Nuvalence

ARCHITECTURE GUIDANCE SERIES

Architecting a Transportation Cloud

WHITEPAPER

NUVALENCE.AI

IMPACTFUL SOFTWARE SOLUTIONS

Purpose & Audience

Nuvalence specializes in architecting and developing digital platforms for large organizations. We've written extensively on the topic of the general architecture of digital platforms, including a whitepaper named The Anatomy of a Digital Platform. The whitepaper you are now reading is a domain-specific extension of The Anatomy of a Digital Platform, focused on mobility and transportation platforms in the automotive industry. While reading The Anatomy of a Digital Platform first is recommended, one will be able to read this paper without that as a prerequisite and still understand a majority of the content.

The intended audience for this whitepaper is a technical leader within the automotive or general mobility sector. Individuals in this audience usually have Chief Technology Officer, Head of Digital Platforms, or Chief Software Architect titles or similar background. The purpose of this whitepaper is to help these individuals and their teams map business and systemic realities in transportation use cases (particularly autonomous vehicle use cases) to a technical architecture. The details in this paper are based on real-world experience in mobility and structured thinking on the general topic of building transportation clouds.

Understanding the Context for a Transportation Cloud

Consumer demand for "connected devices" exploded with the advent of the smartphone. The two reasons for this are clear:

1. If computing power and internet connectivity could fit in one's pocket, anything could be augmented to be "smart" and "connected".
2. With this newfound in-pocket computing power, everything else that was augmented with computing and connectivity could now be directly controlled by a smartphone.

The natural result for the automotive industry was to create connected vehicles. For the first time, cars were embedded with computing power and eventually, with internet connectivity. This meant that an individual could start their car on cold winter days no matter where they were, check maintenance status while watching TV in their living rooms, or remotely close windows during a sudden burst of summer rain. All from their phone. Additionally, vehicles started tracking meaningful telemetry to report on health and risk conditions, making that available for backend systems to take action or for data scientists to perform historical analysis.

Vehicles are undergoing a second era of transformation. In a time where data is worth more than ever before, vehicles are being equipped with the ability to record massive amounts of telemetry data. The ability to control all aspects of the vehicle by API has broadened. Possibly more importantly, vehicles are moving away from being powered by fossil fuels to being powered by electricity, all while becoming autonomous. Use cases for what vehicles can do are changing as a result. Electric, autonomous, data-dense vehicles can be used for anything from delivering pizza and medical equipment to the safe transport of loved ones to the airport. Need more use cases? How about a patrol vehicle navigating our communities to increase neighborhood security. Being able to connect to a vehicle from a phone is no longer considered advanced, but instead, has become table-stakes. What OEMs now need are "transportation clouds" that take advantage of autonomy being embedded in vehicles. Transportation clouds exist to make these vehicles, and the real-world use cases they operate in, easily programmable. But what is a "transportation cloud"?

A transportation cloud is a digital platform that exposes programmable "building blocks" to enable the creation of solutions powered by autonomous vehicles and data. These use cases can be B2B or B2C, impacting everything from how the companies transport goods to how one might hail a taxi. A transportation cloud includes the cloud-based APIs, data, and vehicle connectivity components provided to developers as a coherent platform equipping them to build mobility-enabled solutions. Are you interested in partnering with an OEM to build a mobile application to provide a courier service to consumers via self-driving cars, or an app to analyze vehicle congestion? An OEM's transportation cloud is the cloud-based backend for those and many other use cases. Without a transportation cloud, democratizing the creation of mobility solutions becomes impossible.

The next natural question is: what does the architecture for a transportation cloud that enables "programmable transportation" look like? Answering that question requires understanding the unique considerations specific to the mobility domain.

Key Architectural Considerations

In a transportation cloud, the core architectural tenet is enabling developers to treat vehicles, their data, and their tasks as resources they can program to build new "mobility enabled" solutions. Partners to OEMs integrate via APIs with a transportation cloud to gain control of a vehicle to perform tasks, control the vehicle's in-cabin experience, and orchestrate sophisticated logistics outcomes. This is not dissimilar from many other digital platforms. The pattern is to create a robust, scalable system that is exposed to 3rd parties who can build new commercial solutions. The aforementioned implies five specific key considerations that build atop of the general considerations presented in The Anatomy of a Digital Platform whitepaper. A summary of those considerations is as follows, followed by a section with deeper dive commentary:

1. **Transportation clouds need to optimize for both data and workflow heavy workload expectations:** Digital platforms are often optimized for workflow use cases (e.g. transactions and logic) or data use cases (e.g. analytics). Given the intense reliance on both active workflow execution and both real-time and at-rest data, architectures supporting mobility use cases aren't aligned more with one than the other. They're aligned with both equally and a supporting transportation cloud must optimize for both workload types. This challenge is best solved by creating purpose-built sub architectures: one focused on data, the other focused on orchestration, and thoughtfully fusing those together into a clean platform experience.
2. **Transportation cloud transactions are event-driven, with low predictability on event timing, event reliability, and ordering:** When building a platform for an industry like banking, predictability of transactions drastically simplifies things. Moving money, creating accounts, and other banking transactions are very deterministic in how long they generally take, what order things execute in, and what success and failure look like. This simply isn't the case with transportation. Successful trips could be minutes long, while others could be hours long. Vehicles, people in those vehicles, and the real world context they all operate in all do lots of unpredictable things and in unexpected orders. What system, for example, can plan for a sudden rainstorm that makes driving unsafe? Transportation clouds need to have an event-oriented architecture with appropriate support for event processing and associated state management.

3. **Vehicles are a primary and chaotic actor:** In many industries, digital platforms support primarily deterministic inputs and systems. Platforms that interface with real world actors often face the opposite: significant non-determinism, which leads to unique architectural and engineering requirements. While related to (2), it is important to call out that the core actor in a transportation cloud, the vehicle, can exhibit chaotic behavior. This requires a "failure is expected" architecture to support the potential failure of the focal point of nearly every transaction in the system: a vehicle.
4. **Streaming architecture characteristics are part of expected systems behavior:** Most "standard" digital platforms execute workflow logic that depends on static data at-rest (e.g. load data to support logic being executed). Executing that logic follows traditional patterns: run logic, load data, reason over the data, and influence the logic. The data component has little "critical path" architectural expectations given its at-rest nature. Transportation clouds are different. Central transportation clouds need to initiate workflows that then depend not on static data, but on real time streaming data such as vehicle position streams. This implies that while events, per consideration (2), trigger workflows, those workflows need to rely on streaming data to make decisions. Streaming data is generally real time, which almost always implies that the value of the data diminishes as it gets older. The result is that latency can drastically reduce the value of data, or in some cases, make it completely irrelevant.
5. **Governance, Regulatory & Compliance (GRC) requirements are yet to be defined, requiring a flexible architecture:** Transportation cloud services, particularly when coupled with autonomous vehicles, may have meaningful GRC implications. Unfortunately, because of the nascent stage of the industry, many of those implications are not yet known. Industries like banking and aviation have been regulated for decades. As a result, the approach to Governance, Risk & Compliance (GRC) is more stable. GRC in autonomous mobility and transportation is in its infancy, with much of it still needing to be defined. That said, a transportation cloud can't ignore GRC needs. Instead, a transportation cloud needs to provide flexibility that as GRC expectations emerge, they can be accounted for using a bespoke transportation cloud built today. The result is that a transportation cloud must be built to adhere to, and iterate on, evolving GRC requirements.

Each of these considerations could be written about at depth. While this paper will not cover each to the fullest extent, it will explore how each consideration impacts the architecture of a transportation cloud.

1. Transportation clouds need to optimize for both data and workflow heavy workload expectations

While most platforms have both data and workflow components, one tends to drive the core orientation of the platform (as described in The Anatomy of a Digital Platform). Transportation clouds have a more sophisticated need than a standard platform: both workflow and data components take equally prominent roles in core use cases. Building a single platform that optimizes for both data and workflow use cases would be extraordinarily difficult. A better approach is to build a transportation cloud that fuses a workflow-oriented platform with a data-oriented platform. Understanding why this approach is recommended is best understood by example.

1. **WORKFLOW** - the flagship use case for a transportation cloud is to allow its consumers to programmatically move a vehicle from one location to another. Consumers of the platform build their use cases on top of that primitive: ride hail, food/package delivery, patrol vehicles, and many other yet-to-be-imagined use cases. Behind the scenes there are a number of workflows internally which support this (vehicle management, billing and payments, routing, etc).
2. **DATA** - the workflow use cases tend to rely not only on data at-rest, but on real time data coming from real world actors like people and vehicles. Some of that data may have stream-like semantics (this is broken out as a core consideration for transportation clouds and discussed later in this paper). The result is that data plays not only the traditional role of being stored/retrieved, but also plays a role in analytics/optimization, and it actively modifies real time outcomes through the interplay with active workflows. Data use cases in transportation clouds include:
    c. Analysis of historical data to optimize for key performance indicators (eg: utilization of vehicles, customer satisfaction ratings, capacity/load ratio).
    d. Onboarding many vehicle providers - while an MVP will focus on one flagship vehicle model, over time onboarding vehicles can be simplified and even productized. Building data ingestion workflows with the assumption that, over time, the platform will receive data from many types of vehicles in potentially different formats or via different protocols allows the platform to be adaptive as technology changes and improves over time.
    e. Productization of this data - the platform will amass rich data on traffic patterns and transportation trends; this is impactful data for a number of use cases beyond functionality of the transportation cloud itself.

Taking the approach of fusing a workflow and data platform into a single layer ensures that the features in each can optimally support use cases in their respective categories. Furthermore, this sort of clear architectural delineation will help features evolve more cleanly over time since they can remain aligned with their respective domains.

2. Transportation cloud transactions are event-driven, with low predictability on event timing, event reliability, and ordering

Many transportation workflows will begin with some event generated by a user or a vehicle. These events are non-deterministically generated and may not follow an expected or prescribed pattern. A transportation cloud should heavily rely on distributed, event-based models. Platforms of this type should leverage queueing of events to allow for greater durability and scalability of the platform. Attempts to use traditional request/response style architectures would ensure the opposite: lack of scalability and low availability. Examples of transactions that warrant an event-driven core architecture model include:

1. All trip management transactions, including requesting trips, modifying trip parameters, and canceling trips. An event-based architecture allows a transportation cloud to reconcile bidirectional events (from the cloud to the vehicle and vice versa) to transactions currently being processed. For example, while creating a new trip is a trivial request, knowing that a vehicle has accepted it and is on-task for the trip is a multi-leg communication with many potential points of failure and latency. Waiting for a transaction of this type to complete would undoubtedly result in timeouts or complex remediation logic if not event-driven.

2. Vehicle control directives such as remotely unlocking a vehicle or requesting that an AV pull over. Similar to (1), a cloud platform requesting to control in-cabin or vehicle functions cannot block waiting for a result. Blocking and waiting for a result would meaningfully impact scalability and reliability.
3. Unexpected events generated by the vehicle. While (1) and (2) exemplify scenarios where the cloud originates a transaction and then reconciles events related to that transaction with the original call, sometimes a vehicle will publish an event to the transportation cloud. In these circumstances, the vehicle is simply notifying the cloud of some issue or change in state (such as an unexpected pullover event). This would trigger logic specific to that vehicle-originated signal. There is no practical way to implement a system for this that isn't event-based, particularly given the "fire and forget" nature from the vehicle's point of view.

Clearly, exceptions to this rule exist. In circumstances where transaction times are within predictable bounds (and don't depend on real world actors) and scalability may not be a critical concern, a request/response style pattern is best suited.

What makes this an especially important consideration is the skill set required to build this sort of backend. Architects and developers who have built at-scale cloud and distributed architectures need to be involved to build this successfully.

3. Vehicles are a primary and chaotic actor

While connected vehicles have become commonplace, the majority of time a consumer does not care if their vehicle is actively connected. From the consumer experience point of view, data generated during periods of connectivity loss isn't critical to the operation of the vehicle.

Within a transportation cloud, data that is not important to the consumer or partner is critical to operations of key services, so safeguarding for slow and intermittent connectivity should be planned upfront for all interactions with the vehicle.

1. **BIDIRECTIONAL COMMUNICATION** - Communication to and from the vehicle should leverage some bidirectional streaming protocol which supports buffering on each side of data which can't be sent due to connectivity issues. Consider assigning event priorities. For example, events generated periodically (eg: fuel level, current location) decrease in value as they become stale in that buffer, this data can "age out" or be deferred for more relevant events when connection is reestablished. If that low priority data is useful only from the historical analysis perspective, it can be written to a disk onboard the vehicle and uploaded to the platform when it is not in operation and has a good connection. Conversely, high priority events (eg: critical vehicle hardware malfunction) should not age out and should be the first events communicated up to the platform when connection is reestablished.
2. **EVENT COORDINATION** - To account for vehicle events being received out of order, at a delay, or not at all, a platform-level event coordinator is required. The coordination service can determine, with the help of data it has about the state of the world, whether events should be:
    - discarded; e.g. a position event from yesterday
    - remediated; i.e. an event indicating the vehicle is performing an action very different from what the platform expects it to be doing
    - deferred; i.e. an event that is normal but the platform handles it in conjunction with some other event data which may take a few seconds longer to arrive

3. **ALLOW VEHICLES TO MAKE DECISIONS** - Decide what logic can be decentralized (contained on the vehicle). Having software on the vehicle to handle common events it generates allows not only for a faster response, but also for the vehicle to be more self-sufficient. The vehicle will need to know how to operate when it runs out of commands from the platform - likely move to an area that is safe to park until it is given further instructions. A simple heuristic for determining whether a piece of logic can be decentralized is if the vehicle is sending some event up to the platform *only* to be informed how to respond to that very event. Decentralizing logic not only reduces data and latency, but also provides some peace of mind that the vehicles are operating in a safe manner when they do get disconnected from the platform.
4. **MONITOR VEHICLE CONNECTION HEALTH** - Knowing when a connection is unstable or even lost is valuable to logic contained by the transportation cloud, on the vehicle, and for operations of the service. If the protocol used for vehicle communication does not support this out of the box, consider building a bidirectional heart-beating mechanism such as keep-alive pings. When these disconnects occur, the data platform can trigger an event which notifies any interested parties (either software or human).

4. Streaming architecture characteristics are part of expected systems behavior

As described in the general purpose Nuvalence whitepaper The Anatomy of a Digital Platform, platforms are often built with emphasis on either workflow oriented workloads, or data oriented workloads. The nature of a transportation cloud requires significant architectural emphasis on both. Data generated in a mobility ecosystem is a combination of real time streaming data (vehicle telemetry, position data) and at-rest data elements, which provide the backbone for platform execution. When asynchronous workflows triggered by events are executing, they often depend on real time streaming data to execute their logic. This implies that thousands of simultaneous events will need performant access to these data streams. This introduces a number of key considerations, including:

a. Streaming vehicle position up to the data platform - architecting the platform to properly handle high streaming volume for a large fleet of vehicles, normalizing said position data, and efficiently distributing position data to dependent components in the platform
b. Allowing for a potentially large number of actors to tap into streaming data efficiently, which may in turn, require thoughtful partitioning strategies for inbound streams
c. Incorporate authorization so that only platform components that have the right to access certain streaming data actually do

Location data is particularly critical, especially when it comes to operational continuity and optimizing mobility efficiency. Often, the same data that is used on-platform to validate and act on requests can be used by consumer applications to improve user experience. Designing location APIs in a way that can be leveraged effectively both on and off platform allows consumers to build more robust solutions.

- **Routing & Arrival Times** - routing can be decentralized, and doing so reduces data volume and allows autonomous vehicles to adjust in response to their environment. Consumer applications that display routes and arrival times to their users will need access to routing APIs to ensure the information shown to end users is comparable to the route chosen by the vehicle. This guards against the consumer application telling a user their vehicle is 5m away while the vehicle determines it needs to take a long detour.

- **Mapping "Drivable" and "Non-Drivable" Areas** - the vehicle position or a location defined in the request to move a vehicle will often need to be cross referenced against contextual data that is geospatially bounded. For example, the bounds of a city where the service is operational or zones where vehicles cannot drive due to road closures.
- **Displaying Location Data to Users** - while platform components will communicate location data using exact coordinates, users will expect to consume that same information formatted as addresses. A centralized API for geocoding and reverse geocoding ensures that as independent user-facing applications convert coordinates to and from addresses the data remains consistent.

5. Governance, Regulatory & Compliance (GRC) requirements are yet to be defined, requiring a flexible architecture

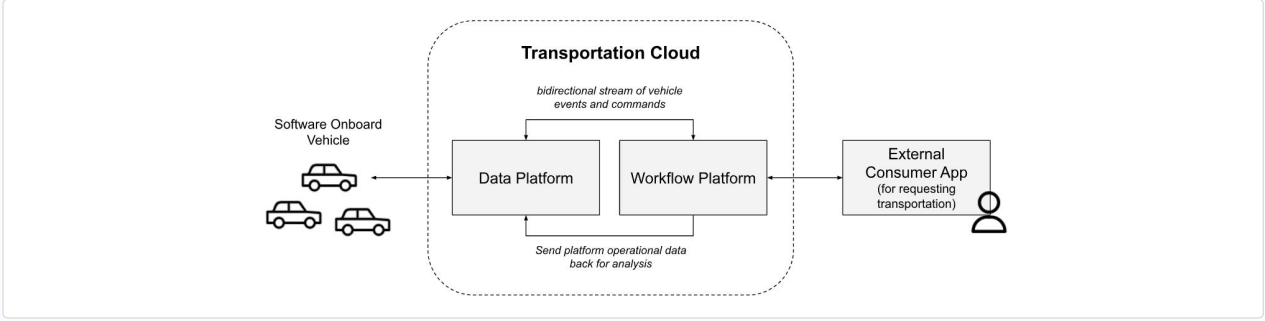
Given the introduction of autonomy and new modes of mobility, the GRC space will rapidly evolve. Currently, however, the industry is at a nascent stage, making concrete GRC expectations unlikely and impractical. There are two options from an architecture perspective:

1. Build a transportation cloud void of any GRC requirements and bolt them on later
2. Build a flexible GRC capability without any rigid expectations, and leverage that flexibility to comply to GRC requirements later

While (2) is an implementation detail of (1), it's important to call out separately. Many initiatives and architecture strategies would prefer to avoid "over-architecting" or not building something that isn't needed. While generally that's a prudent decision, given the timeline, a more intelligent option would be to build flexible auditing, data compliance, and tracing capabilities. That would allow the architecture to "pre-pay" some compliance costs early through abstraction. While likely over-architected for base cases, there are some known near-term legal and compliance considerations which come with all platforms (terms and conditions, data privacy, etc). In the case of autonomous vehicles, these requirements are actively being debated and are likely to be highly fragmented between countries, states and even municipalities. Incorporating a general purpose set of architecture building blocks for addressing those needs over time will reduce risk and overall long term compliance costs.

Bird's Eye View of a Transportation Cloud Architecture

The aforementioned considerations help shape the overall architecture of a transportation cloud. Before defining a detailed component architecture, a higher level visualization of the first consideration *(transportation clouds need to optimize for both data and workflow heavy workload expectations)* alongside vehicle and client-application actors is necessary.



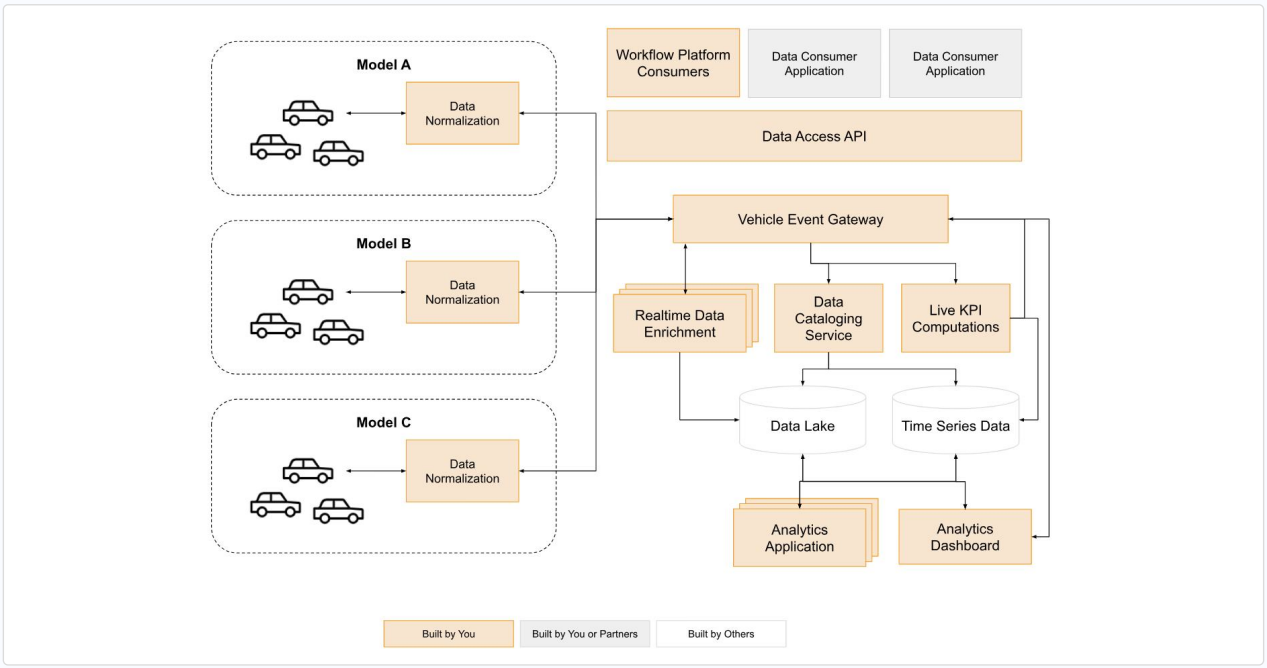
When this paper describes the “fusing” of data and workflow platforms, it implies an input/output loop where workflow transactions depend on data from the data platform, and then produces new data that feeds back into the data platform: a positive feedback loop. That feedback loop operates in a broader, coarse-grained ecosystem that includes software on the vehicle and client software that depends on the platform for its capabilities.

- **Software Onboard Vehicle** represents all decentralized logic. This will include routing, the driving system for autonomous vehicles, event-driven logic that was intentionally decentralized, implementation of the bidirectional streaming connection to the data platform.
- **Data Platform** holds not only the data retention and analysis features described above, it provides the interface for communication with vehicles. Features include: data cataloging, prioritization, buffering, and standardization/cleansing from many sources.
- **Workflow Platform** is the recipient of information streamed from the data platform and also communicates its own data back. It can act only on the data that is relevant without needing to consider analytics and retention use cases. Features include: vehicle management, transportation requests, notifications, location intelligence, billing.
- **External Client App** the direct consumer of the transportation cloud, creating a mobility-enabled solution out of the published “building blocks” and available data from the workflow platform and data platform, respectively.

Special consideration is required for the data platform.

Data Platform

Given the volume and complexity of data involved in a transportation cloud, a dedicated platform for data is recommended. The data platform needs to consider both real-time streaming and analytics as its top two use cases. In either case, a partitioning strategy is critical. A generally useful partitioning strategy is by vehicle model. Partitioning by vehicle model allows for an apportioned allocation of platform resources by popularity and alignment by vehicle features. The generalized data platform in this paper reflects this. In practice, partitioning (or sub-partitioning) on other dimensions might prove more practical.



This component diagram assumes a fairly mature platform which supports vehicle data providers in potentially many formats and/or protocols as well as external consumers of the data. The components in this architecture are as follows:

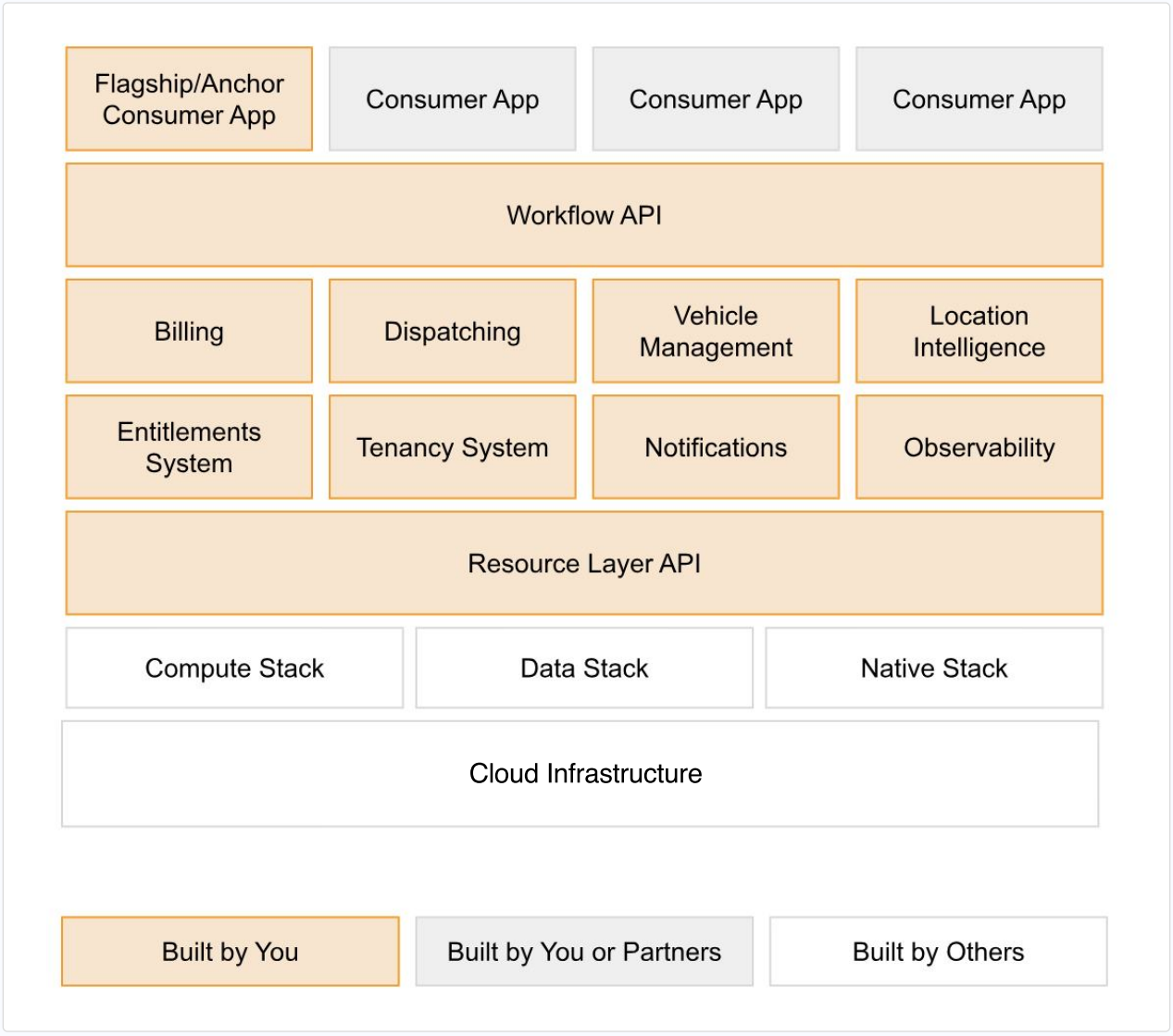
1. **Data Normalization** - as technology onboard the vehicle is improved, it is likely the case that different vehicle models will communicate similar data over different protocols or in different formats. This component will be responsible for normalizing the vehicle data to a platform standard representation and pushing it to the common Vehicle Event Gateway. As with consumer applications, initially this will be done internally for validation of the approach, but going forward partners could build their own integrations for data ingestion.
2. **Vehicle Event Gateway** - standardized stream of temporally consistent data.
3. **Real-Time Data Enrichment** - enriches streaming data with metadata stored on-platform to ensure contextually rich data is delivered to consumers while the data volume from vehicles themselves remains relatively small.
4. **Data Cataloging Service** - writes all data from the Vehicle Event Gateway to a platform data lake to be used by analytics applications and data access APIs later.
5. **Live KPI Computations** - performs computations of KPIs and metrics which must be done live (with low latency). These may in turn become new events on the vehicle event gateway for streaming consumers, be written to the data lake, or be written to a time series database optimized for rolling them up to aggregate KPIs.
6. **Analytics Applications** - perform computations and analysis of data in the data lake or time series database. Results of which could then also be cataloged.
7. **Analytics Dashboard** - user interface for live and historical data of interest from a business or operational perspective.
8. **Data Access API** - public facing interface for the data platform; provides methods for retrieving historical and time series data as well as bidirectional streaming of vehicle events.

Workflow Platform

All digital platforms are a combination of **'Core Services'** focused on supporting horizontal concerns (e.g. logging, tracing, state management) and **'Domain-Specific Services'** built to address business considerations in that given domain. In a transportation cloud:

1. CORE SERVICES provide a general execution environment with traceability, logging, observability, multi-tenancy, platform component management, scalable state and workflow management
2. DOMAIN-SPECIFIC SERVICES supporting the transportation domain with capabilities such as vehicle dispatching, location services, vehicle management, among others

This component diagram is inclusive of both the workflow domain-specific services enumerated above as well as some generalized platform architecture componentry that will help to accelerate the development of scalable domain-specific services.



1. **Data Stack** - independent of the data platform; while the workflow platform is not responsible for holding historical data for the business analytics use case, there will still be a need for services to store their own platform data using appropriate data stores provided by the underlying infrastructure.
2. **Billing** - for both invoicing consumers and allowing consumers to bill end users.
3. **Dispatching** - integrates with the data platform for dispatching events to vehicles.
4. **Vehicle Management** - managing the set of vehicles available for requests.
5. **Location Intelligence** - routing, geocoding, reverse geocoding, relational geospatial queries for location aware metadata.
6. **Entitlements System** - authorization rules for how components of the platform interact.
7. **Tenancy System** - provides onboarding for partners building consumer applications and hooks for provisioning isolated data or components.
8. **Notifications** - notification system for real time data streamed to consumer applications.
9. **Observability** - centralized logging, monitoring, and platform health provided to all services on-platform.
10. **Workflow API** - the publicly facing common experience user interface which ties domain services into a single user facing product.

Conclusion

A transportation cloud has the same requirements of any digital platform when it comes to building an ecosystem of solutions around a core offering. Transportation is quite complex: actors operating on tasks for indeterminate amounts of time with unpredictable behavior at extreme volume across the globe creates special consideration.

When considering building a transportation cloud, it can be hard to decide where to start. Nuvalence has spent considerable time and energy solving these problems in the mobility platform space. While this paper only scratches the surface, hopefully it provides seminal insights and a framework that will help in developing a broader transportation cloud strategy.

ABOUT NUVALENCE

Nuvalence is a next-generation consulting firm specializing in mission-critical, intelligent platforms for the world's most ambitious organizations.

Using our product-driven, AI-centric approach, we empower organizations to build for the intelligent digital future. Our elite team of product leaders, data scientists, designers, and software engineers enables our clients to solve their most complex technology product challenges and positively impact people and the world.

We don't just deliver software, we deliver outcomes.