Transporting People and Freight More Efficiently with Cloud-based Technologies

July 16, 2021

Notices

Contents

Abstract

This whitepaper explains how the creation of new transportation information systems or the refactoring of such existing systems to make use of cloud-native architectures, including serverless technology, provides improved elasticity, operational excellence, security, resilience, and reduced costs.

Introduction

Most people would agree that COVID-19 has had a dramatic impact on the transportation sector. Airlines, airports, and mass transit struggle with reduced ridership; municipalities struggle with reduced revenue from a reduction of vehicles on the road (reduced tolling and gasoline tax), and shop-at-home has become more popular, increasing freight on the roads. Freight transportation companies and logistics service providers are challenged by spikes in the e-commerce volume consuming unprecedented amounts of information technology (IT) resources. The need for real-time, dynamic re-adjustment of global supply chains shortened planning cycles from weeks to hours, requiring faster IT workloads, quicker access to data, increased ability to innovate and nimbler DevOps cycles. Existing information technology systems supporting transportation are evolving to meet these changes.

This whitepaper explains how the creation of new transportation information systems or the refactoring of such existing systems to make use of cloud-native architectures, including server-less technology, provides improved elasticity, operational excellence, security, resilience, and reduced costs for the transportation and logistics industry.

Industry trends

A massive shift in consumer habits, caused by COVID-19, commanded a surge in e-commerce volumes and home delivery. The U.S. DOT projections of truck vehicle miles traveled (VMT) are running at about twice the rate of growth versus personal VMT.

Whereas supply chains and freight/parcel logistics grew in terms of volume and revenue, other sectors of the transportation industry have suffered with a significant downturn. For instance, in the space of public transportation, work-from-home is the main contributor to the decrease in revenue:

- People are driving less, so toll revenues and gasoline tax revenues are down
- People are using less mass transit, so mass transit revenues are down
- People are using less air transportation

Also, the increasing adoption of electric vehicles is having a substantial economic and social impact. With fewer gasoline vehicles on the road, the revenue from the gasoline tax is trending down (the Federal tax is $0.183/gallon and the average State tax is $0.2568/gallon). Hence in the US some states are looking to move from a toll revenue model to a road usage charge (RUC), essentially a cost per driven mile, to recover lost gasoline tax revenue (see the RUC websites for the states of Utah and Oregon). There is also an interest in All-Electronic Tolling (AET) with the goal of reducing COVID-19 transmission and operational costs.

These trends have IT implications affecting both logistics and public transportation. First and foremost, a large, increasing, mostly untapped volume of data exists which can be leveraged by transportation IT systems. The volume of transportation data has exploded, boosted by autonomous and connected vehicles, and an increase in IoT devices such as roadside sensors, imaging and video systems, and even cell phones (e.g. GPS), all of which pave the way for more advanced information systems which incorporate artificial intelligence and machine learning (AI/ML) technology to improve road safety, convenience, and the overall driver experience.

Second, substantial IT adjustments to new revenue models are often required. For example, in the toll-road space, there is a significant infrastructure and process change when moving from traditional tolling to an RUC model.

Third, across global supply chains, the increasing movement of cross-border freight and parcel forces logistic service providers to cope with longer, higher peaks of IT resources utilization, requiring levels of elasticity and scalability difficult to be met by on-premises infrastructures.

Technology debt

Chronic under-investment into more advanced IT technologies has prevented the majority of transportation companies from keeping up with innovation. Most transportation information systems are monolithic infrastructure and application stacks with each customer having their own deployment running on their own hardware in their own data center. To date, very few transportation companies have embraced a journey to refactor applications into multi-tenant, serverless environments hosted in the cloud.

Once in place, these transportation information systems typically remain in operation for six to ten years, sometimes even longer. During the last 30 years, public and private organizations invested massive amounts of resources into developing complex, monolithic architectures supporting legacy systems such as traffic management systems (TMS), warehouse management systems (WMS), terminal operating systems (TOS), tolling systems, and inventory management systems. In-house development enabled these workloads to be highly customized to the customer needs. However, these types of systems come with a heavy burden of infrastructure administration and on-going application maintenance. Data centers are necessary to provide power and cooling. Servers, disks, power supplies, network gear, and other components must be monitored for failures and repaired. Hardware and software annual maintenance costs typically run high, especially in the later years of operation. Operating systems must be patched and kept current, primarily to fix security vulnerabilities. If redundant systems are a requirement (for resiliency or disaster recovery), then you can double the initial capital expenditures and the annual maintenance costs.

Additionally, the high burden of on-premises infrastructure administration slows down the pace of innovation and ties up valuable resources to perform undifferentiated tasks. Above all, at some point, the old IT systems become too difficult to maintain. For these systems, adding new features to improve business continuity, integrate with newer external systems, or provide an improved customer experience becomes difficult if not impossible.

Migration strategies

Multiple cloud migration strategies exist. The most common strategies fall into one of the “Six R’s”:

- **Rehost**: a.k.a, “Lift and Shift”, essentially move the entire stack “as-is” to run on cloud servers
- **Replatform**: move one or more components to a SaaS service (such as Amazon RDS for the database)
- **Repurchase**: move to a completely new application - essentially purchase or subscribe to a hosted service
- **Refactor/Re-architect:** build upon existing systems with the intent of moving to a serverless architecture
- **Retire**: remove infrastructure and software that is no longer required
- **Retain**: essentially, do nothing for now, and re-visit at a later date

Rehosting or Replatforming existing systems to the cloud is possible; however, the more advantageous approach (and the focus of this document) is to re-architect/refactor these systems to be more cloud-native. This approach means using existing cloud services on which to build the needed functionality while also providing the elasticity (the ability to dynamically scale resources up and down with demand), resiliency, and reduced systems administration, all of which leads to better performance and operational excellence at a reduced cost.

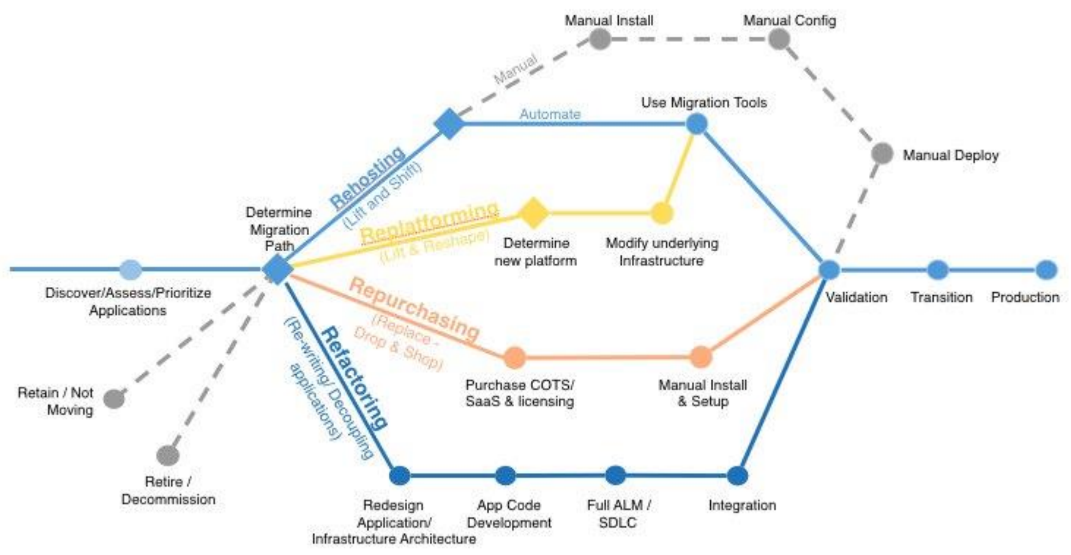

Figure 1: Cloud migration paths

Getting started with refactoring

Migrating an existing application to a serverless model does not need to be a complete re-write. In fact, a re-write approach is usually the wrong path to take for multiple reasons:

- **Re-writes take significantly longer than expected.** This is due to optimism (and internal corporate pressures) but also in part due to the lack of understanding of the system complexity.
- **Re-writes introduce new bugs.** The old code base has been in users' hands for a while and the bugs worked out over time. A new system has new bugs which extends internal testing. New bugs not found in testing make it through to end users, who then think the new system is less reliable.
- **Re-writes create new systems with reduced functionality.** The existing code base has evolved over time and includes certain business logic and error handling, some of which may be lost during a re-write, especially if previous developers are not available and/or documentation is scarce.

Instead of a complete re-architecture and code re-write, in most cases, a more evolutionary approach is recommended. However, before diving into this evolutionary approach, here are some considerations to keep in mind regarding cloud architectures and migrations:

Inventory

Document the existing servers, application components, and their dependencies. It is important to fully understand the system as it currently exists and especially understand the system's scope and boundaries with regards to its interactions with outside entities that act as sinks or sources of data and/or events (e.g., people, other databases or information systems).

Functional decomposition

Break down the existing system into small, functional components. Think in terms of microservices. Here are some aspects to keep in mind:

- **Security.** Implement security best practices including encryption for data both at-rest and in-transit. Do not embed any credentials directly in code, but rather use certificates, a secrets manager, or some other authorization technique.
- **Keep a loose coupling between the various components.** If a single function performs too many tasks, its re-usability is lessened. Implement asynchronous execution, where possible. Message queues and event handlers are great ways to implement asynchronicity and loose coupling.
- Error handling. Build an architecture, mechanisms and user interfaces that prevent errors following the "**poka-yoke**" inadvertent error prevention methodology. When they occur, build structured workflows to handle errors quickly and providing the best user experience
- **Testing.** Think about how this system will be tested throughout the development process and also when it moves into production.
- **Leverage available managed cloud services where possible.** Security, elasticity, and resiliency are usually built into these services, which means less design effort now and less administration effort when the system goes into production.

Prioritization

There are probably several existing services from which to choose. Think about which services, if refactored, will provide the most value at the lowest risk. If this is a first time refactoring exercise, select a service which has a high probability of success. Ask yourself these questions:

- Which services will have a minimal impact on business continuity?
- Which services have low complexity?
- Which services have low coupling (connectivity) to other parts of the system?

Security and Operational Excellence

It is best to think of these considerations at the onset of the project as opposed to an afterthought:

- Which parts of the system require public exposure (e.g., a public web server) and need additional security?
- What will the day-to-day administration look like and what are the various administrative roles necessary? Common roles include Database Administrator, Infrastructure Administrator (which includes server, server-less, networking, local storage).
- Has 'least privilege' been implemented for each of the various users and roles of the system?
- Does the system store any personally identifiable information (PII)? What are the security requirements around this data? Does the PII data need to be anonymized?
- Must the system adhere to one or more of the published security compliance programs (IRS 1075, PCI DSS, FedRAMP, CJIS, ISO 27001, et al.)?
- What are the business continuity and disaster recovery steps? Are any of the components that were identified in Functional decomposition a single point of failure?

Evolution of a serverless migration

The following diagram shows a high level view of a typical application stack. The compute infrastructure on which the application runs may be a single server or multiple servers. This monolithic architecture is a single unit of deployment, whether you are deploying for multiple customers (tenants), deploying as multiple units of scale, or deploying as units of fault tolerance.

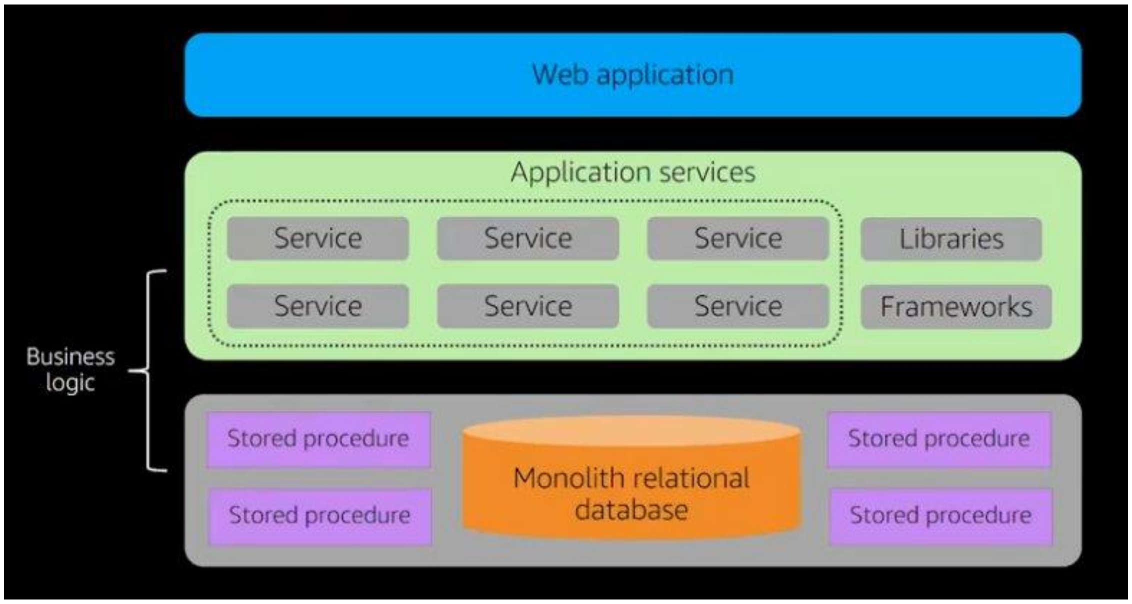

Figure 2: Typical application stack

Step 1. Rehost and add multi-tenancy

In this step, the entire application stack is moved “as-is” to the AWS Cloud. Minimal changes are made as required to host the application stack in the AWS Cloud. In a multi-tenant environment, each tenant receives their own copy of the entire deployment, as shown in the following diagram.

Additionally, common, multi-tenant, ancillary services have been created around the customers’ stack deployments to facilitate the multi-tenant environment. This added functionality includes: Registration, On-boarding, Monitoring, Logging and Metrics Collection, Metering and Billing, and automated Deployment.

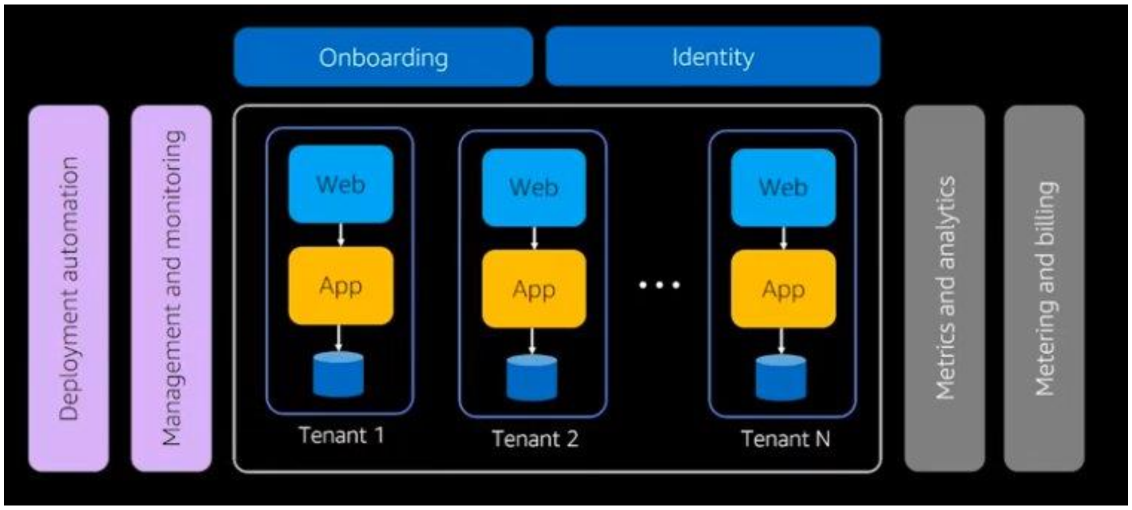

Figure 3: Rehosted application stack with multi-tenancy

Step 2. Tenant routing

Using [Amazon Route 53](), domain names are created for each tenant which directs them to their own application stack deployment running in the cloud.

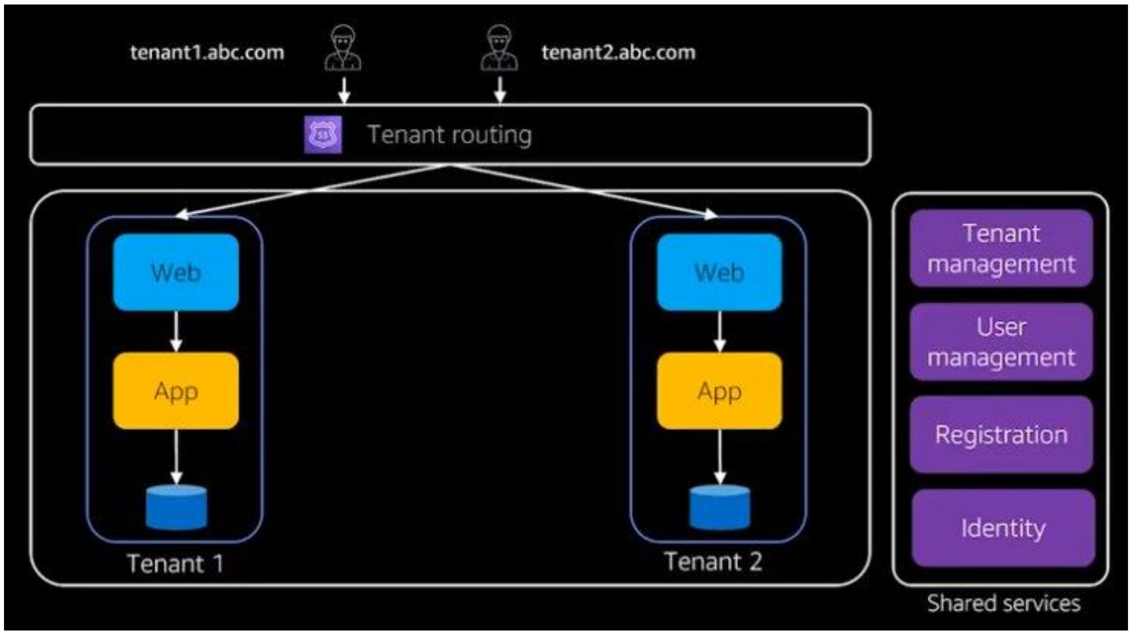

Figure 4: Application stack with tenant routing

Step 3. Refactor for microservices

Next, from the App tier, factor out a common service that can be implemented as a microservice. Common AWS infrastructure services for implementing microservices are AWS Lambda or a container technology such as Amazon ECS, Amazon EKS, or AWS Fargate). The use of these managed services reduces the undifferentiated tasks of maintaining servers (physical or virtual) including operating systems and hardware. Be sure and review the refactoring principles mentioned previously to select proper service candidates for refactoring.

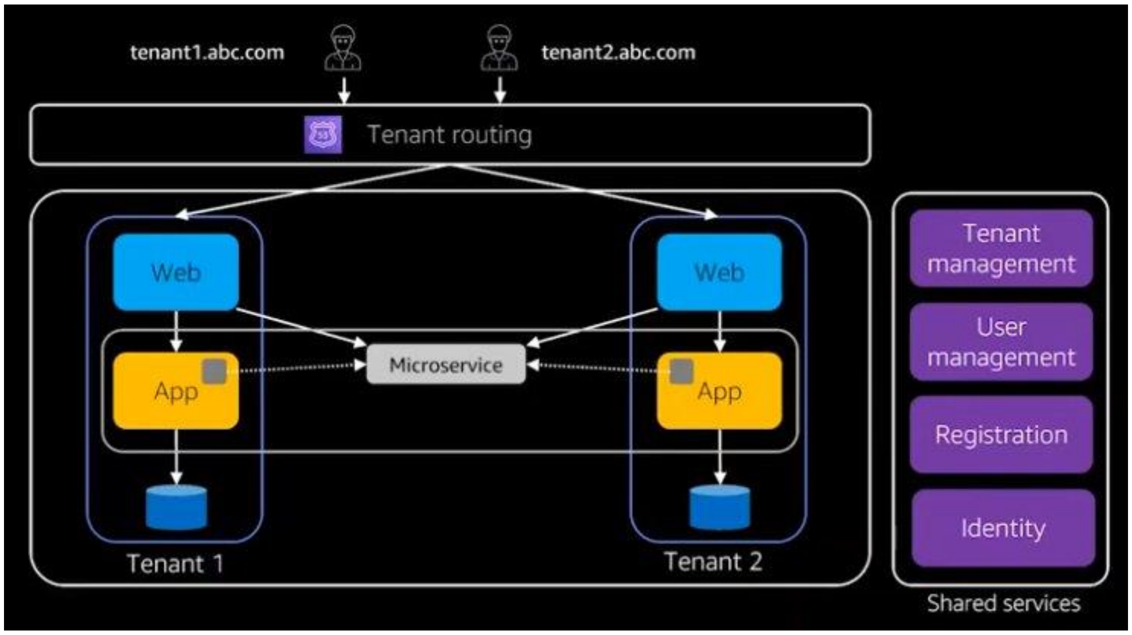

Figure 5: Application stack with addition of microservices

Over time, more and more App tier services can be factored into microservices.

Step 4. Refactor the Web Tier

Eventually the Web tier might also be factored out into a common set of web pages which make use of Amazon API Gateway to invoke the various services and microservices. Although beyond the scope of this document, once the Amazon API Gateway is in place, you can also implement canary deployments as new microservices are developed, to facilitate testing of these new microservices.

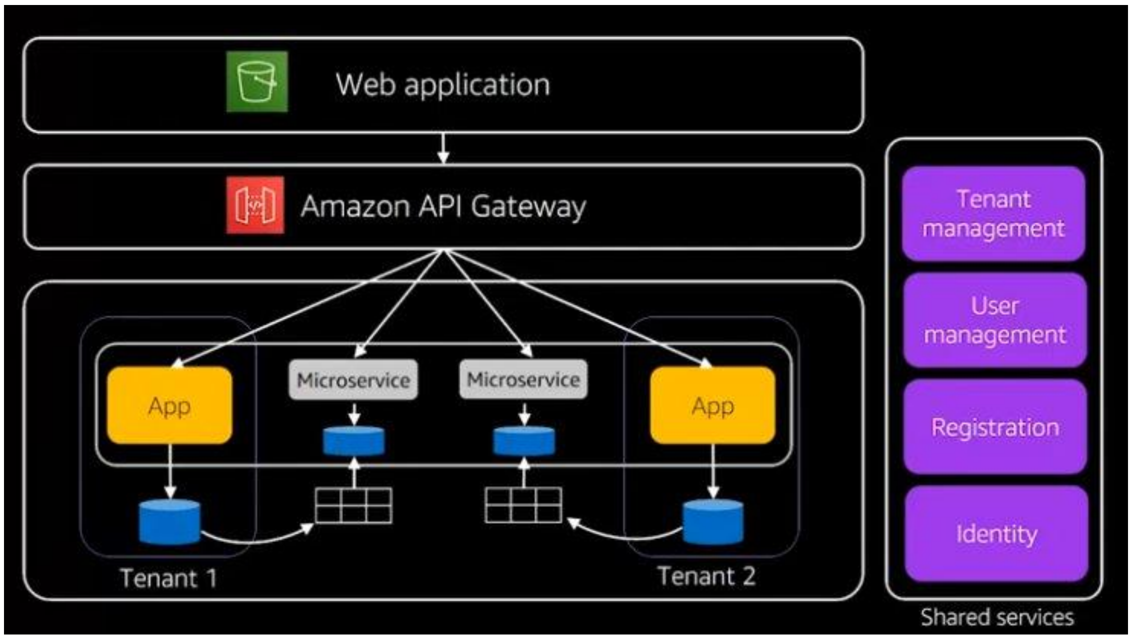

Figure 6: Caption

Conclusion

Cloud-based, serverless architectures are an ideal platform for modernizing transportation information systems. The transition from an on-premises, monolithic architecture does not have to be a complete re-architecture and re-write of the application; the transition can evolve over time with little to no impact to your end users.

Contributors

Contributors to this document include:

- Ashley Miller, Senior AI/ML Evangelist, AWS
- Michele Sancricca, WW Head of Technology – Supply Chain & Logistics, AWS

Further reading

To learn more about some of the organizations which have undertaken a cloud-migration journey with AWS and implemented transportation solutions on the AWS Cloud, see customer stories for Matson, Seaco, Deutsche Bahn, First, Elizabeth River Crossings, Transport for NSW, and the City of Detroit.

External references

Surface Transportation News: Infrastructure Stimulus Bill, Highway Investment and COVID-19

Document history

Date	Description
July 16, 2021	First publication