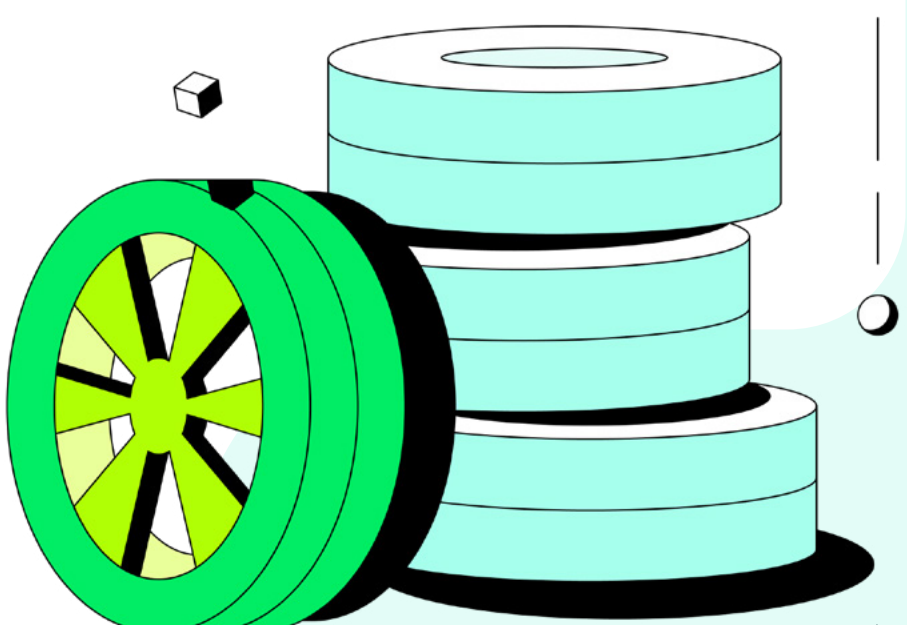
- Data transmission reliability: In case of connection loss, retry and fail-back mechanisms are needed, which means the car has to redownload data packages from the start to the detriment of the user experience.
- Comprehensive security measures: Investment in end-to-end encryption, secure authentication, and continuous monitoring for threats is crucial. This includes protecting the data in transit and at rest, as well as safeguarding against unauthorized access and tampering.

As you can see, although OTA updates offer immense benefits in terms of keeping vehicles up-to-date and enhancing features post-sale, they bring significant challenges in terms of network costs, system heterogeneity, infrastructure demands, bandwidth limitations, data reliability, and security requirements.

Requirements of a Robust Frequent OTA System

In order to overcome these challenges, a robust OTA system requires several key capabilities and features:

- Compressed and efficient data transfer: The amount of data transferred over the telco network must be reduced as much as possible by compressing the data, only syncing the necessary data, and avoiding complete re-starts when there's a failure (pick up update where it left off).
- Offline-first paradigm and user notifications: Notify users reliably via smartphone applications, or the vehicle infotainment system, and decrease retries as much as possible.
- Reliable, robust communication protocol: There should be conflict resolution, automatic retry mechanisms, and real-time monitoring of the update from the cloud.
- Flexible data model: Keep a library of current updates for all models, years, packages, and configurations and also keep track of historical updates, which typically requires more than one database in the cloud: one for current versions and another for historical versions. (We will cover this topic in further detail later.)
- Flexible system, easy to develop and maintain: Reduce as much as possible data piping work for engineers while ensuring efficient development processes. Otherwise, fast, error-free deployments aren't possible.
- Encrypted and secured system: A robust and resilient system requires authentication, authorization, encryption at rest and in flight.



Logical Concepts for a Robust Frequent OTA System

Let's break down and explain the logical concepts or building blocks for a robust frequent OTA system shown in Figure 2.

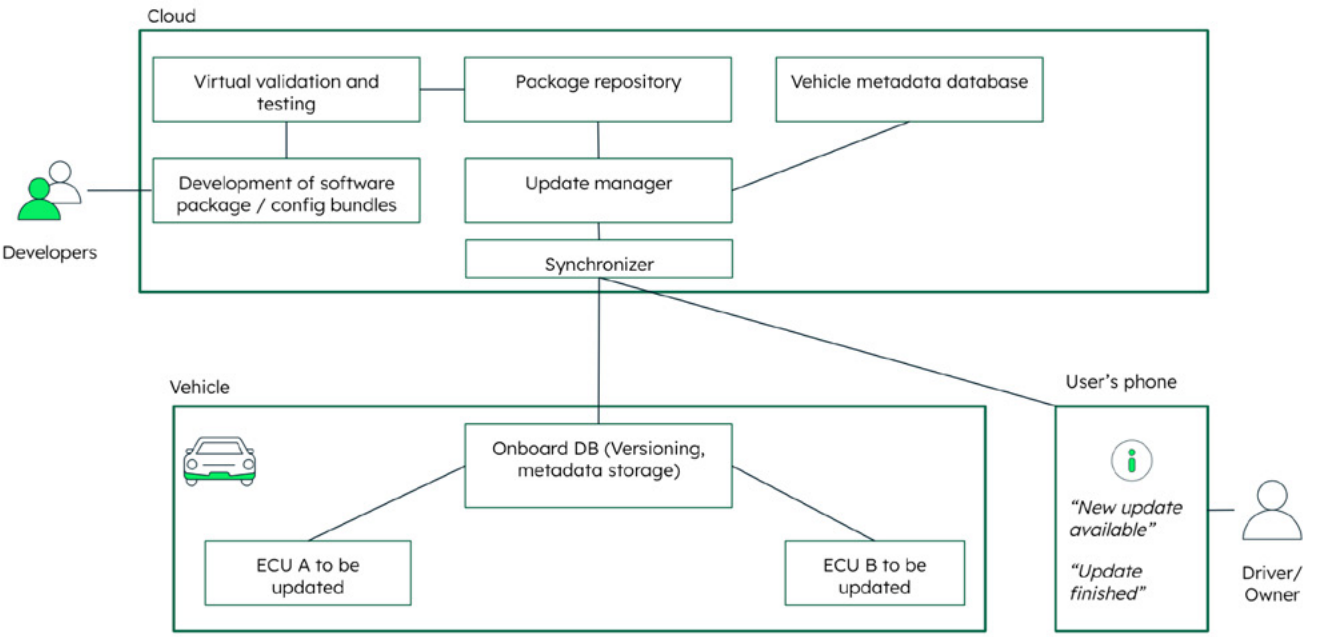


Figure 2. Building blocks for a robust frequent OTA process

Creating a reliable OTA update system for vehicles starts with a detailed plan that aligns with global standards, such as those outlined by the United Nations Economic Commission for Europe (UNECE). UNECE standards are essential for consistency and regulatory adherence in the automotive sector. Additionally, the OEMs have to take into account the principles of Software Update Management Systems (SUMS), which also includes variants management and handling. The OEMs must demonstrate that the SUMS is managed along the entire value chain.

The SUMS framework plays a crucial role in facilitating secure and well-controlled software updates. Due to diverse configurations in different car models, an effective variant management is necessary. Incorporating version control and rollback mechanisms ensures a seamless transition back to the previous software version in case of unexpected complications. Rigorous testing, encompassing both virtual and real-world scenarios, needs to be integrated into the update process to validate compatibility and functionality across various vehicle variants.

Vehicle

A traditional vehicle can have anywhere from 30 to 100 ECUs, and in more modern architectures, vehicles can have 100+ components across High-Performance Computing Units (HCPs), Domain Control Units (DCUs), and Zonal ECUs. A subset of these units need to be synchronized and updated. On each domain (Telematics Unit, Main HPC, Update manager, Gateway, etc.), there's the need to store the software versions and metadata of all the units belonging to that domain. Additionally, the data generated by these computing units can be stored and processed locally for analysis.

Such a database should follow an object-oriented paradigm so that the data can be modeled according to the Vehicle Signal Specification (VSS). VSS is an initiative by Connected Vehicle Systems Alliance (COVESA) to define a syntax and a catalog for vehicle signals. It can be used as a standard in automotive applications to communicate information around the vehicle, which is semantically well defined. It focuses on vehicle signals, sensors, and actuators with the raw data communicated over vehicle buses, and also focuses on data commonly associated with the infotainment systems. VSS includes standardized data definitions for vehicle signals while ensuring the same semantic understanding across different domains and also includes basic definitions for interfaces working on vehicle data (w3c, etc.). Figure 3 shows how example vehicle signals can be modeled via VSS. If we look at the VSS taxonomy in Figure 3, it has a vehicle as the root node whereas all the subsystems in the vehicle are represented by various branches. We also can see attributes such as VIN, model, and brand represented as leaf nodes.

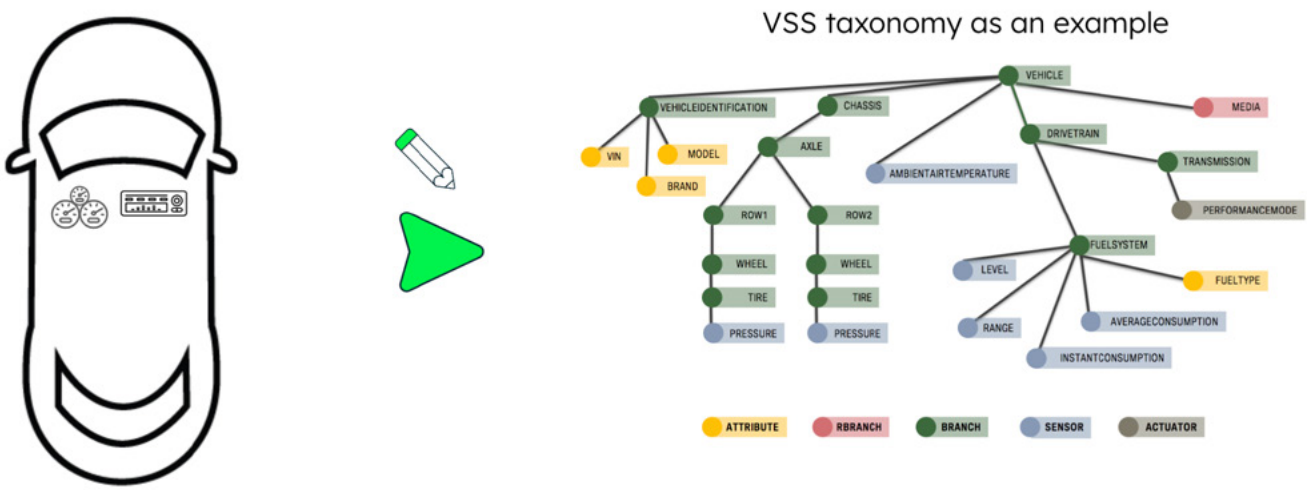


Figure 3. Example snapshot of a VSS tree

The local storage should also contain information about the vehicle's capabilities and applicable configurations, which can be identified and maintained in the cloud backend. This will help with management of vehicle diversity and makes it easy to remove, add, or update configurations directly from the cloud.

Finally there is a need to have a data synchronization mechanism between the vehicle databases on the different domain units and the cloud so that the OTA process can be initiated and monitored in real time. This data sync mechanism needs to take into account resolution of data conflicts, because vehicles could lose connectivity while being updated, creating a discrepancy between what's stored in the vehicle and what's stored in the cloud. Secondly, the data has to be synchronized with the backend in an efficient and compressed manner – as in the case of a large fleet of cars, the OEMs require to keep the mobile network operations costs low.

A simplified config file synchronization mechanism is represented in Figure 4. Note that the same sequence can be adapted for actual software download to the car.

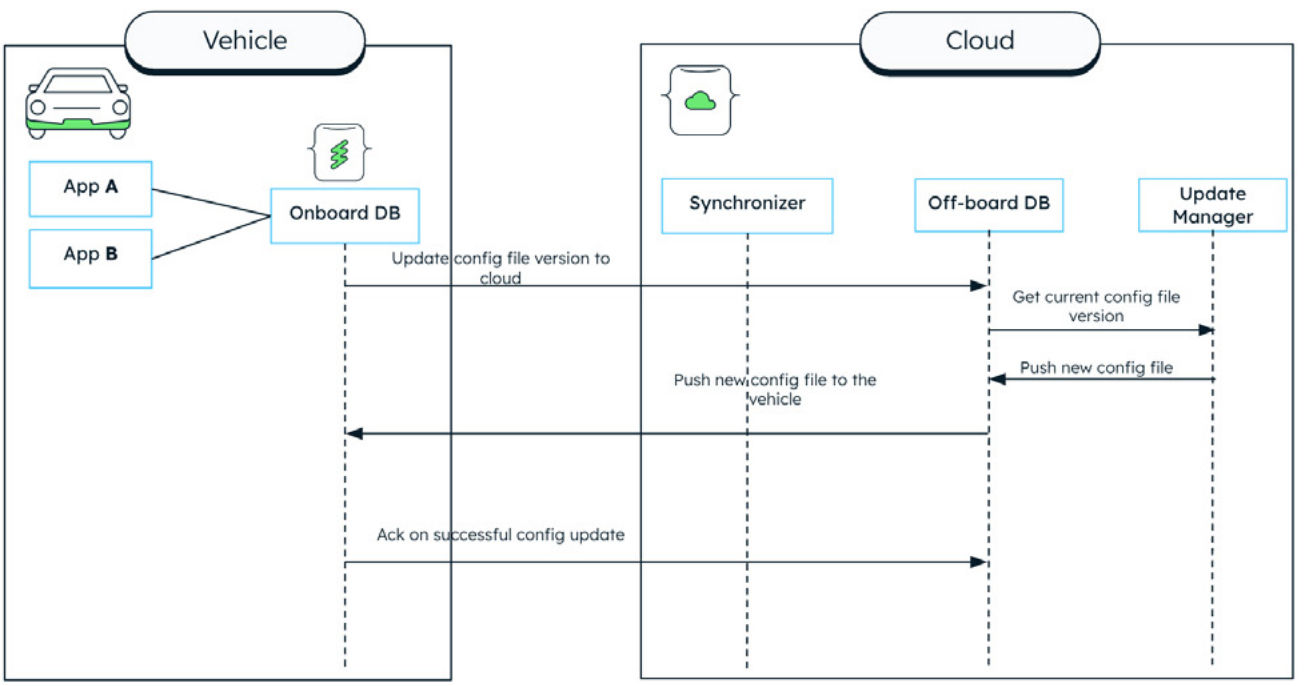


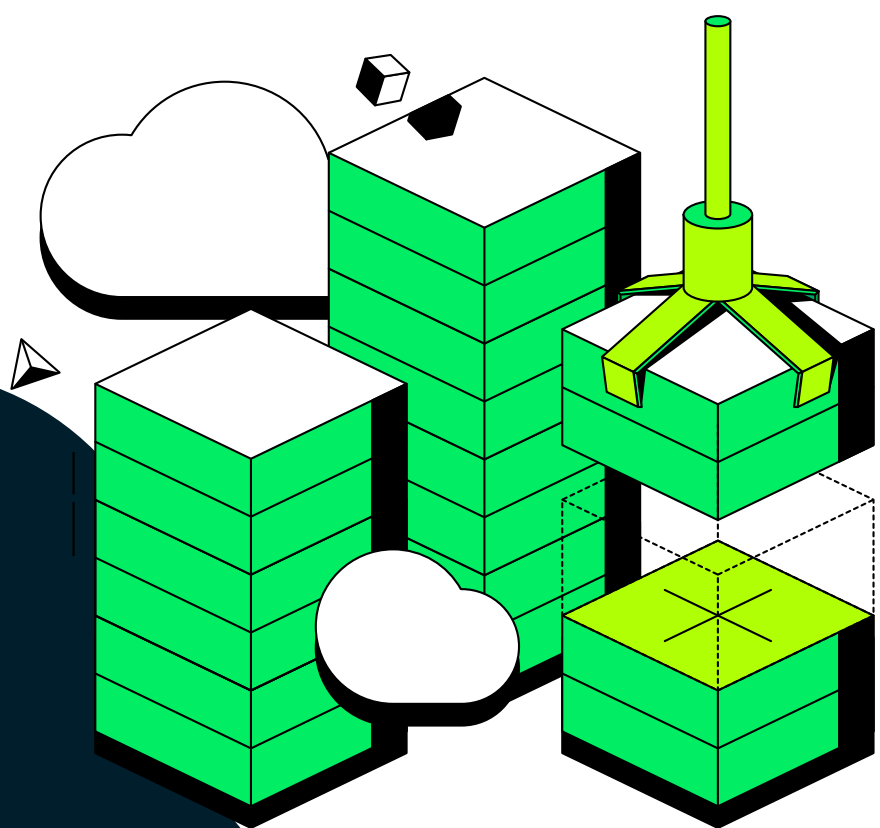
Figure 4. Configuration file synchronization sequence between cloud and vehicle

Cloud

There are a number of requirements for the cloud backend. There is a need for cloud object storage to efficiently store the vastly heterogeneous data (different models, years, packages, configurations, etc). These updates can be configuration files or actual software binaries that need to be pushed to the vehicle in the context of any vehicle feature update, activation, or deactivation. Additionally, the cloud needs to capture the software version history and other metadata associated with the OTA process. There can be millions of cars on the road and to be able to accurately track different software versions running on each car is extremely important for the OEMs. For storing metadata surrounding the actual update packages and handling synchronization with the vehicle, a fast, scalable, distributed, and highly available data platform is required. Finally, robust authorization and authentication protocols must be implemented to safeguard sensitive information and maintain the integrity of the update process.

Client Applications

Taking vehicle mobile applications as an example, key requirements for such applications include implementing robust security measures to authenticate users and protect sensitive information. Once authenticated and logged in, real-time communication channels need to be established between mobile app, vehicle, and the cloud, enabling the app to monitor vehicle status, receive alerts, and access diagnostic information. The incorporation of remote control functionality in the app empowers the users to manage crucial vehicle functions remotely. Similar to the vehicle database requirements, there is a need for an offline-first, embedded mobile database that stores user actions within the app and then pushes these changes to the cloud and the vehicle itself. When a new update is available for the vehicle, both the vehicle and mobile app get a notification instantly and once the update package is downloaded and applied on the vehicle, the mobile app gets another success notification. These requirements collectively contribute to a mobile app for vehicles that not only meets user expectations but also aligns with regulatory standards, fostering a secure and efficient connection between drivers and their vehicles.



How MongoDB Atlas Device Sync Facilitates Frequent OTA Updates

A vehicle is a resource-constrained embedded environment that is capable of generating an estimated 25 GB of data per day. For automotive OEMs, there is always a strong focus on resource management and efficiency. Every single MB of memory use has to be multiplied by millions of vehicles on the road to calculate the full impact of vehicle software updates and telemetry collection.

The MongoDB Atlas product portfolio provides OEMs three building blocks for building a reliable, robust and flexible OTA synchronizer:

- **MongoDB Atlas**: The fully managed, modern, multi-cloud database that provides MongoDB database as a service and enables Atlas Device SDK and Device Sync.
- **Atlas Device SDK**: A hugely popular object-oriented and embedded persistence layer or SDK for a wide variety of environments such as C++, Swift, Kotlin, Flutter, and more.
- **Atlas Device Sync**: A fully managed bidirectional data synchronization mechanism, connecting the offline-first Atlas Device SDK with the powerful cloud backend through a bi-directional data synchronization that is optimized for unreliable connectivity, with low bandwidth through transparent change set compression, and delta data synchronization including deterministic conflict resolution. Additionally, offline-first data sets can dynamically set filters for data sets to be synchronized, further increasing the efficiency of data transmission.

Let us reproduce Figure 2 with MongoDB components (Figure 5) and get a closer look at the three building blocks:

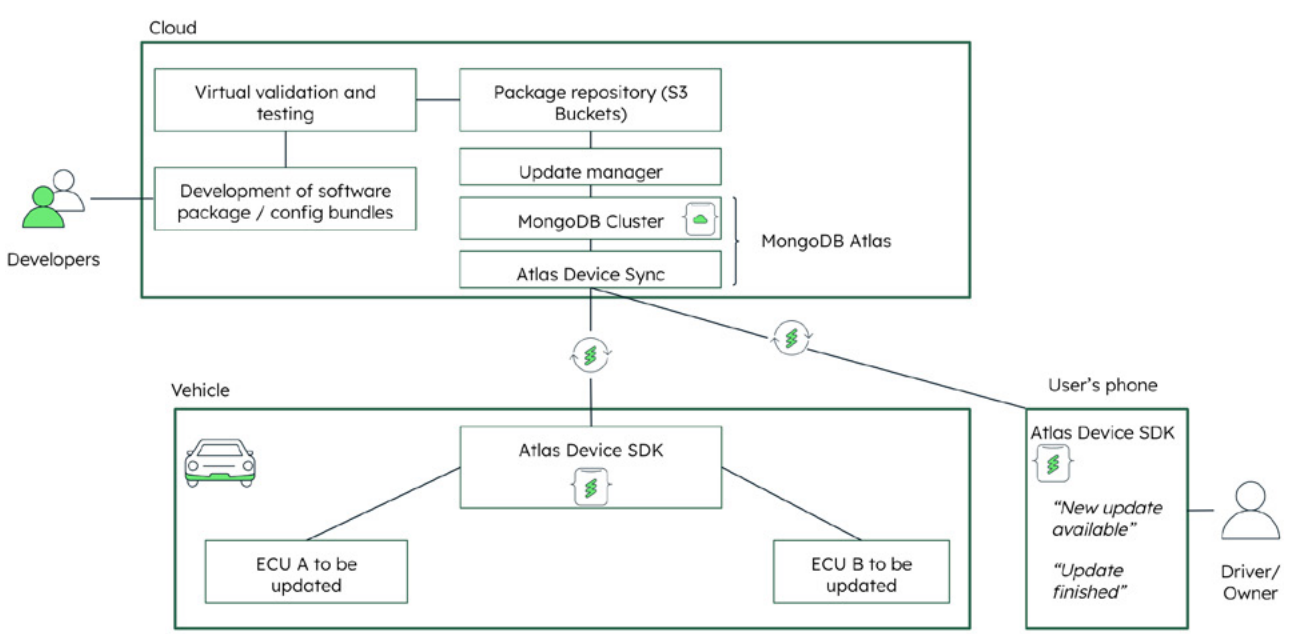


Figure 5. MongoDB-enabled building blocks for a robust frequent OTA process

MongoDB Atlas

Since vehicles produced by a single OEM can be distributed all over the globe, it is important for the cloud backend to be globally accessible. Multi-cloud clusters in MongoDB Atlas enables a single application to use multiple clouds simultaneously. With multi-cloud clusters, data is easily distributed across different public clouds, like Amazon Web Services (AWS), Google Cloud, and Microsoft Azure, enabling data mobility and resilience without the complexity of manual data replication.

Additionally, not only will the amount of data produced per car increase exponentially in the future, the structures and data models will also evolve. In anticipation of the exponential growth in data produced per vehicle, MongoDB Atlas, the modern, multi-cloud database built on the document model, provides the flexibility and horizontal scalability required to handle data coming in from millions and millions of connected vehicles.

Atlas Device SDK

MongoDB Atlas Device SDKs (previously known as Realm), a fast, scalable, and object-oriented alternative to SQLite is already utilized in millions of mobile applications, IoT devices, and infotainment systems. It was designed with resource constraints in mind. Atlas Device SDK runs right on client devices such as mobile apps or vehicle ECUs. Apps can access data as objects using the native query language for each platform. Storing, accessing, and updating data is simple and lightweight. The open-source SDKs are available for most popular languages, frameworks, and platforms (C++, Swift, Node.js, etc.). Each SDK is language-idiomatic to the development environment and includes the core database APIs for creating and working with on-device databases.

Atlas Device SDK follows an offline-first paradigm, persisting local changes before synchronizing them with MongoDB Atlas as soon as connectivity is established. The SDKs intelligently handle sudden network interruptions without compromising device battery life. To optimize for low bandwidth and reduce data transfer costs, the synchronization mechanism sends only the changes on a field level over the network, compressing these changesets before transmission. The distributed architecture and offline-first approach include built-in deterministic conflict resolution within the Atlas Device Sync protocol, freeing developers from the intricacies of conflict resolution and enabling them to focus on feature development.

Atlas Device Sync

MongoDB provides out-of-the-box bidirectional data synchronization through Atlas Device Sync between different Device SDKs (Realms), in client applications such as vehicle ECUs or mobile apps, and MongoDB Atlas on the cloud running on the OEM's choice of cloud.

Putting together these components, OEMs can enable frequent, reliable OTA pipelines with vehicles on the road. Figure 6 illustrates the whole process step by step. The vehicle VSS model is stored in Atlas Device SDK (1), which helps consolidate data across different ECUs and apps in the vehicle. When a new configuration or software update is available, Atlas Device Sync sends a notification to the vehicle (2) and the notification is synchronized with the user phone as well (3). Atlas Device SDK sends back an acknowledgment to MongoDB Atlas that the vehicle is ready for the update (4). The file is then downloaded (5) and upon successful installation, an acknowledgement is sent back to the cloud backend (6) and the user is notified via the mobile app (7). On the cloud backend side, as data grows over time, Atlas Online Archive is triggered to move the historical data into a cloud-managed object storage for cost efficiency (8). The data in operational and analytical data planes can be brought together for trend analytics using a unified API via Atlas Data Federation (9). Using Atlas Device Sync, OEMs get the necessary tools for monitoring the status and performance of the OTA update process. The same data pipeline can be leveraged to keep track of all the software versions running in various applications and ECUs in the car.

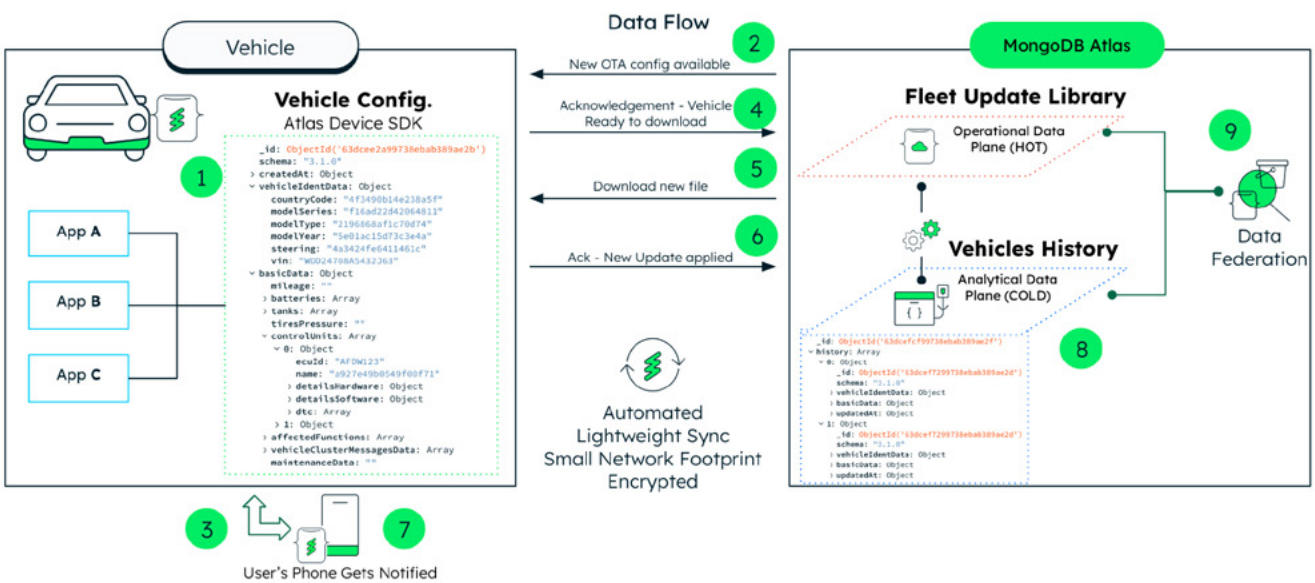


Figure 6. OTA Update process synchronized and monitored via Atlas Device Sync

Summary

This white paper addresses the challenges faced by automotive manufacturers in updating the intricate software systems embedded in modern vehicles. It emphasizes the inefficiencies of traditional manual update processes, highlighting the associated costs and inconveniences for both manufacturers and vehicle owners. The adoption of over-the-air (OTA) updates is a promising solution to streamline the update process and increase customer satisfaction. However, the unique challenges of implementing a reliable OTA process, such as data transfer costs, heterogeneous vehicle systems, and security concerns have to be addressed.

The proposed architecture for a robust frequent OTA (FOTA) system is outlined, encompassing compressed and efficient data transfer, an offline-first paradigm, reliable communication protocols, a flexible data model, and a scalable cloud backend database. The role of MongoDB's Atlas Device Sync in facilitating frequent OTA updates is highlighted, emphasizing its capabilities in handling resource-constrained embedded environments in the vehicle, bidirectional data synchronization, and offline-first local storage. The integration of MongoDB Atlas, Atlas Device SDK, and Atlas Device Sync is proposed as a comprehensive solution for OEMs to build a reliable, robust, and flexible OTA synchronizer. MongoDB offers OEMs the necessary tools for monitoring the status and performance of OTA update processes, ensuring a seamless and efficient connection between drivers and their vehicles.

To conclude, implementing these best practices and leveraging technologies like MongoDB's Atlas Device Sync can enable automotive OEMs to conduct frequent, reliable OTA updates, enhancing customer satisfaction and reducing operational costs. The combination of vehicle data management, cloud storage, and client application integration forms a comprehensive approach to addressing the challenges associated with OTA updates in modern vehicles.

To learn more about MongoDB's role in automotive industry, please visit our manufacturing and motion webpage.

About Capgemini Invent

As the digital innovation, design and transformation brand of the Capgemini Group, Capgemini Invent enables CxOs to envision and shape the future of their businesses. Located in over 30 studios and more than 60 offices around the world, it comprises a 12,500+ strong team of strategists, data scientists, product and experience designers, brand experts and technologists who develop new digital services, products, experiences and business models for sustainable growth.

Capgemini Invent is an integral part of Capgemini, a global business and technology transformation partner, helping organizations to accelerate their dual transition to a digital and sustainable world, while creating tangible impact for enterprises and society. It is a responsible and diverse group of 340,000 team members in more than 50 countries. With its strong over 55-year heritage, Capgemini is trusted by its clients to unlock the value of technology to address the entire breadth of their business needs. It delivers end-to-end services and solutions leveraging strengths from strategy and design to engineering, all fueled by its market leading capabilities in AI, cloud and data, combined with its deep industry expertise and partner ecosystem. The Group reported 2023 global revenues of €22.5 billion.

Get the future you want | Visit us at www.capgemini.com/invent

About the Authors

Humza Akhtar, Ph.D. is a Principal on the Industry Solutions Team at MongoDB focused on manufacturing and IoT use cases. He designs use cases for Smart Manufacturing using the MongoDB modern, multi-cloud database platform. Prior to joining MongoDB, he worked at Ernst & Young Canada as a Senior Manager, in digital operations consultancy practice. Humza attained his Ph.D. at Nanyang Technological University, Singapore, and worked with the Singapore manufacturing industry for a number of years on Industry 4.0 research and implementation. He has spent most of his career enabling smart and connected factories for many manufacturing clients. In 2020-2021, he established a multi-year strategic roadmap for Singapore's smart supply chain initiatives.

Arnaldo is a Senior Specialist on the Industry Solutions Team at MongoDB focused on automotive, manufacturing, and IoT applications. Before MongoDB, he worked as a data science and revenue operations manager at 1build, a Y-Combinator startup based in San Francisco. As the first data science hire, he contributed to building the core data initiatives in the company for more than two years. He holds a Mechanical Engineering degree and a Master of Science in Analytics. Over more than eight years of experience, he has worked for one of the Top 100 global automotive suppliers and developed various robotics applications, which have given him a unique background based on the intersection of data-driven projects, software development, and automotive innovation.

Sherif Hussein is a recognized expert in the automotive industry with 20 years of international experience. Currently, as the Director of Software Defined Vehicles (SDV) at Capgemini Invent, he spearheads the SDV initiative. His expertise spans the development of new operational models, and the implementation of cutting-edge technical solutions. Moreover, he specializes in crafting outsourcing strategies and setting up Delivery Centers, aiming to forge innovative solutions for the future of mobility. Prior to his tenure at Capgemini, Sherif cultivated his expertise at IBM, where he led the global practice for embedded systems, managed IBM's Global Delivery Center, and contributed as a SME within IBM's Global Automotive Team.

Legal Notice

This document includes certain "forward-looking statements" within the meaning of Section 27A of the Securities Act of 1933, as amended, or the Securities Act, and Section 21E of the Securities Exchange Act of 1934, as amended, including statements concerning our financial guidance for the first fiscal quarter and full year fiscal 2021; the anticipated impact of the coronavirus disease (COVID-19) outbreak on our future results of operations, our future growth and the potential of MongoDB Atlas; and our ability to transform the global database industry and to capitalize on our market opportunity. These forward-looking statements include, but are not limited to, plans, objectives, expectations and intentions and other statements contained in this press release that are not historical facts and statements identified by words such as "anticipate," "believe," "continue," "could," "estimate," "expect," "intend," "may," "plan," "project," "will," "would" or the negative or plural of these words or similar expressions or variations. These forward-looking statements reflect our current views about our plans, intentions, expectations, strategies and prospects, which are based on the information currently available to us and on assumptions we have made. Although we believe that our plans, intentions, expectations, strategies and prospects as reflected in or suggested by those forward-looking statements are reasonable, we can give no assurance that the plans, intentions, expectations or strategies will be attained or achieved. Furthermore, actual results may differ materially from those described in the forward-looking statements and are subject to a variety of assumptions, uncertainties, risks and factors that are beyond our control including, without limitation: our limited operating history; our history of losses; failure of our database platform to satisfy customer demands; the effects of increased competition; our investments in new products and our ability to introduce new features, services or enhancements; our ability to effectively expand our sales and marketing organization; our ability to continue to build and maintain credibility with the developer community; our ability to add new customers or increase sales to our existing customers; our ability to maintain, protect, enforce and enhance our intellectual property; the growth and expansion of the market for database products and our ability to penetrate that market; our ability to integrate acquired businesses and technologies successfully or achieve the expected benefits of such acquisitions; our ability to maintain the security of our software and adequately address privacy concerns; our ability to manage our growth effectively and successfully recruit and retain additional highly-qualified personnel; the price volatility of our common stock; the financial impacts of the coronavirus disease (COVID-19) outbreak on our customers, our potential customers, the global financial markets and our business and future results of operations; the impact that the precautions we have taken in our business relative to the coronavirus disease (COVID-19) outbreak may have on our business and those risks detailed from time-to-time under the caption "Risk Factors" and elsewhere in our Securities and Exchange Commission ("SEC") filings and reports, including our Quarterly Report on Form 10-Q filed on December 10, 2019, as well as future filings and reports by us. Except as required by law, we undertake no duty or obligation to update any forward-looking statements contained in this release as a result of new information, future events, changes in expectations or otherwise.

Resources

Case Studies (mongodb.com/customers)

Presentations (mongodb.com/presentations)

Free Online Training (university.mongodb.com)

Webinars and Events (mongodb.com/events)

Documentation (docs.mongodb.com)

MongoDB Atlas database as a service for MongoDB (mongodb.com/cloud)

MongoDB Enterprise Download (mongodb.com/download)

MongoDB Realm (mongodb.com/realm)

Contact MongoDB to learn more about
Over-The-Air Vehicle Updates