Phase 1 Enabling Technology Readiness Assessment

University of Washington ITS4US Deployment Project

www.its.dot.gov/index.htm

Final Report — May 8, 2023
FHWA-JPO-21-889

COMPLETE TRIP ITS4US

U.S. Department of Transportation

Produced by *University of Washington ITS4US Deployment Project*
U.S. Department of Transportation
Intelligent Transportation Systems Joint Program Office
Federal Highway Administration
Office of the Assistant Secretary for Research and Technology
Federal Transit Administration

Notice

Technical Report Documentation Page

1. Report No. FHWA-JPO-21-889	**2. Government Accession No.**	**3. Recipient's Catalog No.**
4. Title and Subtitle Phase 1 Enabling Technology Readiness Assessment—University of Washington ITS4US Deployment Project		**5. Report Date** May 8, 2023
		6. Performing Organization Code N/A
7. Author(s) Anat Caspi, Director of the Taskar Center for Accessible Technology at University of Washington, Mark Hallenbeck, Director of the Washington State Transportation Center at University of Washington;		**8. Performing Organization Report No.**
9. Performing Organization Name and Address University of Washington 4333 Brooklyn Ave NE Box 359472 Seattle, WA 98195-9472		**10. Work Unit No. (TRAIS)**
		11. Contract or Grant No. 693JJ321C000004
12. Sponsoring Agency Name and Address U.S. Department of Transportation ITS Joint Program Office 1200 New Jersey Avenue, SE Washington, DC 20590		**13. Type of Report and Period Covered** N/A
		14. Sponsoring Agency Code HOIT-1

15. Supplementary Notes

Kate Hartman, COR

16. Abstract

This report is the initial draft of the Enabling Technology Readiness Assessment for the Transportation Data Equity Initiative, an effort funded by the Federal Highway Administration's ITS4US Program. The project, led by the University of Washington's (UW) Taskar Center for Accessible Technology and the Washington State Transportation Center, will develop a national pipeline to create, disseminate, and share standardized data about pedestrian environments, transportation environments, and on-demand transportation services to enable better use, discoverability, and data analytics of these assets and services. Specifically, the project will release nationally the OpenSidewalks data standard for digitizing pedestrian ways and will extend the national data standards for on-demand transit services (GTFS-Flex) and for mapping of multilevel transit stations (GTFS-Pathways).

The ETRA describes the underlying technology needs associated with the creation of such an interoperable data environment as planned by the deployment of the TDEI. The report assesses the technology readiness of the various supporting technologies that will be integrated into the TDEI. It presents the technology assessment as suggested by the Technology Readiness Level Guidebook, FHWA publication FHWA-HRT-17-047. The report is intended to help inform end users, developers, agencies, organizations, and staff involved in the system of the planned build-out of the system and responsiveness of the technology to the user needs and subsequent system requirements.

17. Keywords ITS4US; Complete Trip; Deployment; ITS; Intelligent Transportation Systems; Safety Management, Enabling Technology; Accessibility; Sidewalks; Navigation software; Data Standards	**18. Distribution Statement** N/A		
19. Security Classif. (of this report) N/A	**20. Security Classif. (of this page)** N/A	**21. No. of Pages** 98	**22. Price** N/A

Form DOT F 1700.7 (8-72) **Reproduction of completed page authorized**

Revision History

Name	Date	Version	Summary of Changes	Approver
Anat Caspi	7 December 2021	1.0	Initial Draft	
Anat Caspi	2 February 2022	2.0	Final	
Anat Caspi, Mark Hallenbeck, University of Washington	8 May 2023	3.0	Revised Final	

Table of Contents

List of Tables

List of Figures

1 Introduction

The last two decades have been a time of unprecedented change in how people move through built spaces. The rise of open, shared transit information, real time data streams, massive digital cartography platforms, and other transportation data technologies have changed the way we think about the built environment, transport, and mobility data. The components of these new mobility data environments, the focus of this document, represent an exciting future. Most acutely, new, and disruptive transport will result in profound changes in transportation data. There are implications for jobs, accessibility, sustainability, resilience, social equity, and the environment. There are opportunities to shape advances in transportation data infrastructure to improve streets and better connect people; to reshape cities and improve the social and physical health of communities. There are opportunities to reduce pedestrian collisions and improve access to community settings and social services for those who need it most. There is also the potential to connect people to jobs and change the way cities organize space, prioritize resource allocation, and optimize trips. Yet these opportunities also present challenges. The ones addressed in this document are data infrastructure design challenges that are presented by this opportunity to invent the information exchange architecture for this new mobility.

Smarter transportation may not always translate into greater sustainability, access, or equity. There is a risk that leaders from the public and private sector will select data infrastructures that may not have the interoperability, scale, and extensibility necessary to address the full diversity of travel needs represented by the population they serve, or to be responsive to ever changing environments they monitor. The intent of the Transportation Data Equity Initiative (TDEI) is to collaboratively work with diverse stakeholders to achieve the full benefits of new technology, by applying strong design thinking to the creation of this data infrastructure.

A primary objective of the -ITS4US Deployment program is to deploy new and innovative mobility solutions to help underserved populations perform a complete trip. Mobility innovation is a space where the design and policy decisions that planners, engineers, and policymakers make now will definitively frame the future. In the case of the TDEI, we are concerned about the data design and data policy that will machinate this future. This document is meant to identify and assess the maturation of the enabling technologies used to create an integrated solution in the deployment of an open, shared, interoperable data infrastructure. One of the most important factors in our decision processes will be partnership, and we hope that through our ongoing work with the TDEI project partners and collaborators, we will achieve promising technology outcomes and useful data infrastructures.

1.1 Intended Audience

As discussed, huge modal shifts are now being realized by some travelers mainly through new data-driven technologies that are offering some populations a high level of flexibility and convenience in multimodal travel. The TDEI project arose from the recognition that these changes in personal mobility markets have shifted the roles of public authorities as data stewards. Public authorities are moving beyond their conventional role as infrastructure providers by enabling and promoting alternative mobility services and the data infrastructure to support the integrated, seamless use of all modalities. Public authorities and public transport companies are increasingly using the services of new data enterprises and new data platforms to reduce the need for costly investments in new transport infrastructure, equipment, and operation systems, but such adoption should be cautious and deliberate lest it leave some transportation consumers behind. To say “subpopulations are being left behind by data infrastructure” means that some populations may remain unseen (or unrepresented) by the data systems, or that data attributes that some populations are concerned about may be unreported or underreported through those data systems, or that certain types of data or analyses those specific subpopulations care about remain completely unsupported by the new mobility applications that consume these publicly available data.

The intended audience for this document includes the data providers, data publishers, data policy makers, data infrastructure decision makers, application developers and the travelers who require seamless access to discover, use and evaluate travel options. While not all the technology components discussed herein would be of equal interest to all the named stakeholder groups, the text is intended to be understood by all.

1.2 Project Background

In late 2019, the United States Department of Transportation (U.S. DOT) launched a new department-wide initiative. This initiative, referred to as the Complete Trip initiative, aimed to expand access to transportation for people with disabilities, older adults, and individuals of low income. This initiative recognized that all Americans need access to high-quality, affordable, safe, frequent, and accessible transportation options to access employment opportunities, educational opportunities, healthcare services, and other activities, but that some groups do not receive the same quality of service. To support these underrepresented groups, U.S. DOT aimed to increase its investments in innovations that enhance access and mobility for all travelers, including, but not limited to, the following user groups: people with disabilities, older adults, low-income earners, rural residents, veterans, and those with limited English proficiency (LEP) (henceforth referred to as “underserved travelers”).

In support of this initiative, the Federal Highway Administration created a Broad Area Announcement opportunity, (BAA) #693JJ3-20-BAA-0004, “Complete Trip - ITS4US Deployment.” The University of Washington (UW) submitted one of five projects selected for funding under this BAA, “Complete Trips Empowered by Data Standards: Accessible Mapping Standards and Data Collaboration Drive Accessible Multimodal Mobility” (referred to as the “Transportation Data Equity Initiative, TDEI, or the UW ITS4US Project”). This deployment is one of the Phase 1 Complete Trip – ITS4US Deployment Program projects selected to showcase innovative business partnerships, technologies, and practices that promote independent mobility for all travelers regardless of location, income, or disability.

The UW ITS4US Deployment Project aims to create the foundational data tools necessary for both public and private entities to collect, share, manage, and use transportation data that provide equitable outcomes to all travelers. At its core, the project is about creating the foundational requirements for interoperable transportation data sharing that fulfills the informational needs of all travelers, allowing them to discover and use diverse travel options that meet their specific needs. The UW ITS4US project itself consists of multiple parts.

First, it includes work with three existing standards committees to extend and update existing, early-stage international data standards: OpenSidewalks, GTFS-Flex, and GTFS-Pathways. These three data standards enable the consistent collection and reporting of data that provide the underlying information needed by the currently underserved target populations— people with disabilities, older adults, and individuals with low income—to efficiently travel.

Second, it is developing a series of tools that help agencies, jurisdictions, and other stakeholders collect the data that can be stored with these refined data standards. These tools are needed to lower the cost and improve the quality and consistency of those data collection efforts to increase the availability of the data.

Third, it is developing tools, policies, and procedures that allow sharing and governance of the collected data. The tasks performed will enable effective and efficient vetting, aggregation, management, and fusion of the data that participating agencies, jurisdictions, and other stakeholders collect. This portion of the project will also include tasks required to enable and manage the sharing of those data with application developers that write software to deliver requested travel information.

Fourth, it is developing a data repository to contain the data to be shared within the six counties that represent the geographic boundaries for this ITS4US project. The data repository will be developed to illustrate how these data can be collected, stored, governed, updated, and maintained over time and then served upon request to application developers.

Finally, the project is developing three example applications that use the collected data. The three applications are intended to demonstrate three very different uses of the data that are made available to application developers because of the other four aspects of this project. Those data can be used to fulfill a variety of information needs, and those needs can be met through an almost infinite number of applications. The three applications deployed as part of this project are meant to show other application developers how the newly available data can be obtained and delivered.

1.3 Scope

The aim of this document is to set the context of the future technology innovation work performed by the Transportation Data Equity Initiative, the UW deployment project for ITS4US. The document first lays out the necessary knowledge and our working assumptions about the technology requirements and discusses the manifestations of these requirements in enabling technologies. The document relies on the needs analysis of the core stakeholder communities, reviewed in detail in the Concept of Operations (ConOps)[1] document, and subsequently expressed as system requirements in the System Requirements Specification (SyRS)[2] , to offer a

[1] UW ITS4US Concept of Operations. FHWA-JPO-21-861 https://rosap.ntl.bts.gov/view/dot/58675

[2] UW ITS4US System Requirements Specification. FHWA-JPO-21-884 https://rosap.ntl.bts.gov/view/dot/60129

top-level view of the possible technologies that will be used in the implementation of the work packages of this deployment project.

From the needs analysis, we identify the specific technology areas emanating from the service requirements identified in the SyRS document. Note that this document does not aim to provide a complete analysis of the needs or requirements. Also, it does not provide a technology solution of a service specification. These latter aspects will be addressed by other deliverables.

1.4 Goals and Objectives

Our goal is to define, design and prototype data services to support the data lifecycle for data types that are currently missing but are needed to support travelers with disabilities find the information they need. Our initial approach has been to identify the gaps of data services to support the UW ITS4US project for different communities and define a first set of requirements from this gap analysis. In addition to creating equitable data schemas and supporting the data lifecycle for these data, the approach must avoid being biased toward existing communities or towards specifically existing services. There is a need for extensible, scalable, interoperable data approaches that will address the various stakeholder communities' needs as new modes of travel and new data types are added to the marketplace.

We approach this problem from a generic standpoint, making as few assumptions as we could to satisfy the needs of most individual transit agencies, civic organizations, and individual citizens eager to make use of the data and subsequent data services. The ConOps and SyRS documents offer first steps in the analysis, defining the generic data lifecycle requirements for each of the communities with the support of existing defined use-cases. In this document we break down these activities further to reformulate their descriptions to express the requirements in terms of the way in which they require technology development or extensions of existing technology products or services. The result of this work is described in the next sections and evaluated for technological readiness. Documenting the maturity of the technologies will enable others to build upon the investments made in this project to progress toward Complete Trip goals more effectively in future deployments.

2 Identify Enabling Technologies

2.1 Technology Readiness Framework

Our technology readiness assessment will be based on the guidance provided by the following document:

- FHWA Technology Readiness Level Guidebook
  https://www.fhwa.dot.gov/publications/research/ear/17047/17047.pdf

Specifically, we will use the following descriptions and requirements to assess the technology readiness at each of 5 levels, as described by the referenced document:

Table 1 Descriptions and requirements of Technology Readiness Levels (TRLs) as depicted in tables 2 and 3 of the FHWA Technology Readiness Level Guidebook

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements.
Tech Readiness Level 1 **Basic principles and research** • Do basic scientific principles support the concept? • Has the technology development methodology or approach been developed?
Tech Readiness Level 2 **Application formulated** • Are potential system applications identified? • Are system components and the user interface at least partly described? • Do preliminary analyses or experiments confirm that the application might meet the user need?
Tech Readiness Level 3 Proof **of concept** • Are system performance metrics established? • Is system feasibility fully established? • Do experiments or modeling and simulation validate performance predictions of system capability? • Does the technology address a need or introduce an innovation in the field of transportation?
Tech Readiness Level 4 **Components validated in laboratory environment** • Are end-user requirements documented? • Does a plausible draft integration plan exist, and is component compatibility demonstrated? • Were individual components successfully tested in a laboratory environment (a fully controlled test environment where a limited number of critical functions are tested)?

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements.

Tech Readiness Level 5 **Integrated components demonstrated in a laboratory environment**

- **Are external and internal system interfaces documented?**
- **Are target and minimum operational requirements developed?**
- **Is component integration demonstrated in a laboratoryenvironment (i.e., fully controlled setting)?**

2.2 Enabling Technologies Inventory

At its core, the Transportation Data Equity Initiative aims to build and support the infrastructure for securing and sharing accessible transportation data required for all people with a full range of mobility needs and preferences to make informed decisions about their travel. When organizations produce data about the travel environments they control, or about the on demand services they offer, a series of tools will be available for them to share and maintain the data in such a way that downstream application developers will be able to build software to assist travelers in transit, inform them about real-time and static information on the ground, perform personalized route optimization, and manage critical data to make trip planning and rerouting for the complete trip a seamless experience.

Given the wide range of types of data that will be integrated in one platform, the anticipated system is more complex than a typical geographical information system (GIS). However, the various components of GIS will certainly have to factor into the design of this system. Specifically, the data schema design, workflow, and organization, how the data moves through the system and how other entities could consume and analyze it will have to be methodically thought out. In this document, we consider the computing hardware that will support the software produced by our team, as well as the software products themselves. The artifacts that mobile device end users will interact with will tie back to architectural workflow and organization, as well as the specifications of the data and the software.

In this section, we detail the different Data Lifecycle Activities that will be required to completely satisfy the stakeholder requirements in the services of the TDEI. The Data Lifecycle Processes that must be addressed by the TDEI include:

a. DATA COLLECTION- Represents the point at which new and/or existing data are collected or generated.

   i. Different methods for entry

   ii. Validation/certification tools

b. DATA PROCESSING- Represents the activities associated with the necessary preparation of various new or existing acquired data inputs.

   i. Enrichment

ii. Inference

iii. Analytics

iv. Synonymous Conflation

v. Non-synonymous Conflation

vi. Data Upstream

c. DATA QUALITY CONTROL- Represents the activities to measure and monitor data quality to ensure that the data are usable at any stage of the data life cycle.

d. DATA VALIDATION—Represents the activities to assess data upon entrance, offering a certificate before the data enters the stream, ensuring the data are ingestable.

e. DATA DESCRIPTION- Represents the activities of identifying (e.g., digital object identifier (DOI)) and documenting the data with extended metadata (that could be defined with common semantics) to allow for understanding, harvesting, and consuming the data itself.

f. DATA UPDATE Represents the activities that directly change the content of data (reformat, add in updated sidewalk information, etc.)

g. DATA EXTENSIBILITY Represents the activities of identifying and documenting new data attributes or data types to allow for data schema extensions and dynamic growth.

h. DATA SHARING/DATA PUBLISHING Represents the activities associated with making community data stores available through web sites, web services, data catalogues, and so on

i. DATA DISCOVERY Represents the activities involved in finding data based on metadata and/or provenance information.

j. DATA ANALYSIS Represents the activities associated with the exploration and interpretation of well-managed, processed data for the purpose of knowledge discovery.

k. DATA PROVENANCE Represents the activities of documenting the various operations that occurred on data (data processing, data analysis, data transfer) to achieve reproducibility and referencing.

l. DATA PERFORMANCE- Represents the activities that involve data access optimization, as well as other performance issues. To be useful, the data infrastructure must support incoming and outbound data streams with good performance. There are multiple application areas (namely data coming from streams, from sensors and crowdsourcing) and appropriate infrastructure support is vital to handle these challenges.

m. DATA BACKUP Represents the activities that involve the management of physical risks to the data throughout the data lifecycle. Routine local backups are critical to prevent the physical loss of data prior to the final PRESERVATION of the data.

n. DATA STORAGE and LONG-TERM PRESERVATION Represents the activities associated with preserving data for long-term use, re-use, and accessibility.

Three main system components will be highlighted as we consider three core enabling technologies that cover the most critical aspects of the TDEI deployment and the Data Lifecycle Activities. The core technologies TDEI needs to support include the provisioning of the following:

1. **Microservices architecture for data collection, aggregation, transformations, and other lifecycle activities** will enable the actions of joining data from multiple sources, in multiple formats, and performing data aggregations, transformations and integrations to bring input data into the consistent data schemas promoted by the TDEI.
2. **A data architecture enabled by event streams** allows interoperable data sharing to occur based on triggers from events.
3. **TDEI APIs** will enable data usage and consumption that accommodate different use cases.

As shown in Figure 1, TDEI technology subsystem services express functional divisions of engineered components in the TDEI data sharing system. The items in the leftmost panel (shown in green) are not part of the system and are called “inputs” to articulate exemplars of the different types of input data that might be the inputs to the data sharing infrastructure. First, some agencies may contribute data in different formats, some may be GIS data, transportation data, or other data streams. Second, some data providers may share large data batches that are not transportation data but imagery data. Third, some data producers may already be furnishing APIs for downstream consumption that may be integrated into transportation data.

The main activity of provisioning interoperability by the TDEI is found in the panel numbered 2.2.2 “Data Interoperability platform.” However, as the TDEI anticipates some data providers may never directly adopt the data standards, and others may produce partial streams, this deployment project intends to offer some data connector and adapter microservices, depicted in the second panel from the left (2.2.1 Microservices provide adapters or connectors to the TDEI platform).

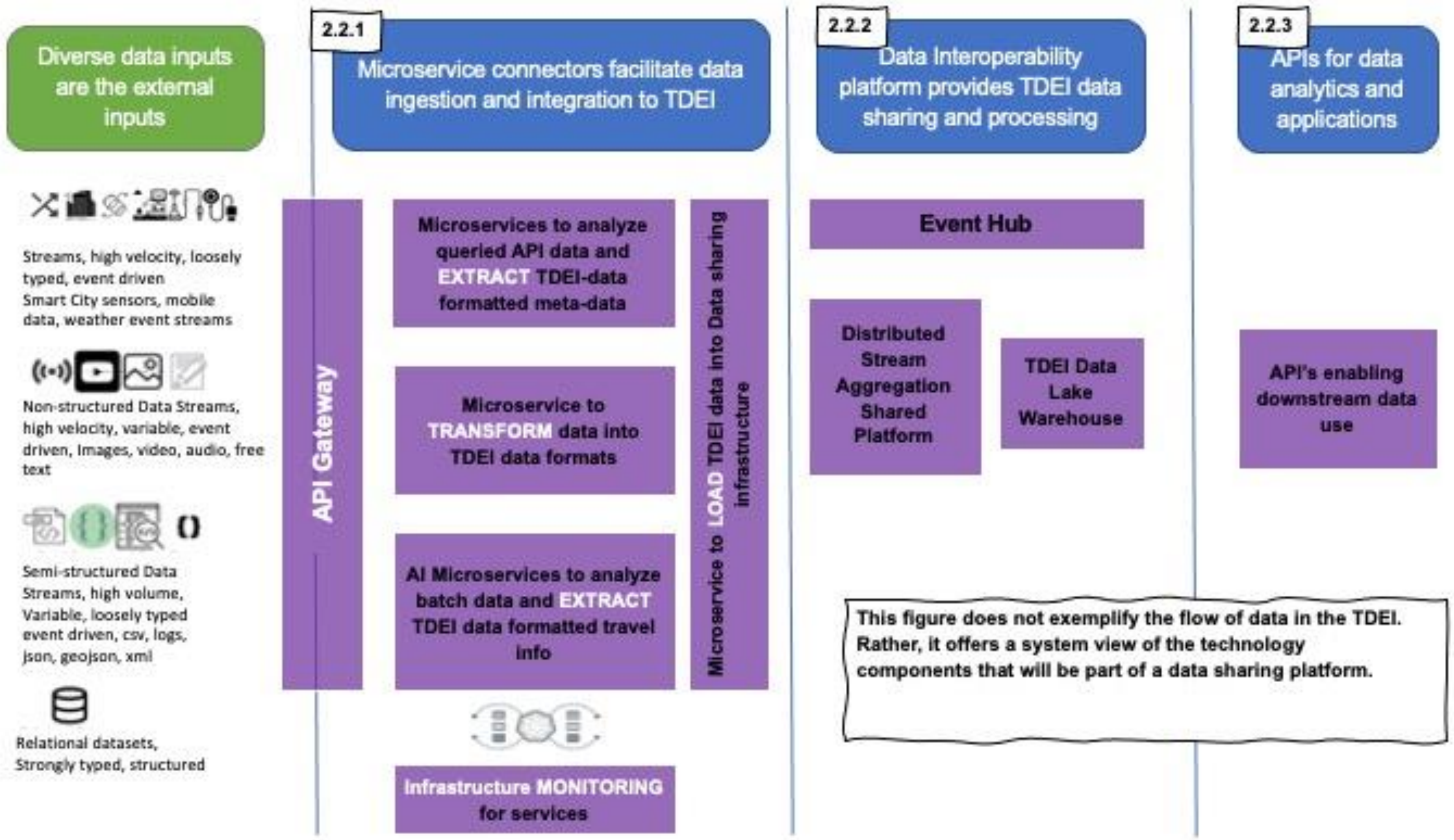

Diverse data inputs are the external inputs
Streams, high velocity, loosely typed, event driven
Smart City sensors, mobile data, weather event streams
Non-structured Data Streams, high velocity, variable, event driven, Images, video, audio, free text
Semi-structured Data Streams, high volume, Variable, loosely typed event driven, csv, logs, json, geojson, xml
Relational datasets, Strongly typed, structured
2.2.1
Microservice connectors facilitate data ingestion and integration to TDEI
API Gateway
Microservices to analyze queried API data and EXTRACT TDEI-data formatted meta-data
Microservice to TRANSFORM data into TDEI data formats
AI Microservices to analyze batch data and EXTRACT TDEI data formatted travel info
Microservice to LOAD TDEI data into Data sharing infrastructure
Infrastructure MONITORING for services
2.2.2
Data Interoperability platform provides TDEI data sharing and processing
Event Hub
Distributed Stream Aggregation Shared Platform
TDEI Data Lake Warehouse
2.2.3
APIs for data analytics and applications
API's enabling downstream data use
This figure does not exemplify the flow of data in the TDEI. Rather, it offers a system view of the technology components that will be part of a data sharing platform.

Figure 1 TDEI technology subsystem services express functional divisions of engineered components in the TDEI data sharing system. The items in the leftmost panel (shown in green) are not part of the system and are called "inputs" to articulate exemplar data sources to the system.

Lastly, the TDEI platform will furnish (as shown in the rightmost panel numbered 2.2.3 in Figure 1) TDEI APIs for data analytics and applications, as well as data processing.

In the following subsections, we first explain the three core enabling technology components: microservice architecture, event buses for asynchronous communication, and APIs/API layers. We will cover each of these subsystem functional divisions below, paying attention to the core requirements derived in the SyRS, and assessing several options for implementation.

For discussion of a typical exemplar of our system, Section 2.3.2 describes the overall integration of these technologies into the TDEI infrastructure. Additional sections describe some of the microservices that will be deployed as part of the initial TDEI prototype technology stack. These will include microservices that are "off the shelf" and almost definitional for this type of interoperable data sharing. Two such examples are microservices for Messaging, and Identity. Next, we provide some examples of the microservices we will build out on our own, using examples from the TDEI Stakeholder needs that may make some of the specific microservices and tools we build unique and not 'off the shelf'. We also highlight some areas where it will be ideal for the TDEI to offload critical parts of microservice applications, which are not part of our core competency or expertise, externally to a cloud-based microservice. Using out-of-the box solutions will allow us to quickly implement key functionality. Some of the components that range from complex to almost impossible to build in-house have various Software as a Service solutions.

2.2.1 Microservices Architecture: Enabling Data Collection, Aggregation, Integration and Transformation

The lack of Interoperable transportation data for pedestrian spaces, travel environments and on-demand travel services present a major challenge in achieving ubiquitous support for complete trip planning. The plethora of diverse GIS information held in silos by municipalities and transit agencies, is widening the gap of interoperability. While many organizations are looking for a standardized solution, conflicting messaging and various systems that are being built concurrently makes it difficult to expect data producers to immediately adopt the TDEI's selected data schemas and provide data in those schemas. This deployment project recognizes a need for an alternate strategy that will allow data producers to seamlessly contribute data in several prevalent formats. Our solution is to provide several microservices which can intelligently mediate amongst a few mainstream GIS systems.

A microservices architecture[3,4] is an architectural paradigm in which a software system is built of small loosely coupled components as opposed to a single large monolithic system, called

[3] What is a Microservices Architecture? Google. https://cloud.google.com/learn/what-is-microservices-architecture

[4] Microservice Architecture Style. Microsoft. https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

monolithic architecture[5]. In a microservices architecture, a number of separate microservices are developed – each microservice has its own code base and potentially its own development team. The microservices can be managed and deployed independently improving the updatability of the system by decentralizing software updates – that is, a microservice can be developed and updated independently of other microservices. The microservices communicate over well-defined interfaces such as Application Programming Interfaces (APIs) or queue-based paradigms such as event busses. Each microservice is designed to implement a specific piece of business logic and in fact, microservices can be implemented in different programming languages and use different technologies. Thus, microservices each have specific business purposes, function independently, are developed independently, and communicate using well-defined interfaces. This is in contrast to traditional monolithic architectures in which the entire system is one code base, written in a single language, typically lack well-defined interfaces between components, and must be updated as a whole. The independence of using microservices can improve the agility of development and management of a software system.

Management and orchestration and API gateways[6] are key components of microservice architectures. Microservice architectures are also most commonly deployed on the cloud. In the TDEI infrastructure, APIs and API Gateways will take a central role in the maintenance and sustainability of data interoperability and sharing. Therefore, these core enabling technologies (API and API Gateway) as a topic will receive separate attention in Section 2.2.3.

- In general, use of microservices traces back to the following system requirements:

Table 2 Microservices, general traceability

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.1.3	Data Collection	F-CO-01	The TDEI system shall be capable of receiving data from multiple sources.	Demonstration	UN-DG2, UN-TS1, UN-DS8,
3.1.3	Data Collection	F-CO-01.01	The TDEI system shall be capable of receiving sidewalk data from data generators.	Demonstration	UN-DG2,
3.1.3	Data Collection	F-CO-01.02	The TDEI system shall be capable of receiving external third-party data for relevant secondary attributes.	Demonstration	UN-DU2,
3.1.3	Data Collection	F-CO-01.03	The TDEI system shall be capable of receiving fixed-route transit data from transportation service providers.	Demonstration	UN-TS1,
3.1.3	Data Collection	F-CO-01.04	The TDEI system shall be capable of receiving on-demand transit data from transportation service providers.	Demonstration	UN-TS1,

[5] Microservices vs. Monolithic Architecture. Atlassian. https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

[6] Microservice Architecture Style. Microsoft. https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.1.3	Data Collection	F-CO-01.05	The TDEI system shall be capable of receiving transit station layout data from transportation service providers.	Demonstration	UN-TS1,
3.1.3	Data Collection	F-CO-01.06	The TDEI system shall be capable of receiving data from crowdsourced applications to enable private citizens to identify needed local map updates and vet data submitted by others.	Demonstration	UN-DG8, UN-AD1a, UNAD12,
3.1.3	Data Collection	F-CO-05	The TDEI system shall provide formal processes for uploading data and metadata.	Inspection	UN-DG2, UN-TS1,

2.2.1.1 Justification for Choosing Microservices Infrastructure to Power the TDE

As noted, use of microservices is an architectural style in which a system is developed from many loosely coupled, independently developed services. The benefits of using microservices include[7,8]:

- Each microservice can be developed, deployed, and updated independently of other microservices, making system updates and the addition of new features easier.
- Each microservice is its own codebase so that different languages can be used for different system functions. (However, the number of languages used in the system should be strictly limited for maintainability.)
- Microservices have well-defined communication interfaces so that the internals of each microservice are encapsulated within that microservice. This means that other microservices need not be aware of those details. Other microservices need only understand the communication interfaces.
- Microservices can be independently scaled – that is, if one microservice is particularly complex or heavily-loaded and needs more resources than other simpler microservices, that microservice can be provided additional resources. Instead of scaling the entire (monolith) system, each microservice can be scaled independently.

Microservices also have some disadvantages:

[7] What is a Microservices Architecture? Google. https://cloud.google.com/learn/what-is-microservices-architecture

[8] Microservice Architecture Style. Microsoft. https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

- Microservices can increase system complexity. Managing the installation and deployment of many small independent microservices can be more complex than managing a single large system.
- Debugging and testing can become more complex. While each microservice can be tested independently, testing and debugging the system as a whole is more complex due to the asynchronous nature of the microservices and their communication.

In the context of the TDEI, each microservice service will implement a particular capability of the system itself based on data, technical, or other stakeholder requirements.

For the TDEI project, it has been determined that the flexibility and agility provided by a microservice architecture outweighs the increase in complexity from using a microservice architecture. Building a "monolithic" application that addresses the many needs and intertwined data logic, user interfaces, and other components that the decentralized municipal agencies, transit organizations and transportation marketplace dictate is not considered to be feasible. In addition, with a monolithic architecture, there would be limited means of growing the TDEI as application needs grow. Adding additional features would make the system progressively more difficult to maintain, deploy, test, and secure. Traditionally, the monolithic style of development has presented numerous challenges to organizations looking to respond quickly, reliably, and efficiently to changing application needs.

As an organically growing organization it is important to be wary of long development cycles, and high infrastructure and licensing costs. In addition, protecting against single points of failure and enabling scaling are key reasons for the choice of a microservice architecture for the TDEI. These benefits transcend the difficulty of testing and fixing bugs, addressing vulnerabilities in complex systems, and challenges integrating new technologies into the mix. In short, legacy architectures and monolithic applications are not compatible with the on-the-ground situation of the TDEI.

Rather than approach the TDEI's future growth through refactoring, repurposing or consolidation of legacy software, the team will be able to add services as use cases arise, to better align the tools and services the TDEI provides with the current data needs of the TDEI's stakeholders. To make the TDEI data infrastructure competitive, open and scalable, the project needs to remain nimble with its software being extensible and with the ability to evolve quickly. Utilizing a microservices based architecture helps achieve these goals. Our team thinks about microservices as individual puzzle pieces that come together to form something useful—like the ideal interoperable data sharing infrastructure environment. Each microservice can evolve independently based on the changing needs of the TDEI and the continuous development of the data schemas. An option would be to utilize outside vendors to fill in the feature and functionality gaps that are not core to the TDEI, allowing the UW development teams to focus on building business value.

In summary, the benefits of a microservices based architecture are clear for the TDEI. New features and functionality are faster to develop, test, and deploy. Services can be deployed independent of each other, and no single point of failure exists. This will allow for increased developer productivity and agility so that the demands of different project partners can be met head on.

Based on these justifications, the use of Microservice architecture in the articulation of the TDEI additionally traces back to the following system requirements:

Table 3: Traceability for microservice architecture within the TDEI implementation

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.1.3	Data Collection	F-CO-04.03	The data collection tools shall include technologies that help generate data that describe on-demand transportation services to enable widespread inclusion of those services by application developers into their trip planning software.	Demonstration	UN-TS2a,
3.1.4	Data Processing	F-PR-01	The TDEI system shall develop the data processing components that accept submitted data.	Demonstration	UN-DG2, UN-TS1,
3.1.4	Data Processing	F-PR-02	The TDEI system shall process the following data file formats:	Demonstration	UN-AD1,
3.1.4	Data Processing	F-PR-02.01	XML OpenStreetMap (.osm) files	Demonstration	UN-AD1,
3.1.4	Data Processing	F-PR-02.02	GTFS Comma Separated Values (.csv) files	Demonstration	UN-AD1,
3.1.4	Data Processing	F-PR-02.03	JavaScript Object Notation (.json) files	Demonstration	UN-AD1,
3.1.4	Data Processing	F-PR-03	The TDEI system shall support processes for:	Demonstration	UN-DG8, UN-DS1a, UN-DS3, UN-AD12,
3.1.4	Data Processing	F-PR-03.01	Vetting the data.	Demonstration	UN-DG8, UN-AD12,
3.1.4	Data Processing	F-PR-03.02	Aggregating the data.	Demonstration	UN-DS3,
3.1.4	Data Processing	F-PR-03.03	Managing the data.	Demonstration	UN-DS1a,
3.1.4	Data Processing	F-PR-03.04	Fusing the data.	Demonstration	UN-DS3,

2.2.1.2 Key Enabling Technology Components for Microservices: Containers and Orchestration Managers

The microservices architecture allows the TDEI team to satisfy the Data Lifecycle Requirements listed above, while still being able to leverage past work and open-source code for some of the more common data lifecycle functionalities. It is planned for the TDEI to use Docker to build microservice containers. Each Docker container is generally constrained in functionality, meaning there will be many of them. The system will then use Kubernetes, an open-source container orchestration platform, to manage the large number of containers required by the TDEI.

Figure 2 contains panels that functionally serve different purposes in the data lifecycle of the TDEI. The smaller labeled compartments within each panel describe microservices that can be decoupled from other activities of the TDEI. Some of the microservices featured in our architecture will be off the shelf and others will be developed in-house. The panels (from ingestion to dissemination) listed from left to right include variable inputs, API layer, Load and Ingest, Store and Replicate, Analytics and Realtime Processing, API Layer, and Dissemination. The leftmost panel is a functional representation of variable data inputs to the TDEI (rather than a panel representing functional units in the data lifecycle of the TDEI data with a containerized microservices view like the remaining panels). Below we explain how each of the functional panels, and the microservice containerized within those panels play a role in the Data Lifecycle of TDEI data.

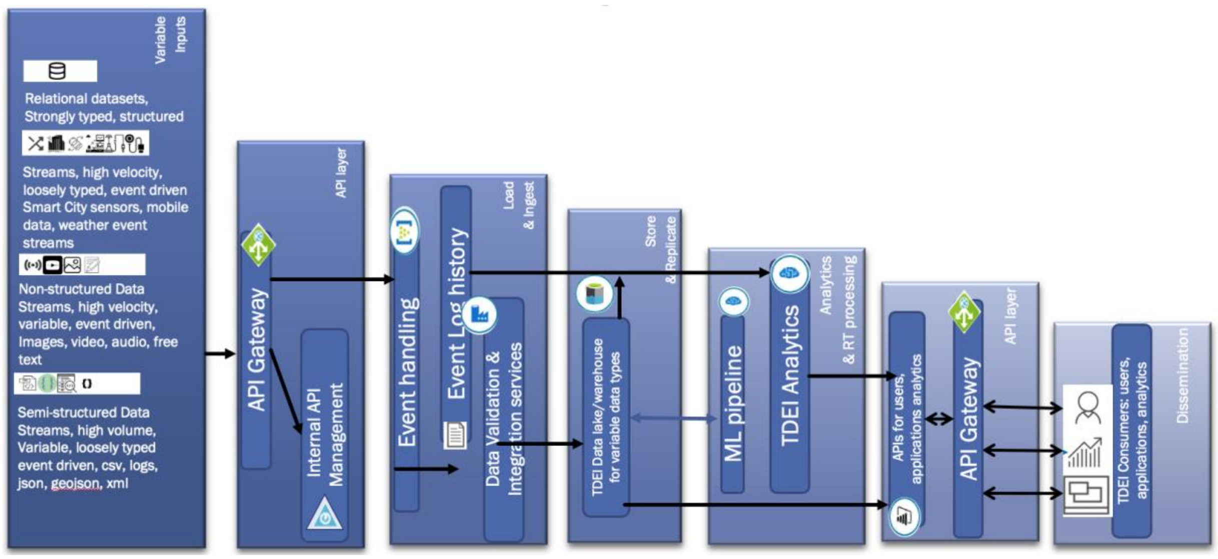

Variable Inputs
Relational datasets, Strongly typed, structured
Streams, high velocity, loosely typed, event driven Smart City sensors, mobile data, weather event streams
Non-structured Data Streams, high velocity, variable, event driven, Images, video, audio, free text
Semi-structured Data Streams, high volume, Variable, loosely typed event driven, csv, logs, json, geojson, xml
API layer
API Gateway
Internal API Management
Load & Ingest
Event handling
Event Log history
Data Validation & Integration services
Store & Replicate
TDEI Data lake/warehouse for variable data types
Analytics & RT processing
ML pipeline
TDEI Analytics
API layer
APIs for users, applications analytics
API Gateway
Dissemination
TDEI Consumers: users, applications, analytics

Figure 2 TDEI core functional view of components in the TDEI.

a. DATA COLLECTION- Represents the point at which new and/or existing data are collected or generated.

   i. Different methods for entry-- TDEI intends to offer connectors for variable data inputs. The leftmost panel represents those inputs, which may include data assets such as:

      - Street (road) network (this may be from ARNOLD[9] or OSM[10] road information) and may be ingested as batch input or as more atomic modifications represented by subgraphs of the network.

[9] ARNOLD – or All Road Network of Linear Referenced Data – is the geospatial roadway referencing database used for FHWA's Highway Performance Monitoring System.

[10] OSM – OpenStreetMap – is an open-source roadway network and data schema.

- On-demand Transit services described using the GTFS-Flex[11] data standard: snapshots of GTFS-Flex data from transit agencies.
- 3-d Images & LiDAR[12]: Three-dimensional images / point clouds of transit facilities provided by operators.
- Pathways Data (GTFS-Pathways[13]): Stream of GTFS-Pathways data from transit agencies.
- Street-level 2-d Imagery: Two-dimensional images of the roadways and footways.
- Crowdsourced Data: Data from human observers. Obtained via mappers who may be providing on-location data or data inferred from street-level imagery.

ii. Certification tools—separate microservices will be provided to authenticate data producers and ensure data is presented in correct digital format. Data is certified and signed by the certification tool, allowing the producer to provide a batch input or update to the TDEI (triggering a data ingestion). This is a TDEI-built microservice and will be containerized in the API Layer.

b. DATA PROCESSING- Represents the activities associated with the necessary preparation of various new or existing acquired data inputs. All the functionalities listed below will be implemented via the data integration services (shown in the Load and Ingest panel). All functionalities listed here are specific to each data schema and built by the TDEI (not off the shelf). These include data enrichment, computer vision inference, analytics, synonymous conflation (identifying the same element is provided twice but with different attributes), non-synonymous conflation (two elements are different spatially, but are actually referring to the same element in the world)

c. DATA QUALITY CONTROL- Represents the activities to measure and monitor data quality to ensure that the data are usable at any stage of the data life cycle. TDEI-specific microservice implementations will be deployed in both the API Layer and the Integration Server (shown in the Load and Ingest panel) to handle quality control (QC) activities.

d. DATA VALIDATION—Represents the activities to assess data upon entrance and offer a deeper analysis about the compatibility of the input data with the existing data

[11] GTFS-Flex: General Transit Feed Specification for flexible route services

[12] LiDAR: Light Detection and Ranging

[13] GTFS-Pathways: General Transit Feed Specification for pathways through transit facilities

before the data enters the stream. These TDEI-specific microservices go further than the digital certification above and tries to assess data compatibility and conflicts.

e. DATA DESCRIPTION- Represents the activities of identifying (PID, DOI, ...) and documenting the data with extended metadata (that could be defined with common semantics) to allow for understanding, harvesting, and consuming the data itself. This is a TDEI-specific microservice to digitally sign and enhance the data before ingestion to provide context to the data and influence downstream confidence metrics.

f. DATA UPDATE Represents the activities that directly change the content of data (reformat, add in updated sidewalk information, etc.). This is a microservice suite which is containerized within the Integration Server (Load and Ingest panel).

g. DATA EXTENSIBILITY Represents the activities of identifying and documenting new data attributes or data types to allow for data schema extensions and dynamic growth. This is hybrid microservice that is containerized within the data lake infrastructure. This would include off-the-shelf technology with customizations.

h. DATA SHARING/DATA PUBLISHING Represents the activities associated with making community data stores. This would be enabled through the publication of APIs wrapped through API Gateways (in the API Layer panel on the Dissemination end) to secure interactions with the data.

i. DATA DISCOVERY Represents the activities involved in finding data based on metadata and/or provenance information. The TDEI will use microservices and a database to set up a Registry for applications (upstream as well as downstream apps) that produce and consume TDEI data. There are several off the shelf containerized registry services that can be customized towards this purpose. These microservices will be part of the API Layer panel.

j. DATA ANALYSIS Represents the activities associated with the exploration and interpretation of well-managed, processed data for the purpose of knowledge discovery. The TDEI will provide aggregated information about TDEI activities, content, and data streaming in the data analytics microservices. These will be hybrid off-the-shelf technologies with customizations and containerized within the Analytics and Realtime Processing panel.

k. DATA PROVENANCE Represents the activities of documenting the various operations that occurred on data (data processing, data analysis, data transfer) to achieve reproducibility and referencing. This is a TDEI-specific microservice to digitally sign and enhance the data before ingestion to provide context to the data and influence downstream confidence metrics. Unlike the Data Description above, the Provenance Microservice interacts and ingests information about the Data Producer rather than the data itself.

l. DATA PERFORMANCE- Separate TDEI-specific microservices will be deployed for microservice and TDEI-system monitoring to diagnose and address any activities involving data access optimization, latency, or other performance degradations. It’s possible we would be able to incorporate some off the shelf technology in this container.

m. DATA BACKUP Represents the activities that involve the management of physical risks to the data throughout the data lifecycle. Routine local backups will be performed through off-the-shelf services which will likely be part of the database infrastructure itself (TDEI may not require a separate microservices architecture for this functionality).

n. DATA STORAGE and LONG-TERM PRESERVATION for long-term use, re-use and accessibility will be enabled by off-the-shelf services, also part of the database infrastructure itself (TDEI may not require a separate microservices architecture for this functionality).

Table 4: Traceability for specific microservices deployed within the TDEI implementation

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.5	Information Management	MAN-04	The TDEI system shall version-control updates made to the sidewalk graph network and transit data that is stored in the data repository.	Demonstration	UN-DG4,
3.5	Information Management	MAN-05	The TDEI system shall only distribute the latest approved version of sidewalk data when requested.	Demonstration	UN-DG4,
3.5	Information Management	MAN-06	The TDEI system shall only distribute the latest approved version of transit data when requested.	Demonstration	UN-DG4,
3.6	System Operations				
3.6.1	System Human Factors				
3.6.1	System Human Factors	S-HF-01	The TDEI system's data vetting tools shall:	Inspection	UN-DG8,
3.6.1	System Human Factors	S-HF-01.01	Have an intuitive user interface.	Inspection	UN-DG8,
3.6.1	System Human Factors	S-HF-01.02	Include clearly understood instructions for vetting data.	Inspection	UN-DG8,
3.6.1	System Human Factors	S-HF-01.03	Only require the minimal number of entries for user input as required.	Inspection	UN-DG8,
3.6.1	System Human Factors	S-HF-01.04	Allow the reviewer to request changes to the published data.	Inspection	UN-DG8,
3.6.1	System Human Factors	S-HF-01.05	Allow the originator of the data to approve or reject changes proposed by other system participants.	Inspection	UN-DG8,
3.6.1	System Human Factors	S-HF-02	The TDEI system's demonstration applications shall:	Inspection	UN-TS9, UN-AD7, UN-DU3, UN-DU4, UN-DU8, UN-DU9, UN-DU10, UN-DU11,
3.6.1	System Human Factors	S-HF-02.01	Communicate route and navigation information to an end user in a manner that is interpreted by the application's targeted user group (e.g., visual information for sighted travelers, auditory cues for blind travelers).	Inspection	UN-DU10,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.6.1	System Human Factors	S-HF-02.02	Interface with an end user using intuitive communication methods (e.g., haptic feedback, text-to-speech, etc.) for providing information, based on that application's target user group.	Inspection	UN-DU3, UN-DU8, UN-DU10,
3.6.1	System Human Factors	S-HF-02.03	Have intuitive inputs (for origin, destination, trip-specific travel preferences) and instructions for path routing applications.	Inspection	UN-DU4, UN-DU9, UN-DU11,
3.6.1	System Human Factors	S-HF-02.04	Provide intuitive explanations of local environmental attributes for spontaneous travel information applications.	Inspection	UN-DU3, UN-DU8,
3.6.1	System Human Factors	S-HF-02.05	Provide intuitive explanations of local built environment attributes for digital twin applications.	Inspection	UN-DU3, UN-DU8,
3.6.1	System Human Factors	S-HF-02.06	Provide users with the ability to provide input or corrections to sidewalk data.	Inspection	UN-TS9, UN-AD7,
3.6.1	System Human Factors	S-HF-02.07	Provide users with the ability to provide input or corrections to transit data.	Inspection	UN-TS9, UN-AD7,
3.6.2	System Maintainability				
3.6.2	System Maintainability	S-MN-01	The TDEI system and associated tools shall have a defined preventative maintenance program to check for issues.	Inspection	UN-DS1a, UN-DS2,
3.6.2	System Maintainability	S-MN-02	The TDEI system's demonstration applications should conduct regular preventative maintenance to detect and resolve any issues.	Demonstration	UN-DS2,
3.6.2	System Maintainability	S-MN-03	The TDEI system's demonstration applications should have a mechanism for user reporting of application errors.	Inspection	UN-DU11,
3.6.2	System Maintainability	S-MN-04	The TDEI system shall have a maintenance log to identify when issues are reported and when they are corrected.	Demonstration	UN-DS1a, UN-DS2,
3.6.3	System Reliability				
3.6.3	System Reliability	S-RL-01	The TDEI system shall operate in the normal mode of operation to be considered fully operational.	Demonstration	UN-DU4,
3.6.3	System Reliability	S-RL-02	The TDEI system shall automatically notify relevant maintenance staff in the event that the mode of operation is in one of the following states:	Demonstration	UN-DU4,
3.6.3	System Reliability	S-RL-02.01	Disrupted.	Demonstration	UN-DU4,
3.6.3	System Reliability	S-RL-02.02	Degraded.	Demonstration	UN-DU4,
3.6.3	System Reliability	S-RL-02.03	Failed.	Demonstration	UN-DU4,
3.7	Policy and Regulation				
3.7	Policy and Regulation	POL-01	The TDEI system shall include policies that allow sharing of the collected data.	Inspection	UN-DS1a,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.7	Policy and Regulation	POL-02	The TDEI system shall publish route information that is understood by the user to be "for information only", with no guarantee or expectation that that the supporting data is accurate.	Inspection	UN-DU2, UN-DU8, UN-DU9,
3.7	Policy and Regulation	POL-03	Any PII data collected by a demonstration application shall not be shared with the data repository.	Demonstration	UN-AD13,
3.8	System Lifecycle Sustainment				
3.8	System Lifecycle Sustainment	LIF-01	The TDEI system and associated tools shall have the capability to remain functional during the duration of the ITS4US project.	Inspection	UN-DG2, UN-TS5, UN-AD12,
3.8	System Lifecycle Sustainment	LIF-02	The TDEI system and associated tools shall not be restricted from being adopted and incorporated into another data service provider's program.	Inspection	UN-DU5,
3.8	System Lifecycle Sustainment	LIF-03	The TDEI system shall be capable of accommodating updated data flows as data schemas or standards change.	Test	UN-DG3, UN-DG5, UN-TS5,
3.1.3	Data Collection	F-CO-02	The built environment features received by the TDEI system shall adhere to the following:	Demonstration	UN-DG1, UN-AD10a, UN-DU8,
3.1.3	Data Collection	F-CO-02.01	The built environment features shall be tagged correctly in the data schema.	Demonstration	UN-AD10a,
3.1.3	Data Collection	F-CO-02.02	The built environment features shall be able to support nongraphic representation.	Demonstration	UN-DG1,
3.1.3	Data Collection	F-CO-02.03	The built environment features shall be intuitive so that digital device end users can indicate their preferences.	Demonstration	UN-DU8,
3.1.3	Data Collection	F-CO-03	The TDEI system shall provide data translation tools.	Demonstration	UN-DG1, UN-TS1,
3.1.3	Data Collection	F-CO-03.01	The TDEI system shall provide tools for sidewalk data producers to translate existing sidewalk data into the OpenSidewalks data format.	Demonstration	UN-DG1,
3.1.3	Data Collection	F-CO-03.02	The TDEI system shall provide tools for transit data producers to translate existing fixed-route data into the GTFS data format and associated extensions.	Demonstration	UN-TS1, UN-TS6,
3.1.3	Data Collection	F-CO-03.03	The TDEI system shall provide tools for transit data producers to translate existing on-demand data into the GTFS data format and associated extensions.	Demonstration	UN-TS1,
3.1.3	Data Collection	F-CO-03.04	The TDEI system shall provide tools for transit data producers to translate existing transit station data into the GTFS data format and associated extensions.	Demonstration	UN-TS1,
3.1.3	Data Collection	F-CO-04	The TDEI system shall provide data collection tools.	Demonstration	UN-DG1, UN-DG3, UN-TS2a,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.1.3	Data Collection	F-CO-04.01	The data collection tools shall convert data into compatible or conflatable to the refined data standards for OpenSidewalks, GTFS data format or, where applicable, a comparable extension.	Demonstration	UN-DG3, UN-AD1b, UN-AD5,
3.1.3	Data Collection	F-CO-04.02	The data collection tools shall include automated sidewalk data collection technologies (e.g., advanced analytics used by mapping technology companies) to populate sidewalk databases.	Demonstration	UN-DG1,
3.1.4	Data Processing	F-PR-03	The TDEI system shall support processes for:	Demonstration	UN-DG8, UN-DS1a, UN-DS3, UN-AD12,
3.1.4	Data Processing	F-PR-03.01	Vetting the data.	Demonstration	UN-DG8, UN-AD12,
3.1.4	Data Processing	F-PR-03.02	Aggregating the data.	Demonstration	UN-DS3,
3.1.4	Data Processing	F-PR-03.03	Managing the data.	Demonstration	UN-DS1a,
3.1.4	Data Processing	F-PR-03.04	Fusing the data.	Demonstration	UN-DS3,
3.1.4	Data Processing	F-PR-04	The TDEI system shall facilitate the processing of data into routable pathways networks.	Demonstration	UN-DG1, UN-DS6, UN-DS6a,
3.1.4	Data Processing	F-PR-04.01	The routable pathway networks shall describe the path infrastructure in objective detail.	Demonstration	UN-DG1,
3.1.4	Data Processing	F-PR-04.02	The routable pathway networks shall include pathway locations.	Demonstration	UN-DG1,
3.1.4	Data Processing	F-PR-04.03	The routable pathway networks shall include pathway connectivity.	Demonstration	UN-DG1,
3.1.4	Data Processing	F-PR-04.04	The routable pathway networks shall include pathway features.	Demonstration	UN-DG1,
3.1.4	Data Processing	F-PR-04.05	The routable pathway networks shall include pathway characteristics.	Demonstration	UN-DG1,
3.1.4	Data Processing	F-PR-04.06	The routable pathway networks shall include connectivity of features across different levels of transit stations.	Demonstration	UN-DS6,
3.1.4	Data Processing	F-PR-04.07	The routable pathway networks shall ensure that data linkages exist when different transit agencies share a physical transit stop.	Demonstration	UN-DS6a,
3.1.4	Data Processing	F-PR-04.08	The TDEI system shall augment relevant links where connectivity exists.	Demonstration	UN-DG1,
3.1.5	Data Quality Control				
3.1.5	Data Quality Control	F-QC-01	The TDEI system shall require a data vetting process for all data before they are deposited into the core data repository to identify invalid data that have been reported.	Demonstration	UN-DG8,
3.1.5	Data Quality Control	F-QC-02	The TDEI system shall provide access to data vetting tools.	Demonstration	UN-DG8, UN-AD11,
3.1.5	Data Quality Control	F-QC-02.01	The data vetting tools shall confirm whether the data conform to standards.	Demonstration	UN-DG8,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.1.5	Data Quality Control	F-QC-02.02	The data vetting tools shall confirm whether the data are of sufficient accuracy.	Demonstration	UN-DG8,
3.1.5	Data Quality Control	F-QC-02.03	The data vetting tools shall confirm whether the data are of consistent quality.	Demonstration	UN-DG8,
3.1.5	Data Quality Control	F-QC-02.04	The data vetting tools shall describe when data are missing.	Demonstration	UN-DG8,
3.1.5	Data Quality Control	F-QC-02.05	The data vetting tools shall support automated data vetting activities (e.g., automated data review to check for data format and permissible data).	Demonstration	UN-DG8,
3.1.5	Data Quality Control	F-QC-02.06	The data vetting tools shall support manual data vetting activities (e.g., owner/hired consultant review, community/organization reviews, traveler feedback).	Demonstration	UN-DG8,
3.1.5	Data Quality Control	F-QC-02.07	The data vetting tools shall generate a degree of confidence associated with the data being published.	Demonstration	UN-AD11,
3.1.5	Data Quality Control	F-QC-03	The TDEI system shall include the development of validation toolsets for assembling sidewalk and transit environment data from multiple providers.	Demonstration	UN-DG7, UN-TS2a,
3.1.6	Data Storage				
3.1.6	Data Storage	F-ST-01	The TDEI system shall include the creation of the centralized data repositories.	Inspection	UN-AD9,
3.1.6	Data Storage	F-ST-02	The TDEI system shall include the operation of the centralized data repositories.	Inspection	UN-AD9,
3.1.6	Data Storage	F-ST-03	The TDEI system shall include the maintenance of the centralized data repositories.	Inspection	UN-AD9,
3.1.6	Data Storage	F-ST-04	The TDEI system shall transmit approved data to centralized data repositories.	Demonstration	UN-AD12,
3.1.6	Data Storage	F-ST-05	The data repository shall include these types of data:	Demonstration	UN-DS4, UN-AD4, UN-AD6, UN-AD8, UN-AD9, UN-DU3, UN-DU7,
3.1.6	Data Storage	F-ST-05.01	Fixed-route transit data.	Demonstration	UN-AD9,
3.1.6	Data Storage	F-ST-05.02	On-demand transit data.	Demonstration	UN-AD4,
3.1.6	Data Storage	F-ST-05.03	Transit station data.	Demonstration	UN-DS4,
3.1.6	Data Storage	F-ST-05.04	Graphed sidewalk network data.	Demonstration	UN-AD8,
3.1.6	Data Storage	F-ST-05.05	Mode transfer options.	Demonstration	UN-AD4, UN-DU7,
3.1.6	Data Storage	F-ST-05.06	Travel environments that connect mode transfers or trip segments.	Demonstration	UN-AD6, UN-DU7,
3.1.6	Data Storage	F-ST-05.07	First- and last-mile options.	Demonstration	UN-DU3,
3.1.7	Data Update				

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.1.7	**Data Update**	**F-UP-01**	**The data repository shall support continuous updates.**	**Demonstration**	**UN-DG4,**
3.1.8	Data Sharing/Data Publishing				
3.1.8	Data Sharing/Data Publishing	F-SH-01	The TDEI system shall include data provisioning services that distribute data-on-demand for use in a variety of applications.	Demonstration	UN-TS1, UN-AD10, UN-DU5, UN-DS5,
3.1.8	Data Sharing/Data Publishing	F-SH-01.01	Data that is shared through the TDEI system shall be published on a web service, either open to the public or through a requested API service.	Demonstration	UN-TS1,
3.1.8	Data Sharing/Data Publishing	F-SH-01.02	Data that is shared through the TDEI system shall be accessible for different geographic locations.	Demonstration	UN-AD10, UN-DU5, UN-DS5,
3.1.8	Data Sharing/Data Publishing	F-SH-02	The TDEI system shall support interoperable sharing.	Demonstration	UN-DS2, UN-DS4,
3.1.8	Data Sharing/Data Publishing	F-SH-03	The TDEI system shall support two-directional communication channels between the central database and the organizations that “own” the facility or service being described with data.	Demonstration	UN-DS8,
3.1.9	Data Discovery				
3.1.9	Data Discovery	F-DI-01	The TDEI system shall use public-facing APIs to exchange data with application developers.	Demonstration	UN-AD1,
3.1.9	Data Discovery	F-DI-02	The OpenSidewalks data service shall perform the following functionality:	Demonstration	UN-AD10a,
3.1.9	Data Discovery	F-DI-02.01	Receive the request from the application through its secure API.	Demonstration	UN-AD10a,
3.1.9	Data Discovery	F-DI-02.02	Verify via the application’s descriptive metadata.	Demonstration	UN-AD10a,
3.1.9	Data Discovery	F-DI-02.03	Request/query all relevant data from the data repository.	Demonstration	UN-AD10a,
3.1.9	Data Discovery	F-DI-02.04	Send all relevant data to the application that made the original request.	Demonstration	UN-AD10a,
3.1.9	Data Discovery	F-DI-03	The GTFS data service shall perform the following functionality:	Demonstration	UN-AD10b,
3.1.9	Data Discovery	F-DI-03.01	Receive the request from the application through its secure API.	Demonstration	UN-AD10b,
3.1.9	Data Discovery	F-DI-03.02	Verify via the application’s descriptive metadata.	Demonstration	UN-AD10b,
3.1.9	Data Discovery	F-DI-03.03	Request/query all relevant data from the data repository.	Demonstration	UN-AD10b,
3.1.9	Data Discovery	F-DI-03.04	Send all relevant data to the application that made the original request.	Demonstration	UN-AD10b,
3.1.10	Data Analysis				
3.1.11	Data Dissemination				
3.1.11	Data Dissemination	F-DS-01	The demonstration applications shall utilize the data returned by the TDEI data services to:	Demonstration	UN-DU9,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.1.11	Data Dissemination	F-DS-01.01	Identify paths using the most up-to-date sidewalk and/or transit data (including any paths that involve a sidewalk path option or transit option, if requested or available).	Demonstration	UN-DU9,
3.1.11	Data Dissemination	F-DS-01.02	Screen those paths based on the user's trip-specific travel preferences.	Demonstration	UN-DU9,
3.1.11	Data Dissemination	F-DS-01.03	Provide one or more recommended routes, when such a route exists, to the end user.	Demonstration	UN-DU9,
3.1.11	Data Dissemination	F-DS-02	The TDEI system shall support the development of tools to make informed, customized travel decisions.	Demonstration	UN-DU9,
3.1.11	Data Dissemination	F-DS-02.01	The TDEI system shall utilize mobile applications to demonstrate the system by providing a sidewalk route based on user-defined travel preferences.	Demonstration	UN-DU9,
3.1.11	Data Dissemination	F-DS-02.02	The TDEI system shall utilize mobile applications to demonstrate the system by providing paths through transit stations based on user-defined travel preferences.	Demonstration	UN-DU9,
3.1.11	Data Dissemination	F-DS-02.03	The TDEI system shall utilize mobile applications to demonstrate the system by providing on-demand transit options based on user-defined travel preferences.	Demonstration	UN-DU9,
3.1.11	Data Dissemination	F-DS-02.04	The TDEI system shall utilize mobile applications to demonstrate the system providing data that supports spontaneous navigation of an end user's local environment.	Demonstration	UN-DU9,
3.1.11	Data Dissemination	F-DS-03	The TDEI system's demonstration applications shall use intuitive interfaces that minimizes confusion for targeted user groups.	Demonstration	UN-DU2, UN-DU4, UN-DU5,
3.1.11	Data Dissemination	F-DS-03.01	The TDEI system's demonstration applications shall help the traveler identify when errors have occurred.	Demonstration	UN-DU2,
3.1.11	Data Dissemination	F-DS-03.02	The TDEI system's demonstration applications shall provide easily accessed "help" functions that allow users to quickly obtain information about how to safely navigate from their current location.	Demonstration	UN-DU2,
3.1.11	Data Dissemination	F-DS-04	The TDEI system's demonstration applications shall be designed to continue operations despite missing data.	Demonstration	UN-AD11,
3.1.11	Data Dissemination	F-DS-05	The TDEI system's demonstration applications shall support the delivery of information to users in different formats (audio, text, tactile displays) based on the intended audience of the demonstration application.	Demonstration	UN-TS5b,
3.1.11	Data Dissemination	F-DS-06	The TDEI system's demonstration applications shall provide information regarding transit service capabilities to	Demonstration	UN-DU2,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
			help travelers avoid potential hazardous outcomes.		
3.1.11	Data Dissemination	F-DS-07	The TDEI system's demonstration applications shall provide insight on areas where data quality is reported to be poor to help travelers make informed decisions.	Demonstration	UN-DU8,
3.1.11	Data Dissemination	F-DS-08	The TDEI system's demonstration applications shall issue low-power warnings within the application.	Demonstration	UN-DU2,
3.1.11	Data Dissemination	F-DS-09	The TDEI system shall safeguard PII data deemed necessary for operation.	Inspection	UN-AD13, UN-DU1,
3.1.11	Data Dissemination	F-DS-10	The TDEI system shall permit approved third-party mobile applications to utilize sidewalk or transit data for other routing and navigation purposes.	Demonstration	UN-AD3,
3.1.12	Data Provenance				
3.1.12	Data Provenance	F-PV-01	Changes approved and committed to the data repository shall document and timestamp a new version.	Demonstration	UN-DG4, UN-AD1a,
3.1.12	Data Provenance	F-PV-02	Change records shall be traceable to the agencies/organizations that perform data vetting.	Demonstration	UN-AD1a,
3.1.12	Data Provenance	F-PV-03	Change records shall be traceable to the agencies/organizations that respond to data vetting reports.	Demonstration	UN-AD1a,
3.1.12	Data Provenance	F-PV-04	Date stamps shall be present to ensure that the data are valid for specific dates and are not used past valid time periods.	Demonstration	UN-AD1a,
3.1.12	Data Provenance	F-PV-05	Two-way information sharing shall reference to the originator of the data.	Demonstration	UN-AD1a,
3.1.13	Data Performance				
3.1.14	Data Backup				
3.1.14	Data Backup	F-BA-01	Data stored in the data repository shall be backed up periodically so that in the event of a system issue (e.g., data loss, data corruption, application outage), failover will occur, and the data repository will remain available.	Inspection	UN-AD9,
3.1.14	Data Backup	F-BA-01.01	Backup methods used shall meet USDOT requirements for records retention.	Inspection	UN-AD9,
3.1.14	Data Backup	F-BA-01.02	Backup methods shall archive, at a minimum, each data contribution that is provided by a data contributor.	Inspection	UN-AD9,
3.1.14	Data Backup	F-BA-01.03	Research data collected as part of the ITS4US Program as well as production data shall be backed up.	Inspection	UN-AD9,
3.1.14	Data Backup	F-BA-02	Data back-ups shall be sent to an offsite location or a cloud service in the event of widespread damage to the proposed system's primary location.	Inspection	UN-AD9,

SyRS Section #	**Requirement Type**	**Requirement ID**	**Requirement Text**	**Verification Method**	**User Need**
3.1.14	Data Backup	F-BA-03	Recovery of back-up data shall occur in a timely fashion upon initiation of the restoration effort.	Test	UN-AD9,
3.1.15	Data Long Term Preservation				
3.1.1	Data Description	F-DE-01.01	Pedestrian built environment shall be described using the OpenSidewalks data standard.	Inspection	UN-AD10a,
3.1.1	Data Description	F-DE-01.01	Pedestrian built environment shall be described using the OpenSidewalks data standard.	Inspection	UN-AD10a,
3.1.1	Data Description	F-DE-01.02	Transportation stations and hubs shall be described using the General Transit Feed Specification Pathways (GTFS-Pathways) data standard.	Inspection	UN-AD9,
3.1.1	Data Description	F-DE-01.03	Demand responsive travel services shall be described using the GTFS-Flex data standard, excluding real-time feeds.	Inspection	UN-AD4,
SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.2	Physical				
3.2.1	Construction				
3.2.1	Construction	P-CO-01	The TDEI system shall have its processing elements and data repository be stored in a networked central server environment.	Inspection	UN-AD9,
3.2.1	Construction	P-CO-02	The TDEI system shall provide network connections to the following tools from many physical locations:	Test	UN-DG2, UN-TS1,
3.2.1	Construction	P-CO-02.01	Data collection tools.	Test	UN-DG2, UN-TS1,
3.2.1	Construction	P-CO-02.02	Data translation tools.	Test	UN-DG2, UN-TS1,
3.2.1	Construction	P-CO-02.03	Data vetting tools.	Test	UN-DG2, UN-TS1,
3.2.1	Construction	P-CO-03	The following tools that are a part of the TDEI system shall operate on standard office computer hardware or standard mobile tablet devices:	Test	UN-DG2, UN-TS1,
3.2.1	Construction	P-CO-03.01	Data collection tools.	Test	UN-DG2, UN-TS1,
3.2.1	Construction	P-CO-03.02	Data translation tools.	Test	UN-DG2, UN-TS1,
3.2.1	Construction	P-CO-03.03	Data vetting tools.	Test	UN-DG2, UN-TS1,
3.2.1	Construction	P-CO-04	The TDEI system's demonstration applications shall operate on standard internet browsers or mobile devices (Android, iOS).	Test	UN-DU2, UN-DU10,
3.2.1	Construction	P-CO-05	The TDEI system and all associated components shall send data successfully over landline or wireless internet without priority or special accommodation (e.g., VPNs).	Test	UN-DG2, UN-TS1, UN-DU2, UN-DU10,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.2.2	Durability				
3.2.2	Durability	P-DU-01	The TDEI system’s processing elements shall be able to accommodate:	Analysis	UN-DG2, UN-TS1, UN-DS2, UN-AD1, UN-DU3,
3.2.2	Durability	P-DU-01.01	Multiple data contributors providing sidewalk and/or transit-related data submissions simultaneously.	Analysis	UN-DG2, UN-TS1, UN-DS2,
3.2.2	Durability	P-DU-01.02	Multiple applications and application account users requesting sidewalk and/or transit-related data submissions simultaneously.	Analysis	UN-AD1, UN-DU3,
3.2.2	Durability	P-DU-02	The TDEI system's data translation tools shall accommodate sidewalk and/or transit data contributions.	Analysis	UN-DG2, UN-TS1,
3.2.2	Durability	P-DU-03	The TDEI system's demonstration applications shall:	Test	UN-AD1, UN-AD4, UN-AD6, UN-AD7, UN-AD9, UN-AD10a, UN-AD10b,
3.2.2	Durability	P-DU-03.01	Request relevant sidewalk and/or transit data to users that is sufficient for their trip needs.	Test	UN-AD1,
3.2.2	Durability	P-DU-03.02	Receive relevant sidewalk and/or transit data to users that is sufficient for their trip needs.	Test	UN-AD9, UN-AD10a, UN-AD10b,
3.2.2	Durability	P-DU-03.03	Present relevant sidewalk and/or transit data to users that is sufficient for their trip needs.	Test	UN-AD4, UN-AD6, UN-AD7,
3.2.4	Environmental Conditions	P-EN-02	The TDEI system's data translation tools shall operate without degradation in environments approved for consumer PCs and mobile devices.	Analysis	UN-DG2, UN-TS1,
3.2.4	Environmental Conditions	P-EN-03	The TDEI system's data demonstration applications shall:	Analysis	UN-DU2,
3.2.4	Environmental Conditions	P-EN-03.01	Operate without degradation in environments approved for consumer PCs and mobile devices.	Analysis	UN-DU2,
3.2.4	Environmental Conditions	P-EN-03.02	Operate with full capabilities to the end user without disruption from ambient background noise common in their travel environment (e.g., sidewalks near traffic, etc.).	Analysis	UN-DU2,
3.3	System Performance				
3.3	System Performance	PER-01	The TDEI system shall be perceived as reliable by end users (e.g., with minimal system freezes, crashes, and failures).	Analysis	UN-DU11,
3.3	System Performance	PER-02	The TDEI system shall adhere to the following system performance targets:	Analysis	UN-DG2, UN-TS1, UN-DU2,
3.3	System Performance	PER-02.01	The TDEI system comprehensively shall be operational 99.5% of the time 24 hours a day, 365 days per year.	Analysis	UN-DG2, UN-TS1, UN-DU2,
3.3	System Performance	PER-02.02	The TDEI system's data collection tools shall be operational 99.5% of the time 24 hours a day, 365 days per year.	Analysis	UN-DG2, UN-TS1,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.3	System Performance	PER-02.03	The TDEI system's data translation tools shall be operational 99.5% of the time 24 hours a day, 365 days per year.	Analysis	UN-DG2, UN-TS1,
3.3	System Performance	PER-02.04	The TDEI system's data vetting tools shall be operational 99.5% of the time 24 hours a day, 365 days per year.	Analysis	UN-DG2, UN-TS1,
3.3	System Performance	PER-02.05	The TDEI system shall allow for data vetting to occur in a timely manner that keeps data current.	Analysis	UN-DG2, UN-TS1, UN-DS7,
3.3	System Performance	PER-02.06	Data that are uploaded to the TDEI system's data repository shall be uploaded without errors 99% of the time.	Analysis	UN-DG2, UN-TS1,
3.3	System Performance	PER-02.07	The TDEI system's processing and data repository components shall be operational 99.5% of the time 24 hours a day, 365 days per year.	Analysis	UN-DU2,
3.3	System Performance	PER-02.08	The TDEI system's data services shall be operational 99.5% of the time 24 hours a day, 365 days per year.	Analysis	UN-DU2,
3.4	System Security and Privacy	SEC-03	The TDEI system shall require permission from end users for use of data that may be considered Locational PII prior to data being collected.	Demonstration	UN-DU1,
3.4	System Security and Privacy	SEC-04	The TDEI system shall protect user privacy to the extent possible.	Inspection	UN-AD13, UN-DU6,
3.4	System Security and Privacy	SEC-09	The TDEI system shall utilize NIST SP800-53, Recommended Security Controls for Federal Information Systems and Organizations for guidance to manage system safety risks.	Inspection	UN-DU11,

2.2.2 Message Streaming and Brokering: Enable Integration of the Data Interoperability Platform

Data systems have become quite complex, particularly with the advent of distributed, cloud computing and real-time data streams. Whereas previously, data systems had synchronous communication between or among modules, it is no longer feasible to have all modules (including applications, microservices, databases and any consuming application or producing application that reads and writes data over a network) communicating directly with each other. Point-to-point communication, as it was called, is simple to maintain and reason about when there are but a small number of systems. However, as described in the book by Mitch Seymour, *Mastering Kafka Streams and ksqld*[14], "when more subsystems need to communicate, point-to-point direct

[14] The Key to Mastering Kafka Streams and ksqlDB by Mitch Seymor, ISBN-10: 1492062499, March 16, 2021

communication is difficult to scale. The result is a complex web of communication pathways that can be difficult to reason about and maintain." In his book, Seymour goes on to summarize the drawbacks of the client-server model and the issues arising in point-to-point communication patterns with the following points:

- "Systems become tightly coupled because their communication depends on knowledge of each other. This makes maintaining and updating these systems more difficult than it needs to be.
- "Synchronous communication leaves little room for error since there are no delivery guarantees if one of the systems goes offline.
- "Systems may use different communication protocols, scaling strategies to deal with increased load, failure-handling strategies, etc. As a result, you may end up with multiple species of systems to maintain (software speciation), which hurts maintainability.
- "Receiving systems can easily be overwhelmed, since they don't control the pace at which new requests or data comes in. Without a request buffer, they operate at the whims of the applications that are making requests.
- "There isn't a strong notion for what is being communicated between these systems. The nomenclature of the client-server model has put too much emphasis on requests and responses, and not enough emphasis on the data itself. Data should be the focal point of data-driven systems.
- "Communication is not re-playable. This makes it difficult to reconstruct the state of a system."

In the case of the TDEI, there are multiple, diverse stakeholders and it is notoriously difficult to perform data communications to multiple stakeholders (or tenants) well. To produce interoperable data infrastructure and negotiate messaging among all the microservice APIs, the TDEI needs to provide a well-managed, low-latency data streaming platform. Emphasis will be placed on efficiency, customizability, power, and reliability.

The need for asynchronous, distributed messaging is greater even in traditionally non-data driven industries like transportation. The TDEI can take note from small and large enterprises that build big, highly customized data pipelines. In large data companies (like Netflix, for example), enterprises make common use of an open-source platform called Apache Kafka as a backbone for this kind of infrastructure. Many other open-source projects are built on top of open-source messaging architecture like Kafka.

Whether the TDEI uses Apache Kafka or one of the other options that have appeared on the market since 2010 (including Apache PULSAR, or Redpanda), the TDEI will use the infrastructure as an **event bus**. In an event bus system, subsystem services called **producers** produce events — or messages, and publish them, or write, events to the TDEI topic streams. The event bus infrastructure receives these events and records them into an ordered message history. Other TDEI subsystem services called **consumers** subscribe to, or read, those events in chronological order and are notified in real time as new messages are produced. In contrast with a traditional database, which is well-suited for queries and updates against a current state, event buses excel for applications that must quickly act on the various changes that lead to a transient current state. This type of handling is appropriate for transportation and mobility data which should not rely on data releases and batch processed types of data updates but can still handle such batch events.

Event bus messaging simplifies communication between systems by acting as a centralized communication hub in which systems can send and receive data without knowledge of each other. The communication pattern it implements is called the publish-subscribe (pub/sub) pattern.

In the pub/sub communication model, instead of having multiple systems communicate directly with each other. Producers publish their data to one or more **topics**, without connectivity to any modules that may consume the data. Topics consist of named streams (or channels) of related data that are stored in a cluster. They serve a similar purpose as tables in a database. However, they do not impose a particular schema, but rather store the raw data, which makes them very flexible. Consumers are processes that read (or subscribe) to data in one or more topics. They do not communicate directly with the producers, but rather listen to data on any stream they happen to be interested in. Consumers can work together as a group (called a consumer group) to distribute work across multiple processes.

2.2.2.1 Justification for Choosing Event Bus Messaging

The publish/subscribe communication model, which puts more emphasis on flowing streams of data that can easily be read from and written to by multiple processes, comes with several advantages, including:

- Systems become decoupled and easier to maintain because they can produce and consume data without knowledge of other systems.
- Asynchronous communication comes with stronger delivery guarantees. If a consumer goes down, it will simply pick up from where it left off when it comes back online again (or, when running with multiple consumers in a consumer group, the work will be redistributed to one of the other members).
- Systems can standardize on the communication protocol (a high-performance binary Transmission Control Protocol (TCP) is used when talking to Kafka clusters), as well as scaling strategies and fault-tolerance mechanisms (which are driven by consumer groups). This allows us to write software that is broadly consistent.
- Consumers can process data at a rate they can handle. Unprocessed data is generally stored (this varies in different platforms) in a durable and fault-tolerant manner, until the consumer is ready to process it. This protects consumers from having to process data at the same pace it is produced. The event handling platform will instead act as a buffer, preventing consumers from being overwhelmed.
- A stronger notion of what data are being communicated is in the form of events. An event is a piece of data with a certain structure and payload. Using event streaming allows the TDEI to focus on the data flowing through the streams, instead of spending time disentangling the communication layer like we would in the client-server model.

- Systems can rebuild their state anytime by replaying the events in a topic. This will come in handy as the TDEI attempts to reconstruct the history of mobility data in a certain data schema for a particular region.

2.2.2.2 Key Enabling Technology Components for Event Bus

Event buses provide an alternative to older messaging queues or monolith data-payload communication capabilities. The architecture can be easily scaled, by adding more nodes to the cluster and partitions to individual topics. Additionally, message brokers can persist messages for a configurable period rather than deleting them as soon as they reach the consumer.

Table 5 Traceability for use of Event Bus Architecture as the enabling technology to integrate TDEI microservices, conferring capabilities for the Data Interoperability Platform

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.2.2	Durability	P-DU-04	The TDEI system and all affiliated tools shall be capable of operating in isolation from other components with reductions in features.	Analysis	UN-DU10,

2.2.3 Application Programming Interfaces and API Layers

The TDEI will rely heavily on the use of Application Programming Interface (APIs) and API layers. On their technology support website,[15] IBM describes APIs as follows.

> "An application programming interface (API) enables different entities and partners to open their applications' data and functionality to external third-party developers, business partners, and internal departments within their organizations. This allows services and products to communicate with each other and leverage each other's data and functionality through a documented interface. Developers don't need to know how an underlying service is implemented; they simply use the API to communicate with other products and services. API use has surged over the past decade, to the degree that many of the most popular web applications today would not be possible without APIs."

APIs and API Gateways will be used in multiple ways within the TDEI infrastructure to achieve TDEI interoperability goals. Microservices are often interfaced with via APIs (as alluded to in the left panel labeled 2.2.2 in Figure 1). However, data publication to downstream TDEI consuming applications like AccessMap MultiModal will also be accomplished using APIs. The internal API management instance (among microservices) could also be exposed to external users to allow for utilization of the full potential of the APIs. Whether internally or externally exposed, this could

[15] https://www.ibm.com/au-en/topics/api

be achieved using API Gateways forwarding requests to the internal API services, which in turn interface with the microservices deployed in the TDEI.

2.2.3.1 Justification for APIs

Using APIs will provide the TDEI with the advantage of being able to serve many different data producers and consumers without having knowledge of the underlying development of the data pipelines. Use of APIs means the TDEI team can leverage many of the same tools and solutions that have grown in the RESTful and web service ecosystem. One of the advantages of developing our microservices with a particular API design, is that it enables natural ways to monitor and test these APIs, which will enable the TDEI to validate the flow of data and information throughout our microservice deployment, even when some of these services are not running on our cloud, but in the hands of some of the Data Service Providers or Transportation Service Providers. APIs have been in wide use for over 30 years in industry and are a proven technology.

In general, use of APIs in the context of the TDEI, traces back to the following system requirements:

Table 6 Traceability for use of APIs in microservice implementation

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.2.2	Durability	P-DU-04	The TDEI system and all affiliated tools shall be capable of operating in isolation from other components with reductions in features.	Analysis	UN-DU10,
3.2.3	Adaptability	P-AD-01	The TDEI system and all affiliated tools shall accommodate scalable information increases as new data is added to the system.	Analysis	UN-DS3,
3.1.1	Data Description	F-DE-05	Data standard specifications shall be scalable, extensible, and interoperable in different geographic markets or to different user populations.	Inspection	UN-AD3, UN-DU5,
3.1.1	Data Description	F-DE-06	Data standard schemas shall be made available to data generators.	Inspection	UN-DG3,
3.1.1	Data Description	F-DE-06.02	Data standard schemas shall use standard classifications and vocabularies.	Inspection	UN-DG5,
4	System Interfaces				
4.1	Internal System Interfaces				
4.1	Internal System Interfaces	INT-01	The TDEI system shall pass sidewalk data from the sidewalk data collectors to the sidewalk data processing components.	Demonstration	UN-DU2,
4.1	Internal System Interfaces	INT-02	The TDEI system shall pass transit data from the transit data collectors to the transit data processing components.	Demonstration	UN-TS1,
4.1	Internal System Interfaces	INT-03	The TDEI system shall pass data from the data processing components to the data repository.	Demonstration	UN-AD3,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
4.1	Internal System Interfaces	INT-04	The data repository shall pass sidewalk data to the sidewalk data service components.	Demonstration	UN-AD3,
4.1	Internal System Interfaces	INT-05	The data repository shall pass transit data to the transit data service components.	Demonstration	UN-AD3,
4.1	Internal System Interfaces	INT-06	TDEI system shall use software toolsets to input observations into translated data.	Demonstration	UN-DU2, UN-TS1,
4.1	Internal System Interfaces	INT-07	The TDEI system shall use software applications to interface between the data and the end user.	Demonstration	UN-AD2, UN-AD4, UN-AD6,
4.2	External System Interfaces				
4.2	External System Interfaces	EXT-01	The TDEI system shall pass data to approved third-party applications.	Demonstration	UN-DU3,
4.2	External System Interfaces	EXT-02	The TDEI system shall pass data to an USDOT-managed system.	Demonstration	UN-DU3,

2.2.3.2 TDEI Governance in Using APIs

TDEI development partners include transit agencies, mobility service providers, and mobility data service providers, application developers and the technology components designed to consumer mobility data. Regardless of what subsystem services will be designed by TDEI partners, the following API development principles will be used:

Open Standards: All development partners will describe their data through specific ontologies, schemas or formats that meet the criteria of an open standard, as defined by v1 Mobility Data Interoperability Principles.[16]

Open Standards Compatibility: All development partners will publish open standards with data stored in a way providing the ability to ingest and consume valid open standards.

Publish Data: All development partners will publish data via a documented application programming interface (API), which may require a generated API key to access. All development partners will expose their mobility data and functionality through API service interfaces.

Programmatic Access Only: All development partners providing data will provide a method of accessing information in which computer programs can exchange information and commands without requiring human intervention. There will be no other form of inter-process communication among TDEI partners and mobility data stakeholders, this includes not allowing any direct linking,

[16] https://www.interoperablemobility.org/

direct data reads, direct database access of another data store, or any use of shared memory. The TDEI restricts subsystem services to communicate only via service API calls over the network.

Full Capabilities Programmatic Access: All development partners must provide means of programmatic access equivalent to any actions that human users can perform by means of a graphical user interface or direct data store access.

Human Readable Open Standard and Programmatic Access Documentation: all open standards and APIs will be documented via a format that:

- Is published in its entirety on a publicly accessible webpage in human-readable form.
- Is documented in a language-neutral machine-readable format at a permalink in a format applicable to the following categories of schemas:
  - API: OpenAPI
  - Tabular data schemas (i.e., csv): frictionless data table, data resource or data package
  - Tagged data schemas: json-schema. TDEI deployment data specification extensions, use JSON:API specification to describe data models, describe how a client would be requesting data resources (fetching or modification requests), and how a server needs to respond to such requests. For more information on the JSON:API specification, please see https://jsonapi.org/.
  - Uses structured releases, versions, or changelogs.
  - All service APIs must be designed to be externalizable. That is to say, the TDEI development partners must plan and design to be able to expose the interface to developers in the outside world.[17]

For additional overall general guidance in other TDEI API governance and design questions, the TDEI team has found that it likes material on RESTful API design published by the Bank of Belgium.[18] This site provides excellent links to a wide variety of sources that provide excellent guidance to possible issues that are likely to arise in the design of microservice APIs to be constructed for use in the TDEI system. For example, the services that support interchangeable data infrastructure or externalization of transportation data. Use of guidance like this will ensure proper API design and reduce development risk.

[17] Amazon, who also uses microservices extensively also has a famous mandate from its Founder Jeff Bezos that states this same thought. https://nordicapis.com/the-bezos-api-mandate-amazons-manifesto-for-externalization/

[18] https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki

Table 7 Traceability for governance of TDEI microservice development and operations

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.1.1	Data Description	F-DE-01	Data standards shall use attributes that support travel preferences of travelers.	Inspection (covered by the adherence to Interoperability Principles, Open Standard community support criteria)	UN-AD2, UN-DU8,
3.1.1	Data Description	F-DE-02	The TDEI system shall utilize a common data model.	Inspection	UN-TS2,
3.1.1	Data Description	F-DE-03	The TDEI system shall provide unambiguous guidance/guidelines for its participants.	Inspection	UN-TS2b, UN-DG3, UN-AD1,
3.1.1	Data Description	F-DE-03.01	Guidance/guidelines shall be provided through data standard specifications.	Inspection	UN-DG3, UN-DG8, UN-AD1,
3.1.1	Data Description	F-DE-03.02	Guidance/guidelines shall be provided through data standard schemas.	Inspection	UN-DG3,
3.1.1	Data Description	F-DE-03.03	Guidance/guidelines shall be provided through coding instructions.	Inspection	UN-DG3,
3.1.1	Data Description	F-DE-03.04	Guidance/guidelines shall cover generating data in approved formats.	Inspection	UN-DG3,
3.1.1	Data Description	F-DE-03.05	Guidance/guidelines shall cover quality assurance requirements of the data.	Inspection	UN-DG8,
3.1.1	Data Description	F-DE-03.06	Guidance/guidelines shall cover accessing data.	Inspection	UN-AD1,
3.1.1	Data Description	F-DE-04	Data standard specifications shall be publicly available.	Inspection	UN-DG4, UN-DG6, UN-AD1a,
3.1.1	Data Description	F-DE-04.01	Data standard specifications shall include OpenSidewalks, GTFS-Flex, and GTFS-Pathways.	Inspection	UN-DG4a, UN-DG6, UN-TS7, UN-TS8,
3.1.1	Data Description	F-DE-04.02	Data standard specifications shall be published.	Inspection	UN-DG4, UN-AD1a,
3.1.1	Data Description	F-DE-04.03	Data standard specifications shall be version-tracked.	Inspection	UN-DG4, UN-AD1a,
3.1.1	Data Description	F-DE-04.04	Data standard specifications shall be vetted.	Inspection (covered by the adherence to Interoperability Principles, Open Standard criteria)	UN-DG4, UN-AD1a,
3.1.1	Data Description	F-DE-04.05	Data standard specifications shall include a data dictionary.	Inspection	UN-DG6,
3.1.1	Data Description	F-DE-04.06	Data standard specifications shall contain standardized metadata.	Inspection	UN-DG5, UN-TS5a,
3.1.1	Data Description	F-DE-04.06.01	Metadata shall describe the origin of collected data.	Inspection	UN-DS8,
3.1.1	Data Description	F-DE-04.06.02	Metadata shall indicate metrics for reviewers to determine the level of accuracy/completeness.	Inspection	UN-AD11,
3.1.1	Data Description	F-DE-04.06.03	Metadata shall describe the data standards and structure.	Inspection	UN-DG5, UN-TS5a,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.1.1	Data Description	F-DE-04.07	Data standard specifications shall include governance provisions that allow for effective management of data updates and revisions.	Inspection	UN-DG4, UN-TS3, UN-TS5, UN-DS1,
3.1.1	Data Description	F-DE-04.08	Data standard specifications shall include specified allowable values and error tolerance levels for data standard elements and attributes, where applicable.	Inspection	UN-DG4b, UN-AD1c,
3.2.2	Durability	P-DU-04	The TDEI system and all affiliated tools shall be capable of operating in isolation from other components with reductions in features.	Analysis	UN-DU10,
3.1.1	Data Description	F-DE-06.01	Data standard schemas shall include information about the database structure and database metadata.	Inspection	UN-DG5,
3.1.2	Data Extensibility				
3.1.2	Data Extensibility	F-EX-01	Updates to the data schema structure shall follow a formal update process.	Inspection	UN-DG4,
3.1.2	Data Extensibility	F-EX-02	Notifications shall be provided to approved TDEI system users when data schema updates occur.	Demonstration	UN-DG4,

2.2.4 Intermediary API Gateway Layers Help Integrate APIs

Material taken from IBM's web site on microservices describe API Gateways as follows[19]:

> Microservices often communicate via API, especially when first establishing state. While it's true that clients and services can communicate with one another directly, API gateways are often a useful intermediary layer. The API gateway is the entry point for clients. Instead of calling services directly, clients call the API gateway, which forwards the call to the appropriate services on the back end.
>
> API Gateways grow in importance as the number of services in an application grows over time. An API gateway acts as a reverse proxy for clients by routing requests, fanning out requests across multiple services, and providing additional security and authentication."

There are multiple technologies that can be used to implement API gateways, including API management platforms, but if the microservices architecture is being implemented using

[19] https://www.ibm.com/topics/microservices - note that minor text editing has been performed on the IBM material to make it more directly applicable to the TDEI specific material in this chapter.

containers and Kubernetes, the gateway is typically implemented using Ingress or, more recently, Istio.

The main function of the API Gateway is to prevent attacks by inspecting the messages passing through the gateway. Functionalities that will be implemented via the API Gateway include API firewalling, content validation and message integrity checks which are in place to only allow legitimate messages to enter the TDEI data infrastructure.

The API Gateway's *Content validation* will ensure that the requests made against the TDEI microservices API are appropriate. Content validation will check that the incoming request and data payload contain the appropriate parameters and values and that the payload adheres to the TDEI Data schemas. The API Gateway Content Validation will essentially be an API wrapper to engulf multiple APIs that help TDEI combine various calls that help access a set of related functions, rather than constructing multiple HTTP API requests from scratch. This will broaden the idiomatic ways of accessing and manipulating TDEI data. With an API wrap, TDEI will not need to fetch any information from another API when making a call. The specific functions for content validation will be encapsulated into a single package. Whereas the wrapper structure for content validation may be off the shelf, the actual data schema validation microservices will be specifically created by the TDEI, as described in section 3.2.8.1.

The Gateway's *Message Integrity Check* will verify the integrity of the signed message (signed tokens, headers, payloads) to confirm that the message has not been tampered with prior to the API call. In addition, it can ensure that some aspects of the payload remain confidential by encryption or other techniques. Using Message Integrity Checking, the Gateway can act as an enforcement point which can delegate to the TDEI API Call Validation the decision as to whether the call itself passes TDEI governance structure and can delegate to a third-party validation service whether the message identity and intent are good or bad (i.e., call ICAP server, PingIntelligence, etc.). The Gateway will enforce the decision from the third-party system.

Finally, since the API Gateways encounter all inbound traffic, everything can be logged. This allows for important *System Monitoring functionality,* increasing the ability for TDEI to have visibility, reporting and analytics over the use of the TDEI infrastructure. The API Gateway will allow external monitoring of the status of the TDEI APIs that are known and governed and also to highlight any traffic which is not governed. In this way, API Gateway provides visibility and insights for API consumers and providers. With sufficient time and resources, the TDEI will be able to provide usage reports to API Consumers and to the API Provider so that they can see the traffic and trends related to their API and applications. It can also provide detailed traffic logs for the API Provider to help with debugging of internal infrastructural issues.

2.2.4.1 Justification for API Gateway Layers

Justification for using an API gateway includes:

- API Gateways help decouple clients from services. Services can be versioned or refactored without needing to update all of the clients.
- Services can use messaging protocols that are not web friendly.
- The API Gateway can perform other functions such as authentication, logging, secure sockets layer (SSL) termination, and load balancing.

- Out-of-the-box policies, like those for throttling, caching, transformation, or validation do not have to be bundled into other services.

Table 8 Traceability for adding API Gateway layers to TDEI microservices

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.3	System Performance	PER-02.10	The TDEI system's demonstration applications shall be operational 99.5% of the time 24 hours a day, 365 days per year.	Analysis	UN-DU2,
3.3	System Performance	PER-02.11	The TDEI system's demonstration applications shall fulfill users' data requests and provide information within 15 seconds of a query.	Test	UN-DU2,
3.3	System Performance	PER-03	The TDEI system shall support performance tracking.	Demonstration	UN-DU3,
3.4	System Security and Privacy	SEC-01	The TDEI system shall include user permissions that ensure the safe and secure transmission of data and metadata.	Inspection	UN-DG2, UN-TS1, UN-TS4,
3.4	System Security and Privacy	SEC-08	The system design of the central database shall include redundancy and encrypted data archiving to ensure the continued operation of the system if major failures of or attacks on the system occur.	Inspection	UN-DU11,
3.5	Information Management	MAN-01	The TDEI system shall encrypt all system communications that travel over public data links.	Demonstration	UN-DU1,
3.1.1	Data Description				

2.2.4.2 TDEI Governance in Using API Gateways

In the context of the TDEI system and the system requirements of our specific instantiation, API Gateways provide enforcement capabilities that allow us to adhere to the requirements set forth in our Systems Requirements (driven by the multiple stakeholder partners). For instance, through gateways, we will be able to enforce that only trusted messages (authentication and authorization) can pass through to the APIs. Gateways will provide multiple ways for API consumers to authenticate and get access to API resources. Gateways can support any one of the many open standards that can be used to determine the validity of an API Consumer (i.e., OAuth, JWT tokens, API Key, HTTP Basic/Digest, SAML, etc.) which we foresee being used by the TDEI for authentication purposes. While we do not foresee the following flexibility being necessary for our instance, Gateways can also be used for non-standard means to locate credentials in headers or payload of the message. This increased flexibility ensures that we do not have to backtrack if we find it necessary for messages to self-authenticate.

In addition to authentication and authorization functions, the use of API Gateways is specifically important in the TDEI context because they can inject additional metadata information into the message about the original API Consumer. In addition to the typically exchanged metadata (IP address, roles, attributes, claims, etc.), we foresee this being especially important in the case of data requests and data updates, where data Producers using the API may be asked to provide meta data like the digitally signed data verification certificates they received after using the TDEI data vetting tools to validate the data they are contributing to the data repository. This aligns particularly well with the requirements around the presentation of “best available data” in the TDEI. This technical capability will help the TDEI enforce data provenance guidance around the identity information of the data Producers and the history of the data that is flowing into the TDEI repository, so that downstream data consumers have context about the data production and data context. Presently, the TDEI is not tied to a specific meta data structure, but identity and provenance propagation may be in the format of a new JWT claim or SAML token or simply inserted into the payload (like the digital verification signature).

Additionally, use of Gateways enables the TDEI to call out to other systems or services to determine authenticity, validation, and certification. This is important for extensibility and sustainability of the TDEI system, for example, if eventually some of the data schemas the TDEI stores will be federated by other organizations that will have data vetting services (for example, MobilityData federating use of GTFS-Flex v2). This is also a useful capability in the context of invoking some of the off-the shelf microservices we will likely be using, as described in section 2.2.3. For example, it could call out to an external Identity Provider or authorization system. Similarly, a customs agent might check an individual’s information against a known database of Producers.

Table 9 TDEI-specific governance using API Gateways traces back to associated System Requirements.

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.2.2	Durability	P-DU-04	The TDEI system and all affiliated tools shall be capable of operating in isolation from other components with reductions in features.	Analysis	UN-DU10,
3.3	System Performance	PER-02.09	The TDEI system shall fulfill application developers' data requests and provide approved information within 15 seconds of a query.	Test	UN-DU2,
3.4	System Security and Privacy	SEC-02	The TDEI system shall include procedures that ensure the safe and secure transmission of data and metadata.	Inspection	UN-DG2, UN-TS1,
3.4	System Security and Privacy	SEC-05	The TDEI system shall ensure that IT policies and safeguards are consistently up to date to reduce unauthorized access to routing request data.	Inspection	UN-AD13, UN-DU6,
3.4	System Security and Privacy	SEC-06	The TDEI system shall make efforts to ensure that the overall security of the data lake or repository are not compromised.	Inspection	UN-DU11,
3.4	System Security and Privacy	SEC-07	The TDEI system shall include an audit/reporting system that routinely scans for security risks.	Inspection	UN-DU11,
3.5	Information Management	MAN-02	The TDEI system shall contain different access levels (e.g., open and private), with defined user roles, to prevent unauthorized access of data and provide protection for sensitive private data.	Inspection	UN-DU1, UN-TS4,

SyRS Section #	Requirement Type	Requirement ID	Requirement Text	Verification Method	User Need
3.5	Information Management	MAN-03	The TDEI system's demonstration applications shall not share user account information with the processing or data repository components.	Demonstration	UN-AD13,
3.7	Policy and Regulation	POL-01.01	System-specific cybersecurity policies shall be implemented to protect restricted datasets from unauthorized access.	Inspection	UN-DS1a,

2.3 Integration Architecture

In this section, we describe an end-to-end approach to utilizing the three enabling technologies named above. In the first subsection below, we describe how the technologies integrate into the functional requirements of the TDEI. In the second subsection, we provide an end-to-end approach to utilizing all three enabling technologies in a workflow that uses imagery input data, run through a computer vision pipeline, in a workflow that traces from the Data Service Providers to the TDEI data store and back for model retraining, when applicable.

2.3.1 Component Integration

Figure 3 (repeated from its introduction in the discussion of microservices architecture) provides a high-level view of component integration--a functional view of the microservices architecture, APIs, and event buses and how they might interact in the context of the TDEI. The overall composition lends agility and scale to the architecture. The image is subdivided into seven functional panels, with each panel having white vertical text in the upper right corner of the panel that describes the overall function of that panel.

The first panel is labeled “variable inputs.” We anticipate handling a large variety of inputs in the eventual implementation of the TDEI interoperable infrastructure. The conceivable data streams we may eventually handle are highly varied, ranging from the sidewalk and transit data of specific interest in the ITS4US project, but potentially expanding in the future to include weather data (are the sidewalks covered in snow), incident data (are the buses being re-routed), and a variety of potential smart-city sensors that at some point in the future might be valuable inputs to a traveler’s navigation decision.

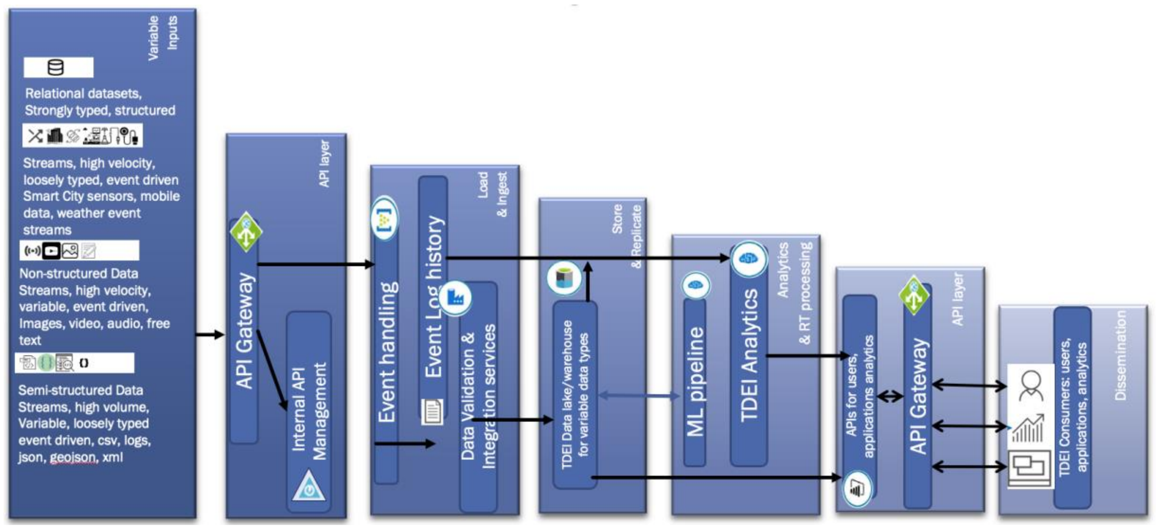

Variable Inputs
Relational datasets, Strongly typed, structured
Streams, high velocity, loosely typed, event driven Smart City sensors, mobile data, weather event streams
Non-structured Data Streams, high velocity, variable, event driven, Images, video, audio, free text
Semi-structured Data Streams, high volume, Variable, loosely typed event driven, csv, logs, json, geojson, xml
API layer
API Gateway
Internal API Management
Load & Ingest
Event handling
Event Log history
Data Validation & Integration services
Store & Replicate
TDEI Data lake/warehouse for variable data types
Analytics & RT processing
ML pipeline
TDEI Analytics
API layer
APIs for users, applications analytics
API Gateway
Dissemination
TDEI Consumers: users, applications, analytics

Figure 3 Component Integration: Showing how microservices, APIs, and event buses interact in the TDEI.

Within each panel are text-labeled elements, for example “API Gateways” are one of the elements in the second panel, “API layer.”. The elements are intended to represent a group of microservices that, when considered together, provide the TDEI with the functionality described in the element’s text. So, “API Gateway” is a group of different smaller applications that together function as an API Gateway, with the specific features that were discussed in Section 2.2.4, Intermediary API Gateway Layers Help Integrate APIs.

The specific functional elements described in Figure 3 include the following:

- “API Gateway” “APIs for users, applications and analytics” and “Internal API Management” all serve as functional units in the “API Layer” Panel. A detailed view of the expansion of this API Layer is provided within the Technology Readiness Level discussion, please consult Section 3.5.3. Based on the Technology Readiness Framework introduced by the FHWA Technology Readiness Level Guidebook, we followed the procedure above and conclude that message brokerage technologies are at readiness level 5.

  “Event Handling”, “Event Log History” and “Data Ingestion and Integration Services” all serve as functional elements within the “Load and Ingest” Panel.

  “TDEI data Lake/Warehouse for variable data types” comprises managed storage for multiple data types under the “Storage and Replication” Panel.

  “ML Pipelines” and “TDEI Analytics” provide processing units under the “Analytics and Real Time Processing” Panel.

2.3.2 Sample Integration and an Image Data-Stream Ingestion Example

To exemplify our eventual TDEI data architecture and microservices orchestration, we provide an example workflow for our humans-in-the-loop, mostly automated, smart OpenSidewalks data ingestion pipeline. The pipeline demonstrates use of computer vision algorithms in a microservices context to analyze data, microservices and event streams to run and trigger system activities, and we use decoupled services to leave room for improving processes over time. Computer vision is an increasingly popular smart city application used in safety, quality assurance and asset monitoring applications. In this example, 2-D street-level imagery is assumed to be the input (for example, cameras are attached to trash collection trucks that are pointed at pedestrian environments). The data serves as input to a machine learning model. The model makes calculations and inferences, returning output that can be used for creation and update of OpenSidewalks data as well as for troubleshooting assets in the built environment.

This example architecture shows an end-to-end approach to computer vision from the edge to the cloud and back. While this architecture is reaching beyond the end point of the ITS4US Deployment Project, it offers a vision of how the architecture we are building in this deployment project and the associated demonstration projects can be operationalized and scaled to sustain and maintain the data collected and pipelines created under this ITS4US Deployment Project.

The example architecture below is divided into operational areas:

The first operation area consists of microservices to operationalize real time processing as well as load and ingest. Our machine learning operations are part of the processing microservices. This architecture reflects a best practice to productionize machine learning. These microservices automate the process of using computer vision models for analyzing street-level imagery and producing OpenSidewalks data schema-compliant data. The key to this pipeline is a tight coordination of the microservices and the event handling,

The second operation area displayed here offers a data life cycle management approach based on DevOps techniques.

The third operational area display here is event handling and notification. This example architecture describes a human-in-the-loop approach, in which people are notified to intervene at certain steps in the data conflation and vetting. Their interventions become part of the intelligence captured by the models, creating a continuous cycle of training, testing, tuning, and validating the machine learning algorithms.

Figure 4 is a TDEI specific version of a sample Microsoft architecture diagram. The data flow envisioned in this figure as applied to the TDEI is as follows:

(Step 1) TDEI Image/Batch Data/Mobile Data Processing Application consists of a microservice that gathers data from the edge device (an image data stream) and an associated TDEI microservice to analyze that data. On the edge device, it captures the live image stream, breaks it down into frames, and calls the TDEI service that performs inference on the image data to extract the OpenSidewalks data schema.

(Step 2) TDEI raw media storage allows for upload and storage of raw imagery data files in the TDEI raw media store. These files will be used for training and testing purposes if the data producer (for example, the city of Seattle) allows the TDEI to make use of imagery in this way. If data is not allowed to be used in this way, the imagery is discarded and only the OpenSidewalks extracted data persists.

(Step 3) Extracted OpenSidewalks data represents the inference results and metadata captured by the TDEI computer vision analyzer microservice on the edge module. Inference results are sent to the TDEI Hub that is an event hub that publishes this data entry to a topic. The TDEI Hub acts as a central message hub for communications with other microservices.

(Step 4) TDEI Integration Server listens to the TDEI topics for messages about data input events.

(Step 5) The Integration Server routes inferencing results and metadata to the TDEI Data Lake/Warehouse for storage. The Integration Server also publishes to a topic in the TDEI event stream all the changes committed to the OpenSidewalks graph for that region. The TDEI event streams are designed to provide a full history and provenance of the data in the TDEI shared data. All data entries are logged (in the TDEI event log) and the up-to-date data can be fully traced through the event stream.

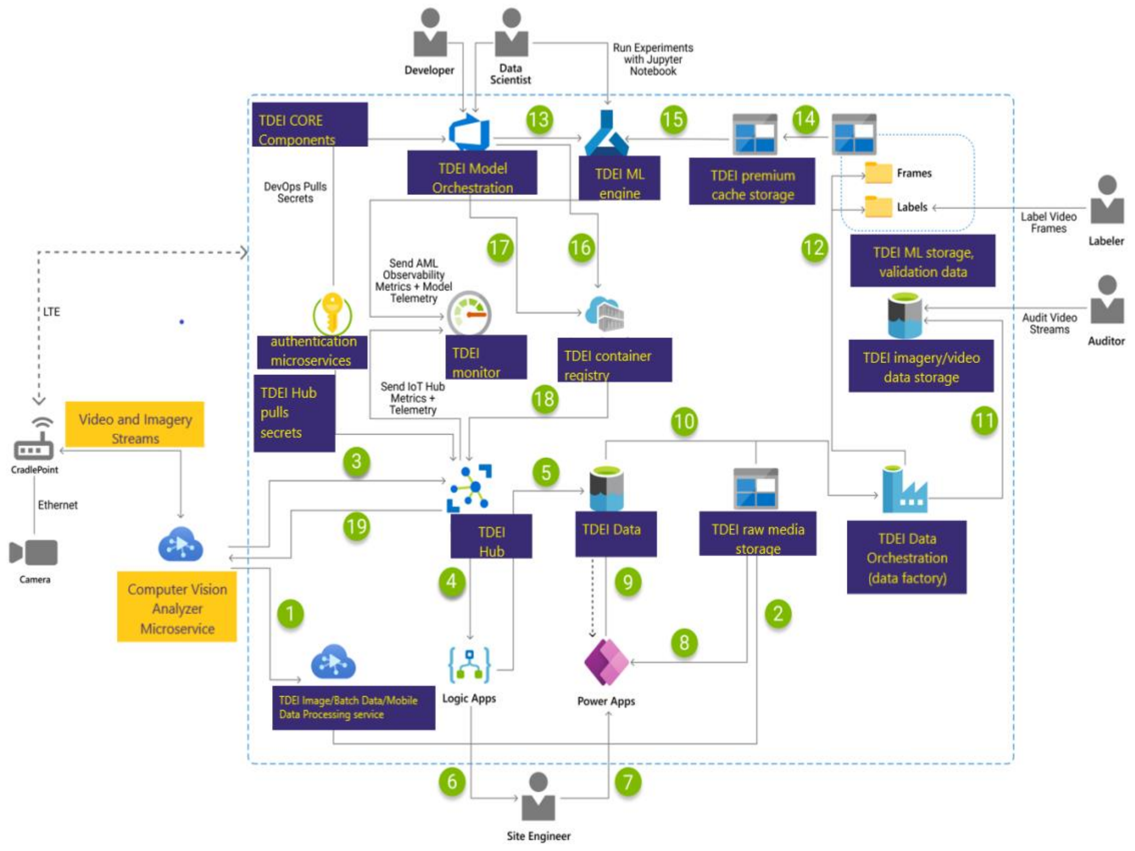
(Step 6) The Integration Service routes any conflation problems (for instance, data just entered through the inference module is identified to conflict with the current OpenSidewalks data for a particular sidewalk asset) to a human in the loop. The Integration Server publishes to a topic which is then listened to by the TDEI Registration module. The TDEI Registration module identifies the entity that produced the data (city of Seattle) and accesses the e-mail information for the human identified as the person to notify in the event of data conflicts. The person is e-mailed a notification asking them to assist in conflict resolution.

(Step 7) The notified individual (identified in the image as a site engineer) opens a TDEI client application (for example, a Vespucci client used in the Common Paths application) to acknowledge and resolve the conflict. The TDEI Power Apps deployment is the server-side microservice to listen for the conflict resolution event and trigger (by publishing to topics) the appropriate downstream TDEI response to the conflict resolution effort performed by the site engineer.

(Step 8) The TDEI Power Apps may also be asked (by the site engineer) to provide more context for the conflict resolution, including having to pull inferencing results or any metadata from the TDEI Data Lake, or the raw image files (if available) to display the relevant information about the data conflict.

(Step 9) TDEI Power Apps updates TDEI Data Lake with the conflict resolution provided by the site engineer. This step provides for human-in-the-loop input for enhancing validation, which allows for model retraining. The TDEI Power Apps also publishes anything to the Event Handler topics to trigger any further downstream TDEI response to the conflict resolution effort performed by the site engineer.

(Step 10) TDEI Data Orchestration (Data Factory) microservice is the data orchestrator that fetches raw imagery files from the TDEI raw media storage together with the corresponding inferencing results and metadata from TDEI Data Lake/ Warehouse.



Developer
Data Scientist
Run Experiments with Jupyter Notebook
TDEI CORE Components
TDEI Model Orchestration
TDEI ML engine
TDEI premium cache storage
Frames
Labels
Label Video Frames
Labeler
DevOps Pulls Secrets
TDEI ML storage, validation data
Audit Video Streams
Auditor
LTE
Send AML Observability Metrics + Model Telemetry
authentication microservices
TDEI monitor
TDEI container registry
TDEI imagery/video data storage
TDEI Hub pulls secrets
Send IoT Hub Metrics + Telemetry
Video and Imagery Streams
CradlePoint
Ethernet
Camera
Computer Vision Analyzer Microservice
TDEI Hub
TDEI Data
TDEI raw media storage
TDEI Data Orchestration (data factory)
TDEI Image/Batch Data/Mobile Data Processing service
Logic Apps
Power Apps
Site Engineer
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

Figure 4 Sample architecture[20] describing human-in-the-loop approach to ingesting, creating, and maintaining sidewalk data in TDEI interoperable data sharing infrastructure

20 https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/ai/end-to-end-smart-factory

(Step 11) If allowable, the TDEI Data Orchestration might also store the raw imagery data files, along with the TDEI metadata, in the TDEI Data Lake/Warehouse (in other databases than those devoted to OpenSidewalks data). The Lake serves as an archive for auditing purposes, if allowed by the data producer.

Note: the remaining steps described in the data flow are not part of the SyRS or requirements defined for the UW ITS4US/TDEI Deployment Project. We are interested in demonstrating this integration to make sure that the architecture we build under this deployment project will provide us the capability of extending and scaling to these capabilities in the future.

(Step 12) If allowable by the data producer, the TDEI Data Orchestrator would provide street-level 2D image frames. It could then use the machine learning engine (TDEI ML engine) to infer results (e.g., sidewalk characteristics) from those results a generate infrastructure labels. It would then upload the results into the TDEI machine learning storage (the higher latency, less accessible machine learning data store for future model training and testing).

(Step 13) The TDEI ML engine listens to a topic specific to posted changes in the training dataset. The previous step is published to an event stream topic. That topic is listened to by the TDEI DevOps model orchestration microservices which triggers downstream training, testing and validation processes.

(Step 14) Changes to the model also post to an event stream. That stream is also listened to by the TDEI ML engine. This step would be an alternative route to the model training, along with a possible manual trigger, all of which lead the TDEI ML Engine to perform machine learning model training and validation processes.

(Step 15) TDEI ML engine starts training the model by validating the data from the TDEI ML data storage. TDEI ML engine then uses that dataset to train the model. It also can validate the trained model's performance. Finally, it can score testing data against the newly trained model. The TDEI ML engine thus evaluates the performance of the newly trained ML model and determines if the new model is better than previously trained models. If the newly trained model is better, the ML engine builds a new version of the TDEI image/batch data/mobile data processing service as another TDEI data tenant application.

(Step 16) Assuming a new TDEI data tenant application was created, the TDEI ML engine registers the new model into the TDEI Application Registry. In effect, the TDEI ML engine registers the new inference model microservice as an external data tenant application from which the TDEI will accept API calls to push data to the TDEI (much like the TDEI image/batch data/mobile data processing service that was previously used in Step 1 to process images delivered from the data producer’s image collecting engine, process those images, and push the data to the TDEI). By being a TDEI registered application, data posted to the TDEI API asserts that the data is coming from an authorized data producer application. An alternative would come from other authorized registered applications, like the Common Path application, which is another planned data tenant-registered application for pushing OpenSidewalks data.

(Step 17) The TDEI API Gateway reviews the application that the TDEI ML engine tried to register as an authorized TDEI data producer application in the TDEI Application Registry.

(Step 18) If the application passes the API Gateway review, it is registered via the TDEI Application Registry which publishes the new registrant to a topic. That topic is published to the TDEI Event Log.

(Step 19) The message published to the event log triggers the TDEI Hub to replace the old data processing service with the new. The application is containerized and pointed to by the API call for image processing. The TDEI Hub also reaches out to the data producer (City of Seattle) to inform the camera on the trash collection truck that a new version of the application is now available, but no other changes need to take place on the edge device.

2.3.2.1 Procurement for the Sample Integration

In this section, we describe a possible procurement scenario for this integration example. Given the complexity of the example, we have only investigated actuating this on the Azure Cloud platform, which is the cloud platform we currently use for all OpenSidewalks and AccessMap development. As a result, the following discussion references specific Microsoft Azure services which are described in the company's technical literature.[21]

While use of the platform is not currently free, all our code is and will be open-source and free to use. Currently, our code is not cloud-platform dependent. Some of the components described below will tie the TDEI to the Microsoft platform, which the TDEI will attempt to avoid. We foresee that a viable development path is to first implement the infrastructure as a working implementation in the Microsoft Azure platform and then progressively identify or build non-platform dependent microservices to replace the Azure-specific modules.

The following components will be used to implement this architecture:

- TCAT's Computer vision pipeline for street-level imagery enables developers to quickly build an artificial intelligence (AI)-powered image analytic solutions on the edge to extract viable OpenSidewalks data from images, whether stored or streaming. The publication, Zhang, Yuxiang, Sachin Mehta, and Anat Caspi. "Collecting Sidewalk Network Data at Scale for Accessible Pedestrian Travel." *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, 2021 describes the technique.
- The TDEI Event Hub can be implemented using the Azure IoT Hub. This would serve as the central message hub for communications in both directions between external applications, data streams, attached devices, and the TDEI infrastructure.
- The TDEI ML engine can be deployed with the Azure Machine Learning module, capable of building, training, deploying, and managing ML models in a cloud-based environment.
- The TDEI data storage and TDEI raw media storage can both be implemented via Microsoft Dataverse, the cloud-based storage platform used by Power Apps to support

[21] https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/ai/end-to-end-smart-factory

human-in-the-loop notifications and to store variable data (such as meta data and other data schemas) associated with other components used by the data pipeline.

- TDEI Application Registry as well as the TDEI Container Registry can be both implemented using the Azure Container Registry (both are needed for orchestration in our architecture, but the TDEI Application Registry was not shown in Figure 4.) Azure Container Registry creates and manages the Docker registry. Container Registry builds, stores, and manages Docker container images, including containerized machine learning models. Currently our computer vision models are not containerized.
- The TDEI ML storage can be implemented via Azure Blob Storage. This storage service provides a local ML data store and data cache for when one is needed for training the ML model.
- The TDEI Data Lake/warehouse can be implemented with the Azure Data Lake Storage Gen 2, which provides a low-cost, tiered storage on top of Azure Blob Storage. In our example, it provides the archival street-level image store for the raw image files and metadata.
- The TDEI Integration Server will have a lot of TDEI-specific logic, but the component itself can use the infrastructure of Azure Logic Apps to create and run the automated notification workflow that sends SMS and email alerts to the site engineers (a communication component that we have not yet implemented in the integration server).
- The TDEI Analytics engine can be implemented via Azure Monitor which collects telemetry from Azure resources in order to proactively identify problems and maximize system performance and reliability.
- The TDEI Data Orchestration module can be implemented via Azure Data Factory. This is an Extract, Transform, Load (ETL) pipeline and data integration service that allows us to create fast data-driven workflows for orchestrating data movement and transforming data at scale. In our running example it orchestrates the data used in the ML process in an ETL pipeline to the inferencing data, which then stores it for use in retraining the ML model. In our original functional component depiction in Figure 2, almost the entirety of the panel named “Load and Ingest” can be implemented with the use of this component.
- The Power Apps can remain decoupled apps, services, and connectors that the TDEI will custom build, along with a data platform. This will free us to move off the Azure paid tier once the heavy lifting components (all those mentioned above) are replaced.
- Finally, although not mentioned in the example above, a CI/CD (continuous integration and continuous deployment) pipeline is a good investment for us to be able to scale and swap out applications without impacting TDEI data producers or consumers. Azure DevOps can provide this team-based developer service functionality. In our example, it will displace some of the functionality we described in the TDEI ML Engine in that it can take over triggering the ML Engine when it learns from new data. (This component will do so with serverless tasks.) It can take over the ML model comparison and the new builds of the inferencing service container application on the edge. It has these additional capabilities not originally scoped for the TDEI because it features Azure Pipelines for creating continuous integration (CI) and continuous deployment (CD) pipelines.

2.3.2.2 Integration Alternatives and Considerations

In this section, some alternative implementations still within the Azure cloud infrastructure are listed.

Model orchestration can also be done using Azure DevOps, which has the benefit of being closely tied to the model code. The training pipeline can be triggered easily with code changes and through a standard CI/CD process. The TDEI has not explored all of the possible options in this approach.

Model orchestration can also be done using Azure Data Factory. The benefit of this approach is that each Data Factory pipeline can provision the required compute resources. One concern is that Data Factory doesn't hold on to the Azure DevOps agents to run ML training. This might congest the normal CI/CD flow.

Instead of using the data pipeline to stream data from producers, and then break down the data push into separate image frames, one option would be to deploy an Azure Blob Storage module onto the TDEI data tenants that are producing image streams. Then the inferencing module can work on the device owned by the data tenant and only upload the inferred TDEI-compliant data to the TDEI. The TDEI will determine when to upload the frames directly to the ML data store. The advantage of this approach is that you remove a step from the data pipeline and potentially avoid conflict with organizations that do not want to share the raw imagery data. The tradeoff is that the data streaming devices at the TDEI data tenants are tightly coupled to Azure Blob Storage.

As part of the human-in-the-loop transactions, TDEI data tenants assign people to check and evaluate conflicts in data, as well as check and evaluate the results of machine learning predictions. Human expertise is captured and is used to validate the model downstream. If the model's results are inaccurate, the data is checked again, and the algorithms can be retrained.

Roles can be assigned to the humans intervening in this loop, including data labelers. (This applies to working with image data or 3D point cloud data, extracting information that applies to traveling through spaces and annotating it.) The resulting labeled data sets can be used for training and retraining algorithms that can automate the extraction of path information from imagery or 3D point cloud volumes.

The role of the Data Scientist (a TDEI personnel) is to use labeled data sets to train the algorithms to make predictions. Data scientists register, deploy, and manage models. Currently our data scientists use our own python infrastructure and although we use GitHub for versioning, we do not have automated solutions for continuous integration processes (processes that automatically trains and validate a model). In the integrated scenario, we would aim to make use of pipeline processes that allow for automated triggers. New training should be triggered when new data populates the dataset or when a change is made to the training scripts.

Data tenant engineers oversee tenant applications within their native institutions (like IT personnel at King County Metro paratransit). The tenant data applications run in containers and are registered with a Container Registry. Using a continuous deployment pipeline, they can deploy and scale the infrastructure on demand.

When the site engineers receive any conflict or incident notifications, they can manually validate the results or predictions of the machine learning model. For example, they might examine a road shoulder that the model predicted as a sidewalk and they would indicate that the algorithm had failed. This position would be called a conflict resolution engineer.

When questions arise about a model's predictions, safety and responsible AI auditors can review the archived input data streams (or those portions that made it to the TDEI storage) to detect anomalies, assess compliance, and confirm results. This position would be called a safety and responsible AI auditor.

Availability

Most of the components used in this example scenario are managed services that will automatically scale. The availability of the services used in this example varies by region.

According to the Microsoft Azure technical documentation,[22] apps based on machine learning typically require one set of resources for training and another for serving. Resources required for training generally don't need high availability, as live production requests don't directly hit these resources. Resources required for serving requests need to have high availability.

Monitoring

The TDEI analytics engine as well as the TDEI monitoring microservice provide metrics and quantitative assessments to enable TDEI monitoring to enable diagnosis, interrogation, and troubleshooting.

Scalability

The TDEI hopes to lay the foundation for long-term scalable interoperable shared mobility data architecture. Scalability applies to the data ingestion pipeline, where we hope that TDEI maximizes data movement by providing a highly performant, cost-effective solution.

Security

Access management mechanisms in the API Layer are designed to help ensure that only authorized users can access the environment, data, and reports. Storage is encrypted using customer-managed keys.

DevOps

DevOps practices are used to orchestrate the end-to-end approach for the TDEI infrastructure. If your organization is new to DevOps, the DevOps Checklist can help you get started. The Integration Server example and the Common Paths example are single-tenant projects that include a deployment and pipeline examples.

[22] https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/ai/end-to-end-smart-factory

3 Technology Readiness Level (TRL)

Using the framework documented in Section 2.1, we provide the TRL for each ET identified in Section 2.2 of this document. We demonstrate this individually for each ET below. We also considered the entire group of related ETs that are integrated together.

3.1 TRL Assessment Process

In this section, we provide the steps and resources we plan to use to follow the framework documented in Section 2. We describe only those areas of the enabling technologies that require specific investigation on the part of our TDEI team, current technical gaps and questions pointing to next steps in the technology's development that may be uncovered. Where appropriate, we consider the level of effort required to move the enabling technology from its current tech readiness level to deployment ready.

Within each subsection devoted to a specific enabling technology (ET), we will consider the following questions pertaining to the deployment of microservice architecture in the TDEI:

- What questions remain gaps in knowledge for the team in implementing and deploying this enabling technology?
- What are the evaluation steps to follow for each question?
- How will you evaluate the ET TRL in context of the conditions for your project and site?
- Which team members or roles will you engage in this investigation?
- How will you avoid potential bias of your group which could influence TRL results?
- How will you ensure the data you use for the TRL results is valid and current?
- Will your process require the reevaluation of the TRL results in a later time in the project to support future Phase 2 and 3 documents?

3.2 Microservice Architecture

3.2.1 What Questions Remain Gaps in Knowledge for the Team in Implementing and Deploying This Enabling Technology?

From the TDEI perspective, there are two primary considerations in choosing how to apply the microservices architecture, (a) evaluating the benefits/detriments of using specific languages for implementing and standardizing the microservices architecture and (b) taking careful consideration to designing the microservices, determining how they are decoupled and separated if each microservice is to truly have separate data access and view of the data stores, and whether we strictly adhere to 'no data sharing allowed' among microservices.

3.2.2 What Are the Evaluation Steps to Follow for Each Question?

3.2.2.1 TRA Question 1: Choosing a Language and Technology for Microservices

With microservices, we can build a reliable platform to extend and grow while taking advantage of diversity in languages. It's possible to use different technologies or languages for different services, but we will likely want to choose at most two, one to support our backend integration server infrastructure, which will have access to all the transportation data layers and graphs, and the second to support all the externally facing APIs. The microservices architecture can result in additional operational overhead, since microservices are often running on other machines and require a network hop between your services.[23] This can slow down the whole system considerably. The choice and diversity of programming languages used in the system can also increase that performance overhead. Consequently, this task requires (1) coming up with a coherent assessment criteria and path for comparing and contrasting the choices, and (2) performing the comparison. In the least, the evaluation should include some of the most common languages and their attributes: Java (along with its many microservices extensions), Golang, Python, Node JS, and .Net. Please see the Technology Readiness Level Assessment in Section 3.5.1 TRA: Microservice Architecture I which we used industry sources to evaluate the languages named here.

The goals of microservices lead us closer to serverless architecture. It helps limit the degree to which data and services are integrated. This in turn can help limit limiting the growth in required compute time as both the data and size of the TDEI increases.[24] Microservices are also particularly useful for large, complex systems, which require the ability to frequently add or change system capabilities or features in a rapid, reliable manner. It separates complex systems into a more granular and modular design, allowing modules to be replaced, when necessary, with limited impacts on other modules in the system.[25] But the choice of language(s) will impact the degree to which we gain value from the use of the microservice architecture.

3.2.2.2 TRA Question 2: Architecting Separate TDEI Microservices, How They Are Decoupled, What Resources They Are Allocated and How They Interact

This, indeed, is the heaviest lift and uncertainty we currently have with the TDEI.

[23] https://adamdrake.com/enough-with-the-microservices.html

[24] IView Labs – 9 Key Points to Decide on Microservices Architecture - https://iviewlabs.medium.com/9-key-points-to-decide-on-microservices-architecture-c390d9827db7

[25] Microservice Architecture - https://microservices.io/

One common concern with microservice architecture is that it is difficult to define appropriate and effective boundaries between the microservices as the system is developed. The TDEI team believes we have sufficient experience to perform this task. Our previous work with the data flow and system processing required in our earlier OpenSidewalks applications give us the use cases needed to inform the starting points to begin outlining the microservices breakdown, at lease for the uses the TDEI wishes to prioritize and address. The additional use cases would serve as keeping us abreast of potential future issues, risks, and points of failure, and provide the backdrop to curate our approach and give us a deeper understanding of what tools and modularization would provide the best, most decoupled approach. Every technology decision depends on the tools we use to develop for data consumers or producers in other parts of the interoperable data sharing platform of the TDEI.

The goal is to have the TDEI powered by applications that are decoupled from one another. There are numerous downsides to failure, and our evaluation of our proposed microservices arrangement will attempt to avoid requiring real-time, consistent access to functionality or data managed by another service, [26] as maintaining clean segregation of services and passing data asynchronously ensures efficient service provision. We will also clearly define the middle layers or event stream topics that help services communicate.

The topic of messaging and separation of services bridges into the next enabling technology, but it is worth addressing the topic here in that there is interaction between the microservices used and the message brokers the TDEI may choose to use, and we should ensure that there is compatibility between the microservice architecture and popular open-source message brokers such as ActiveMQ, RabbitMQ, and other managed Apache Kafka services.

Finally, we will ensure that our microservices can connect into one or more event bus topics, publish new events and/or consume events. These actions, which can consist of simple notifications of actions, state changes, or other microservice dataset activities, need to take place sequentially.

Ultimately, the decisions will also depend on the current knowledge of our development team. Specifically, this effort will be led by the Data Management Architect and Lead, with support from the Technical Application Lead and the Deployment Development Lead.

3.2.3 How Will You Evaluate the ET TRL in Context of the Conditions for Your Project and Site?

We will spend one week detailing all our known use cases for the TDEI (for all three data standards) and apply priorities to these use cases to identify the use cases we wish to address first, but those that we are aware of must be considered in the planning and evaluation phases.

We will spend two to three additional weeks using those ranked use cases to identify a coherent evaluation criterion that clearly delineates our necessary features and desired attributes for (1)

[26] SHIFT Commerce's Journey: Deconstructing Monolithic Applications into Services - https://blog.heroku.com/monolithic-applications-into-services

the language and technology choices we make in the building of microservices and (2) the microservices build plan which will determine what microservices we build, how they are decoupled and how they interact within the context of the TDEI and in what order we build them.

We will spend five additional weeks executing and researching this comparison, evaluating the different options, and coming up with a plan and contingency plan.

3.2.4 Which Team Members or Roles Will You Engage in This Investigation?

Specifically, this effort will be led by the Data Management Architect and Lead, with support from the Deployment Development Lead and the Technical Application Lead.

3.2.5 How Will You Avoid Potential Bias of Your Group, Which Could Influence TRL Results?

We believe that using on-the-ground use cases and the real data we have had the privilege of having will help us avoid bias. We will also conduct a hypothetical discussion of how we may extend this infrastructure to support additional prioritized data schemas, such as supporting bicycle and micromobility data types.

3.2.6 How Will You Ensure the Data You Use for the TRL Results Are Valid and Current?

We are fortunate to be a trusted partner to organizations and institutions who have shared and continue to share data with us. They keep us up to date. Additionally, the device end-users are the best validity checks on our data. We maintain our data as up-to-date as we can in order to allow our application users to benefit from the use of the data.

3.2.7 Will Your Process Require the Reevaluation of the TRL Results in a Later Time in the Project to Support Future Phase 2 and 3 Documents?

Our deployment projects will provide the most informative evaluation and evidence of the results of this effort. In the sections below we discuss some of the initial explorations we have made to identify various off-the-shelf as well as TDEI-specific microservice infrastructure.

3.2.7.1 Sample Microservices – Off-the-Shelf and TDEI-Specific

It is important to understand that the integration example provided in Section 2.3.2 *Sample Integration and an Image Data-Stream Ingestion Example* can (and should) be coupled with a microservices architecture. In fact, it is assumed that microservices play a role, and even with the procurement example given, it is still important for the TDEI to architect separate microservices well, responding to the second TRA question above (Section 0) For the purposes of better understanding the TRL of Microservice deployment, we offer an exploration and a more nuanced view of some core microservices we may want to develop and those we may be able to easily procure and use off the shelf.

As noted, using microservices provides a distinctive method of developing software systems that focus on building single-function software modules, each of which has a well-defined mechanism for interfacing that function with other modules in order to achieve smooth and efficient operations. The additional advantage to using such architecture is that certain single-function modules can be used off-the-shelf (OTS) without adaptation. This will provide a significant advantage to our small, decentralized development team. Several single-function established microservices are expected to be used off-the-shelf, providing low-risk, high-returns established technological advantage.

3.2.7.2 OTS: Security, Identity, and Authentication Microservices

- Digital identity is at the core of any data application, but especially data provisioning - invisible yet crucial. Any Data Generators, Providers or Consumers will require a digital identity known to the TDEI, to represent entities interacting with it, and be associated with a certain level of access to data.
- Identity is complex. There’s username and password authentication, social connections, single sign on, and these are just ways to login. Multifactor authentication, breached password detection, anomaly detection, securing sensitive data like passwords, and many other topics comprise identity. These are outside our domain of expertise, but off the shelf solutions exist for this purpose. We have identified an open-source solution in OAuth.

3.2.7.3 OTS: Messaging TDEI Data Generators, Providers, Consumers

There are numerous GIS data platforms, tools, and sensors that transportation and municipal agencies use today to maintain GIS assets about pedestrian mobility. There are also many different mechanisms by which user organizations (whether data generators, providers or consumers) are communicating with other organizations. Given all these data channels and all these messaging channels, getting a consistent message to all the data stakeholders can be a challenge. Messaging is especially important in this instance where the TDEI will be creating the data infrastructure at the same time as the data specifications themselves are changing and extending. Moreover, some of our stakeholders (like CALACT, for instance) will require bi-directional communication and require engaging with organizations in conversation rather than just ‘broadcasting’.

In particular, a specific use case we foresee requiring this kind of messaging is in the case of two data producers having geographically overlapping jurisdictions where entries into the TDEI may identify conflicting information that only the data producers on the ground can resolve. An example of such a conflict that may arise is having OpenSidewalks data furnished by both the City of Seattle DOT as well as King County Metro, where the location for a particular bus stop may diverge between one producer and another. The TDEI may identify the conflict but is unable to adjudicate between the two versions and therefore must be able to message all the producers that overlap in that region that a data conflict needs to be resolved, potentially via on-site confirmation. The messaging interface would be utilized in that case to alert all parties involved. (This same example also appeared in the workflow example in Section 2.3.2)

Twilio, or Azure Power Apps and Azure Communication Services are all examples of platforms that provide messaging and communications API and services to handle SMS, voice, and other forms of communications to deliver consistent messaging across all channels. While a TDEI-specific instantiation would be appropriate for the TDEI, this technology is already in market, offering Technology Readiness Level of 5, per the FHWA Technology Readiness Level Guidebook.

3.2.7.4 OTS: Data Consumer and Producer Registries

- Microservices are inclusive of numerous ‘plugins.’ Although the previous integration into Azure Cloud example provided a single monolithic application of the Azure solution to the Extract, Transform, Load and Data Integration problem, there are additional alternatives we can use in addition to creating our own with instance Data Consumer and Producer Registries.
- To a degree, this is the traditional way to do Extract, Transform, Load (ETL) – data is transformed on ingest and outputted with new structure and schema to the data platform (this could be a database or data warehouse). In this case we’re treating all the input organizations as just another data source that provide organizational systems of record.
- There are a few aspects to these microservices that can be TDEI-specific architected to be responsive to the following system requirements:
- TDEI data microservices shall perform basic tasks of transforming GIS data to TDEI standard schema format (the specific formats that will be available will be refined based on partner inputs, for example, for now we are prepared to transform centerline and polygonal asset management data for sidewalks)
- TDEI data microservices shall account for unique characteristics of event streamed data from on-demand transit services.
- TDEI data microservices shall capture the type of data that lends itself to provenance information and metadata analysis – such as application logs, clickstream, and sensor data, which are frequently used in data science initiatives. (This will likely drive a further set of requirements as we are leaving this unspecified to accommodate versatile and yet unrealized data, for example, internet of things data streams from elevator infrastructure inside transit facilities).
- TDEI data microservices shall not be lossy even if not all the data provided necessarily fits into the TDEI data schemas, the streamed information will be preserved.
- Integration: TDEI data microservices can be identified in a microservice registry framework that uniquely describes these software connectors/adapters and that publishes this information to other services. This requirement is a child requirement to providing data provenance and accountability throughout the TDEI system. To be able to track data provenance successfully, it is necessary to identify which agents acted on a data record and what was the transformation performed on the data. In our case, the notion of an agent used here is extended to include our own software connectors/adapters that will do something to the data. It is thus necessary to provide a framework that uniquely identifies and describes this software service and makes this information accessible to other services in case data must be rebuilt and traced. The best approach for this is to store this information into registries. A registry is defined as a system for keeping an official list or record of items. In the case of the TDEI, we will use a registry database populated with information that can be updated and accessed. This concept is used in micro service-based architecture to define a service registry, which is a database populated with information about services.

- Several technology work products will be at play and developed here:

a. Providing the microservices that service the data producer community with the necessary Extraction, Transformation, and Loading tools.

b. Providing a registry database for the available services. A natural extension of this work, but which is outside the scope of work for this effort and is provided on top of this registry database, would be to build interfaces for services to automatically register into the registries and for clients to discover these services in an automated fashion.

c. Providing guidance and ability for the open-source community to contribute their own microservices and the documentation to enable others to register their contributed services and any type of meta-information their registry entries should contain.

3.2.8 TDEI-Developed Microservices

Some of the Data Life Cycle functions will be specific to the TDEI and require the development team and/or partnering development teams to build out these capabilities. Provided below are the fundamental services that will provide the functional capacities responsive to the system requirements of the TDEI.

3.2.8.1 Microservice: Data Validators for All TDEI Data Schemas

Data Schema validators will be featured as standalone microservices offered by the TDEI for each of the Data schemas stored by the infrastructure. (**)

3.2.8.2 Microservice: Data Collection Using Computer Vision Pipelines

Computer Vision Pipelines are used to extract OpenSidewalks data from imagery data that is provided by Data Service Providers. The extracted information and metadata are provided as an output in TDEI data conforming to TDEI data specifications.

Section 2.3.2 describes an end-to-end integration of the workflow of the TDEI provided some street-level imagery to ingest. The architecture discussion mentions a microservice application in the embodiment of the TDEI Image/Batch Data/Mobile Data Processing Services application that is used to collect TDEI-specification compliant data from imagery data provided by Data Service Providers. The presumption is that the DSP’s do not wish to share their imagery data, and that the computer vision pipelines are provided as a microservice application to the DSPs to run in-house, with the choice to run these algorithms without exposing any of the proprietary imagery data to the outside world. There is an option to share some sanitized imagery for validation purposes only.

The computer vision pipeline is functionalized as a microservice using machine learning (ML) built by the Taskar Center for Accessible Technology, used to analyze imagery data to extract vectorized travel path information. The TDEI project must still harden this pipeline, associate the pipeline with an API and API Gateway, and ensure that the microservice can run on remote systems. In the TDEI instance, the computer vision services are used for data collection and data updates. As the pipeline is encapsulated in a microservice that has an API, and an API gateway layer. The backend runs a machine learning model. The model makes calculations, predictions, and inferences, returning output that can be used for transportation layers in built environments or transit facilities (depending on which instance is discussed and the input data type). The pipeline either creates new sections of the transportation layer or can be used to update existing layers of the transportation layer. It is also responsible for ensuring compliance with the appropriate TDEI data schema. Once this pipeline produces new data, additional data cleaning, validation and verification operations are triggered, as was already described in Section 2.3.2.

The technology pipeline itself automates the process of using ML models for complex decision-making and is created as an open-source backend pipeline. The key to integrating the pipeline into the TDEI is to build additional services and layers (as described in Microservices Architecture: Enabling Data Collection, Aggregation, Integration and Transformation) to enable a full life cycle management approach to the ML pipeline based on DevOps techniques. This includes adding services that coordinate among the Data Service Provider (who provides the data) and the Taskar Center development team (who build, train, evaluate, and deploy the machine learning models).

Machine learning inference pipelines are run on images (either batch or real-time) from Data Service Providers (may consist of Microsoft, Google, or Coco Robotics). The ML models are run to infer a transportation layer data in the OpenSidewalks format. Cached imagery streams are optionally used for auditing purposes and to retrain the models. The machine learning pipeline itself depends on the specific use case for the TDEI. While out of scope for the ITS4US Deployment Project, we would like our current microservices and API breakouts to enable three use cases for data extraction and we will ensure that our implementation would be able to extend to handle these instances in the future:

OpenSidewalks data extraction from multi-viewpoint (mostly satellite) imagery: Computer Vision pipeline to collect OpenSidewalks data (vectorized data model for the built environment and right of way) using multiple image sources including- satellite imagery, oblique aerial imagery, map tiles, digital elevation model, mobile GPS traces (if available)

OpenSidewalks data extraction from Street-level viewpoint imagery: Computer Vision pipeline to collect OpenSidewalks data (vectorized data model for the built environment and right of way) using multiple image sources including- Street-level sidewalk imagery (for instance, collected by a sidewalk delivery robot), map tiles, digital elevation model.

GTFS-Pathways from LiDAR 3D point cloud imagery: Computer Vision pipeline to collect GTFS-Pathway's data (vectorized data model for transit facilities) using 3D point cloud or other LiDAR collection, along with any RGB-D or RGB data (if available)

Note: in Phase II and III of the ITS4US project, our architecture describes a human-in-the-loop approach, in which people are notified to intervene at certain steps in the automation. The human intervention become part of the intelligence captured by the models, creating a continuous cycle of training, testing, tuning, and validating the machine learning algorithms. This is necessary due to the acknowledged level of readiness of these enabling technology artifacts.

3.3 Using Event Streaming in the Context of Microservices Architecture

3.3.1 What Questions Remain Gaps in Knowledge for the Team in Implementing and Deploying This Enabling Technology?

From the TDEI perspective, there is one primary concern in using event streams- it is in identifying the message broker we use. A corollary to this question would be in choosing how to structure the event streams (“topics” as defined below) in a way to seamlessly apply it to the microservices architecture, (a) evaluating the benefits/detriments of using specific topics or more generalized topics for implementing and standardizing the microservices architecture and (b) taking careful consideration to designing the microservices, determining how they are decoupled and separated if each microservice is to truly have separate data access and view of the data stores, and if we strictly adhere to ‘no data sharing allowed’ among microservices.

3.3.2 What Are the Evaluation Steps to Follow for Each Question?

With microservices communicating via event streams, we can build a reliable platform to extend and grow while taking advantage of scalability and extensibility of the data schemas and types we intend to handle. An event is a particular activity your software system is doing. An example might be collecting temperature and humidity data. An event stream is an organization of similar events in chronological order. So, the temperature and humidity measures could be captured every minute of every day, and presented (organized) chronologically in an event stream, along with the time and date each measurement was taken.

A “topic” is a categorization of events of a related nature into a single stream. So event data about wind speed might be included in a separate event stream from the temperature and humidity data. The event stream containing wind data would be one topic, and the temperature and humidity data would be another.

In the TDEI, topics will define the interactions among the microservices and the message brokers the TDEI uses. Furthermore, we must ensure that there is compatibility between the microservice architecture and the open-source message broker we choose to use. Ultimately, we want the topics we design into the system to be both necessary and sufficient, and to ensure viability of these topics (namely, that microservices can connect to one or more topics. They can publish new events. They can consume the events. Events can also be simple notifications of actions that have been taken by the system, or they can carry state changes, allowing each microservice to maintain its own dataset.

The goal is to not be overrun by the constraints we create when defining the message streams and have the TDEI powered by resulting applications that are decoupled from one another and communicating effectively through event streams, passing data around asynchronously. We must avoid the indirection of one service calling another, that calls another. In this TRL extension, we are tasked with designing the communication topics through which the microservices will publish and listen to topics.

Importantly, we will need to establish additional criteria when evaluating the event stream design and the asynchronous messaging among the microservices. Specifically, we will want:

- Messages to be ordered chronologically
- Message delivery should be guaranteed. (all microservices should have an accurate picture of the system at large)
- Durability
- Resilience
- Latency kept to a minimum.

The system must also avoid outages as much as possible and have mechanisms that quickly recover from those failures when they do occur, including preventing (if possible) and minimizing (when necessary), data loss or service degradation.

Here we describe an eight-step process we will take to generate and evaluate the necessity, sufficiency, and viability of each event stream topic we define. The process extracts services and their associated needed topics progressively. This process allows us the iterative evaluation of whether a particular defined microservice provides the right functional encapsulation before fully committing to it because we will not create the microservice implementation or send production traffic until we have iterated through most of the priority use cases for the TDEI system. Below are the eight steps:

Step 1: Add Producer Logic

The entire process revolves around a dummy monolith application that roughly does what the TDEI interoperable data sharing server and data integration server would do. We start by providing access to data in the new extracted microservice, so the first step is to use the dummy monolith to push that data into the message event bus producer within the dummy monolith.

Step 2: Consume the Stream into the Database

To accept the data being produced in Step 1, a data store must be prepared. The data store keeps our service decoupled from the rest of the TDEI ecosystem. This new microservice contains both a consumer and the data store, allowing local access to state. To ensure that the data needed by requests to the microservice, it may be necessary to pre-load the data store with historical data.

Step 3: Test the Consumer

To ensure that the microservice functions as intended, we must perform validation testing. This ensures that the microservice is correctly processing the events without errors. Tests must also be performed to ensure that the consumer in the microservice is able to keep up with the data being added to the event stream. Event streams are designed such that consumers can get behind for a period of time – so our prototype will have to persist events for a period of time (on some event buses, like Kafka, the longevity of the events is user defined). We want to ensure that there is no data loss, and the tradeoff is that sometimes services end up with an out-of-date view of the data. An additional attribute of the event stream that needs to be evaluated is whether the chosen streaming technology acts as a buffer when the system is under high load. It is worthwhile to take time in this step to understand and prepare for scenarios where data consumers get behind.

Step 4: Determine the Logic Needed in the Microservice

In this step, we will extract the relevant functionality from the prototype monolith into the new microservice application. We will then test that it works in isolation, by manually executing procedures / API calls. We will need to validate that it is reading from the new data store within the microservice, without the need to process API calls to the data producer. It's important to test event triggers, but also be certain that we do not have any unintended consequences that impact other microservices that might be using the same event streams.

Step 5: Add, Test, and Consume Event Triggers

In this step, we add unit tests and take steps towards fully implementing API call producers for events. Different event streaming platforms provide capabilities to automate checks that events are streamed and triggered, as well as ensuring the events are flowing through. Once those are tested, we will add consumers to the service which will process these events, executing the relevant procedures within the service.

Step 6: Send Events Back from the Microservice

If a step is needed to communicate back to the monolith, we will need to add producers in the new microservice and consumers in the monolith to handle this.

Step 7: Test Microservice Events

At this point we can activate and test the new microservice with a test account (or user) and evaluate whether the actions desired from the microservice occur as desired and whether the dummy prototype monolith continues to operate as expected.

Step 8: Finally, Remove Deprecated Logic from Monolith

In this step, we can ramp-up the microservice usage along with the associated event stream topics. Once all accounts are using the new service, we can remove the functionality altogether from the dummy monolith and make the new microservice and its associated topic streams the default code path. Likewise, any logic still being used in the dummy monolith can be removed from the monolith.

These steps are intended to allow for a safe and thoughtful creation of topics that are tied to real use cases of the TDEI infrastructure. Ultimately, in this instance, too, the decisions will depend on the current knowledge of our development team and the ability to take into view multiple use cases of the TDEI. This effort will be led by the Data Management Architect and Lead, with support from the Technical Applications Lead and the Deployment Development Lead.

3.3.3 How Will You Evaluate the ET TRL in the Context of the Conditions for Your Project and Site?

We will take the previously performed details all our known use cases for the TDEI (for all three data standards) and their applied priorities.

We will spend eight additional weeks following the eight-step iterative process described above.

We will spend three additional weeks elevating the prototypes we had built into nearly functional prototypes so we can evaluate this functionality with some downstream data consumers (for instance, with a consumer app like AccessMap Multimodal or with a producer tenant app like Common Paths). This experience will provide us with actionable modification and data to evaluate the design choices we had made.

3.3.4 Which Team Members or Roles Will You Engage in This Investigation

Specifically, this effort will be led by the Data Management Architect and Lead, with support from the Deployment Development Lead and the Technical Application Lead.

3.3.5 How Will You Avoid Potential Bias of Your Group, Which Could Influence TRL Results?

We believe that using on-the-ground use cases and the real data we have had the privilege of having will help us avoid bias. We will also conduct a hypothetical discussion of how we may extend this infrastructure to support additional prioritized data schemas, such as supporting bicycle and micromobility data types. However, in the case of event buses and event streaming, it seems that many enterprise applications continuously add new topics. The most important endeavor in removing bias here will depend on the degree of flexibility and scalability we can maintain in the event bus, while still producing a functional instance for the deployment projects.

3.3.6 How Will You Ensure the Data You Use for the TRL Results Are Valid and Current?

We are fortunate to be a trusted partner to many organizations and institutions who have shared and continue to share data with us. They keep us up to date. Additionally, the device end-users are the best validity checks on our data. We maintain our data as up-to-date as we can to allow our application users to benefit from the use of the data.

3.3.7 Will Your Process Require the Reevaluation of the TRL Results in a Later Time in the Project to Support Future Phase 2 and 3 Documents?

Our deployment projects as well as TDEI tenant clients will provide the most informative evaluation of the outcomes of building this communication, messaging, and concurrency effort.

3.4 Using APIs and API Gateways in the Context of Microservices Architecture

3.4.1 What Questions Remain Gaps in Knowledge for the Team in Implementing and Deploying This Enabling Technology?

From the TDEI perspective, there is one primary concern in using APIs and API Gateways- it is in identifying where we use “off the shelf” API management and where we build our own. An additional, often controversial, question has to do with API governance. In a previous paragraph, we discussed some of the concerns and institutional solutions offered by other organizations in managing this aspect of producing APIs and API gateways. We will want to advance the idea that the TDEI APIs need to be sustainable and extensible and therefore we must (a) evaluate the benefits/detriments of using certain specific API calls with typed payloads versus more generalized API calls (b) taking careful consideration to design the API calls in a way that matches the use cases we addressed in 3.1.1 and 3.1.2 and that the API calls are sufficiently decoupled and separated if each producer and consumer is to truly have separate data access and view data stores in both overlapping and non-overlapping regions.

3.4.2 What Are the Evaluation Steps to Follow for Each Question?

These steps allow for a safe and thoughtful creation of API and API-Gateway interactions that are tied to real use cases of the TDEI infrastructure. Ultimately, APIs are the most visible mechanism for all our stakeholders to interact with the TDEI, so there are high stakes in the decisions made with respect to safeguarding the TDEI data and providing the services stakeholders require. It would be a good idea to reach beyond our development team and engage our partners and stakeholders in the evaluation of the API creation and governance.

The steps we follow in evaluating the API Gateway solutions need to be tied to the purpose of having that gateway in the first place: the implementation must accelerate, govern, secure, and provide a seamless experience to engage with our microservices-architecture based system.

As noted above, there are multiple options for open-source API Gateway solutions and we will need to ensure that our solution offers the right kind of support not only for our data schemas and data ingest protocols, but also for caching and traffic management, for monitoring API and service usage, provide adequate content vetting and filtering, and importantly, the security needed in authenticating users and understanding their roles with respect to the data.

3.4.3 How Will You Evaluate the ET TRL in Context of the Conditions for Your Project and Site?

The API Gateway’s main purpose is to provide the “First Line of Defense” by filtering out malformed message, query bombs, Denial of service attacks, data injection, or other breaches from external consumers. We will have to unit test all of these functionalities. We may want to do this first in the dummy monolith version, and then in the API designed (much as suggested in the prior step evaluating the event streaming technology).

Many API gateways can virtualize web services location, thereby hiding their real location and implementation details from its external consumers keeping it safe from attacks. Many API Gateway provides have numerous inbuilt Out-of-the-Box functionalities that we can implement and evaluate at this step, as we attempt to filter threatening external messages. This is also the opportunity to test load and throttling of inbound message flow which we are likely to deploy with an out of the box solution.

3.4.4 Which Team Members or Roles Will You Engage in This Investigation?

Specifically, this effort will be led by the Data Management Lead, with support the Deployment Development Lead and the Technical Application Lead.

3.4.5 How Will You Evaluate the ET TRL in Context of the Conditions for Your Project and Site?

The API Gateway's main purpose is to provide the "First Line of Defense" by filtering out malformed message, query bombs, Denial of service attacks, data injection, or other breaches from external consumers. We will have to unit test all of these functionalities. We may want to do this first in the dummy monolith version, and then in the API designed (much as suggested in the prior step evaluating the event streaming technology).

Many API gateways can virtualize web services location, thereby hiding their real location and implementation details from its external consumers keeping it safe from attacks. Many API Gateway provides have numerous inbuilt Out-of-the-Box functionalities that we can implement and evaluate at this step, as we attempt to filter threatening external messages. This is also the opportunity to test load and throttling of inbound message flow which we are likely to deploy with an out of the box solution.

3.4.6 How Will You Avoid Potential Bias of Your Group, Which Could Influence TRL Results?

We believe that using on-the-ground use cases and the real data we have had the privilege of having will help us avoid bias. We are also privileged to be part of two communities that are integral to this conversation, and we can engage these two groups in outside discussions, probing and critiquing our APIs and approach to the TDEI data infrastructure. The two communities are MobilityData.org (a non-profit international organization specifically engaging) and Mobility Data Interoperability Principles (a community of co-authors that recently published with us the White Paper and Principles by the same title. The web home for the principles can be found here: www.interoperablemobility.org). Both these communities will be instrumental in providing the diverse views of mobility service providers (transit agencies and other companies who provide rides), transportation technology companies (software or hardware vendors), research institutions, transportation system managers (DOTs and other regulators of transportation infrastructure), and the public.

3.4.7 How Will You Ensure the Data You Use for the TRL Results Are Valid and Current?

We are fortunate to be a trusted partner to many organizations and institutions who have shared and continue to share data with us. They keep us up to date. In the case of the APIs, the communities of practice mentioned in the previous section are the best validation for our APIs and their usability and relevance.

3.4.8 Will Your Process Require the Reevaluation of the TRL Results in a Later Time in the Project to Support Future Phase 2 and 3 Documents?

Our deployment projects as well as TDEI tenant clients will provide the most informative evaluation of the outcomes of building this API Endpoint effort.

3.5 TRL Ratings for Inventoried Enabling Technologies

In this section, we follow the same order for each enabling technology from section 2.2 Enabling Technologies Inventory, and provide a detailed Technology Readiness Assessment (TRA) per the Framework introduced in Section 2.1. We intend to provide enough information to justify the Technology Readiness Level we establish at this time, and also highlight the remaining needs and path forward for each enabling technology that is required specifically in the development of the TDEI infrastructure.

3.5.1 TRA: Microservice Architecture

Table 10 Technology readiness assessment (TRA) for 2.2.1 Microservices Architecture: Enabling Data Collection, Aggregation, Integration and Transformation

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

Tech Readiness Level 1 **Basic principles and research**

- Do basic scientific principles support the concept?
  Yes, we are learning from the experiences of thousands of companies (such as Netflix and Capital One) and projects. The experience of others makes is clear to our team that for the TDEI to create a sustainable data interoperability infrastructure, requires creating a non-monolith architecture. Moreover, the concept of Microservices has been tried and proven in this context.

- Has the technology development methodology or approach been developed?
  Yes, the technology has been developed and is currently on offer both as an architectural paradigm, through open-source projects, as well as available as pre-programmed templates for purchase through large cloud infrastructure and data vendors like Microsoft Azure, Amazon AWS, etc.

Tech Readiness Level 2 **Application formulated**

- Are potential system applications identified?
  Yes, many organizations have deployed this strategy. From the TDEI perspective, there are two primary considerations in choosing how to apply the microservices architecture (a) evaluating the benefits/detriments of using specific languages for implementing the microservices architecture (b) careful consideration of the microservice separated from each other ensuring decoupling and no-shared data access.

  For these two considerations, we intend to apply the criteria described below. These criteria are taken in large part from a Clarion Technologies web site designed to help

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

explain what Microservices architectures are, and what languages are well suited for use in those architectures.[27]

Choosing Technologies for Use in Microservices

Microservices, can be built using a variety of programming languages. Different technologies or languages can be used for different microservices. One downside of using multiple languages is that using diverse programming languages can raise the performance overhead, and microservice architectures often already have considerable operational overhead. Standardizing the technology stack used can limit that outcome.

As for any specific platform to deploy. Common thought is that it is not recommended to start microservices architecture from scratch since it is difficult to define the boundaries of each service at the beginning. There is no better way to choose the perfect microservices breakdown other than outlining the use cases TDEI wishes to address and come to a deep understanding of what tools and modularization will be best for our microservices. Ultimately, technology decisions will also depend on the current knowledge of our development team.

Common languages used when building microservices architecture, and their attributes, are given below. The five languages described below not the only technology options. However, they are excellent examples of the fact that the current state of the practice supports the microservice architecture choice.

Java

A quick examination of material posted by software firms that support microservices shows that all support use of Java for microservices. Jetbrain's annual survey[28] of developers reports that more survey respondents use Java (41 percent) than any other language. Java is recommended for a variety of reasons, including the following:

- The approach for placing annotations in Java is particularly easy to read, making it developer friendly.

[27] https://www.clariontech.com/blog/5-best-technologies-to-build-microservices-architecture

[28] https://www.jetbrains.com/lp/devecosystem-2021/microservices/#:~:text=Primary%20languages%20among%20microservices%20developers&text=The%203%20most%20popular%20languages,%2C%20and%20Python%20(25%25).

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

- The Java Platform Enterprise Edition (EE), the current edition aimed at large distributed enterprise or Internet environments[29] comes with considerable functionality that is specifically designed to support microservices architectures. Some of this functionality includes:
    - JAX-RS is an annotation driven JAVA API designed to make development of Web services more straightforward and intuitive,
    - JPA is a specification that lets a developer define which objects should persist and how they persist,
    - CDI (Contexts and Dependency Injection) is a Java EE feature that helps knit the web tier and the transactional tier within the Java EE platform.
    - A number of service discovery solutions have been built that support connection of microservices, including Consul, Netflix Eureka or Amalgam8.[30]
- Several frameworks exist for Java for developing microservices. These frameworks make work easier and faster, as well as helping simplify the configuration and setup process. They also help with communication between microservices. Among these frameworks are:
    - **Spring Boot – This framework works on top of various languages for Aspect-Oriented programming, Inversion of Control and other functionality.[31]**
    - Dropwizard – This Java microservices framework for developing ops-friendly, high-performance, RESTful web services[32] and that helps assemble libraries of Java into a simple and light-weight packages.
    - **Restlet – helps Java developers build better web APIs that follow the REST architecture style.[33]**
    - Spark – a microframework for creating web applications in Kotlin and Java 8 with minimal effort.[34]

[29] "Java Platform, Enterprise Edition (Java EE)". *Oracle Technology Network*. Oracle. Archived from the original on December 17, 2014. Retrieved December 18, 2014.

[30] https://stackoverflow.com/questions/13047807/why-use-cdi-in-java-ee

[31] https://spring.io/

[32] https://www.dropwizard.io/en/latest/

[33] https://restlet.talend.com/

[34] https://sparkjava.com/

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

Go / Golang

Go, also known as Golang because of its original domain name, is a statically typed, compiled high-level programming language designed at Google.35 Go is considered to be easy for code writing, has a high level of security, a high execution speed, and a law entry threshold.36 It is also popular for its concurrency and API support, resulting in productivity of various machines and cores. Web reviews also recommend Go because its simple syntax and excellent testing support.

As with Java, Go has a wide range of libraries available to simplify and speed coding. Examples libraries are:

- Gizmo[37] – a toolkit that simplify providing packages of PubSub daemons which asynchronously exchange messages in real time.
- Go Kit[38] – which provides support for infrastructure integration, Remote Procedures Call (RPC) safety, system observability and program design.
- GoMicro[39] – which is an RPC framework which supports load balancing, server packages, PRC clients, and message encoding.
- Kite[40] – an RPC library that helps a developer write distributed systems.

These frameworks all have specific advantages or limitations. For example, one major difference between Go Kit and GoMicro is that Go Kit must be imported into a binary package. However, it is more advanced for explicit dependencies, domain-driven design, and declarative aspect compositions.

[35] Kincaid, Jason (November 10, 2009). "Google's Go: A New Programming Language That's Python Meets C++". TechCrunch. Retrieved January 18, 2010.

[36] https://surf.dev/why-golang-with-microservices/

[37] https://github.com/NYTimes/gizmo

[38] https://gokit.io/

[39] https://github.com/go-micro/go-micro

[40] https://github.com/koding/kite

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

Python

Python is a high-level, general-purpose programming language. It offers active support for integration with various technologies. Python allows developers to quickly write application code, plug in boilerplate functions and test the programs before converting them to script. Python is also a strongly typed language, meaning it ensures uniform consistency and minimizes errors by enforcing data types.[41] Python is specifically recommended for use in microservice architectures for a variety of reasons including the following:

- Python's advanced scripting capabilities also allow developers to automate systems provisioning and configurations for microservices.
- Python's standard library is augmented by thousands of third-party libraries for writing REST services.
- Python provides strong support for containers.
- Prototyping with Python can be easier and quicker than in other languages.[42]
- Python is compatible with legacy languages like ASP and PHP that help create web service front-ends to host Microservices.[20]

Python provides a broad range of microservices frameworks. Among the framework choices are:

- Flask - is used for developing web applications and is implemented on Werkzeug and Jinja2.[43] It is lightweight, supports secure cookies, and has a built-in development server and fast debugger, and support for unit testing is built-in. [44]

[41] https://www.techtarget.com/searchapparchitecture/tip/How-viable-is-it-to-create-microservices-in-Python

[42] https://dzone.com/articles/is-python-effective-for-microservice-architecture

[43] https://www.analyticsvidhya.com/blog/2021/10/easy-introduction-to-flask-framework-for-beginners/

[44] https://flask.palletsprojects.com/en/2.2.x/

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

- Falcon – is a high-performance web framework for building a REST API and micro-services in Python.[45] Falcon allows the creation of smart proxies, cloud APIs and app back-ends.
- Bottle - is a fast, simple and lightweight web service gateway interface micro web-framework.[46]
- Nameko - comes with built-in support for: RPC over Advanced Message Queuing Protocol (AMQP) and Asynchronous events (pub-sub) over AMQP.[47] It is designed to quickly build a service that can respond to RPC messages, dispatch events on certain actions, and listen to events from other services. It could also have HTTP interfaces for clients that can't speak AMQP, and a websocket interface for, say, Javascript clients.
- CherryPy – is a pythonic, object-oriented web framework. It allows developers to build web applications in much the same way they would build any other object-oriented Python program. This results in smaller source code developed in less time.[48]

<u>Node JS</u>

Node.js is a cross-platform, open-source server environment that can run on multiple operating systems. It is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.[49] The runtime is intended for use outside of a browser context (i.e. running directly on a computer or server OS). As such, the environment omits browser-specific JavaScript APIs and adds support for more traditional OS APIs including HTTP and file system libraries.[50]

The main Node.js web site[51] states that "Node.js is great for decoupled applications as you can use lots of npm modules to sew up a great microservice. Node.js is fast and its

[45] https://phrase.com/blog/posts/falcon-python-i18n/

[46] https://bottlepy.org/docs/dev/

[47] https://nameko.readthedocs.io/en/stable/what_is_nameko.html

[48] https://docs.cherrypy.dev/en/latest/

[49] https://en.wikipedia.org/wiki/Node.js

[50] Mozilla Developer Platform: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_nodejs/Introduction

[51] https://nodejs.org/en/about

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

event-based nature makes it a great choice even for real-time applications." It also states that "many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep" making the system more efficient than many other types of concurrency models.

Node.js also has a large database of JavaScript modules that simplify and speed up application development, and as with the other languages examined, Node.js has multiple frameworks that can be accessed and used. Three examples include:

- Express - is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.[52]
- Sail – is an open source model-view-controller web application framework developed under the MIT license designed to make it easy to build enterprise-grade software[53]
- Hapi – is a framework used to build scalable web applications including Application Programming Interface servers, HTTP-proxy applications, and websites.[54]

.Net

ASP.NET is an open-source web framework created by Microsoft and derived from .NET used for building modern web applications. It is specifically adapted for writing backends for web pages and web applications. Developers can use the same tools, libraries, and infrastructure to build web and desktop projects. One reason for using .NET is in its simplicity. You can quickly comply with the outfit, use library components, and manage framework classes.[55]

Microsoft states that ASP.NET "makes it easy to create the APIs that become your microservices. ASP.NET comes with built-in support for developing and deploying your microservices using Docker containers. .NET includes APIs to easily consume microservices from any application you build, including mobile, desktop, games, web,

[52] https://expressjs.com/

[53] https://sailsjs.com/

[54] https://www.section.io/engineering-education/introduction-to-hapi/

[55] https://jelvix.com/blog/asp-net-vs-asp-net-core

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

and more. You can find the official Docker images for .NET on DockerHub, meaning the initial setup is done and you can focus on building your microservices."[56]

.NET can be used with other technology stacks. It allows a mix of technologies between each service, allowing it to be used for some aspects of a larger application but not for all components of that application. For example, .NET microservices can be mixed with those written in any of the other options described earlier in this section.

- Are system components and the user interface at least partly described?
  Yes, all architectural components are fully described. In particular, the architectural paradigm we intend to use is frequently used in industry. It is described below this table.

- Do preliminary analyses or experiments confirm that the application might meet the user need?

Yes, Microservices architecture has become omnipresent in software and mobile app development. This model is helping developers to address multiple diverse stakeholders needs, scale and extend their applications without having to refactor or rebuild code each time. The architecture has been transformative in cloud application development. The TDEI small team can specifically benefit from the modular nature of building out microservices.

***Tech Readiness Level 3* Proof of concept**

- Are system performance metrics established?

  System performance metrics are not necessarily established for microservice architectures because the performance of the architecture is entirely a function of the tasks being performed. However, there are criteria that are recognized as being important for selecting between open-source solutions that must be integrated together. Following these criteria, the TDEI will have to evaluate the following:

  - Data producer/consumer-first approach (including device end-users, app developers, and analytics)
  - Programming language based on ease of use to their developers
  - Ability to be independently deployed
  - Ability to support automation
  - Decentralization of components

[56] https://dotnet.microsoft.com/en-us/apps/aspnet/microservices#

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

- Support for continuous integration

As noted above, many frameworks, versions, and tools can support the development of Microservices. Java, Python, C++, Node JS, and .Net are exemplars.

Ease of Management

In the instructional material Microsoft provides on microservice gateways,[57] it is recommended that when services are updated or new services are added, the gateway routing rules may need to be updated. The TDEI will have to consider how updates to services will be managed. The TDEI expects to apply these same management approaches for SSL certificates, IP allow lists, and other aspects of configuration.

- Is system feasibility fully established?
  Yes, we believe the hard part would be to select the system subdivisions that are the best fit for our system and team constraints and maintain microservices separate.

- Do experiments or modeling and simulation validate performance predictions of system capability?
  There are current system deployments that allow us to make rough predictions. We are in the process of prototyping with some of these tools to assess how well they may fit with our project.

- Does the technology address a need or introduce an innovation in the field of transportation?

We do not know of other implementations in the field directly, although ride hail companies or micromobility companies may be working with similar infrastructure, though it is not open**,** nor does it produce shareable data.

Tech Readiness Level 4 **Components validated in a laboratory environment**

- Are end-user requirements documented?
  Yes, due to the numerous industry operators in this space, multiple stakeholder needs have been observed and documented, although not necessarily in the transportation or accessibility domains.

- Does a plausible draft integration plan exist, and is the technology's compatibility demonstrated?

[57] https://learn.microsoft.com/en-us/azure/architecture/microservices/design/gateway

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

Please see figures below for a draft integration plan modelled after often-used deployments of these enabling technologies.

- Were individual components successfully tested in a laboratory environment (a fully controlled test environment where a limited number of critical functions are tested)?
  Individual components are tested and in use daily in industry. In our laboratory, we are in the process of implementing prototype integrations.

Tech Readiness Level 5 **Integrated components demonstrated in a laboratory environment**

- Are external and internal system interfaces documented?
  External system interfaces and documented. Some are actual turnkey system deployments that allow us to use them off the shelf until we have the capacity to tune them to our specific TDEI needs.

- Are target and minimum operational requirements developed?
  Yes, these will be aligned or exceed the requirements identified in the Systems Requirements documents for the TDEI.

- Is component integration demonstrated in a laboratory environment (i.e., fully controlled setting)?
  Integration of these components has been demonstrated both in controlled environments and in the wild.

Based on the Technology Readiness Framework introduced by the *FHWA Technology Readiness Level Guidebook,* we followed the procedure above and conclude that Microservices Architecture technologies are at readiness level 5 but requires significant architecture planning efforts in order to gain the most benefit from use of this technology.

Additional Notes: Assessing the use of Microservices and how we may decouple microservices required us to investigate actual architectures that currently use these technologies in other data service system. This resulted in some of the comparisons that were drawn above.

3.5.2 TRA: Using Event Streams in the Context of Microservices Architecture

Table 11 Technology readiness assessment (TRA) for 2.2.1 Microservices Architecture: Enabling Data Collection, Aggregation, Integration and Transformation with APIs and API Gateways

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.
Tech Readiness Level 1 **Basic principles and research** • **Do basic scientific principles support the concept?** • **Event Streams were discussed earlier for their importance in removing the complexity of point-to-point communication among disparate part of an application by acting as a communication hub between systems.** • **Has the technology development methodology or approach been developed?** • **Yes, the technology has been developed and is currently on offer both as an architectural paradigm, through open-source projects, as well as available as pre-programmed templates for purchase through large cloud infrastructure and data vendors like Microsoft Azure, Amazon AWS, etc.**
Tech Readiness Level 2 **Application formulated** • Are potential system applications identified? Yes, the primary considerations will go to evaluating the benefits/detriments of using the following Message Brokers: • Apache Kafka   • Kafka in the cloud   • Apache PULSAR   • Redpanda • Are system components and the user interface at least partly described? Yes, all architectural components are fully described. In particular, the broker services we intend to use are frequently used in industry. Additional criteria to compare services are described below. • Do preliminary analyses or experiments confirm that the application might meet the user need? Yes, Message Brokers have become commonplace among most data providers, albeit not in transportation or accessibility. While we are trailblazing this development in this industry, the components themselves are not novel. We believe the concurrency model will really help transportation IT developers to address multiple

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

diverse stakeholders needs, scale and extend their applications without having to refactor or rebuild code each time.

***Tech Readiness Level 3 Proof* of concept**

- Are system performance metrics established?
  System performance metrics are not necessarily established, but there are message broker features that are generally established as implementation tradeoffs. For any of the selected open-source solutions named above, the TDEI will have to evaluate how each system handles:

  - Brokers
  - Limits on the number of topics
  - Cluster coordination
  - Multi data center replication and offset handling
  - Service discovery
  - Scaling clusters
  - Clients
  - Reading Messages
  - Reading and persisting data from external sources
  - Integration with other sources
  - Message query operations
  - Long-term storage
  - Community support and documentation

  A number of companies have published comparisons of Kafka and Pulsar. These include:

  - Cloud Infrastructure Partners (**https://cloudinfrastructureservices.co.uk/kafka-vs-pulsar-whats-the-difference/**)
  - Digitalis (**https://digitalis.io/blog/kafka/apache-kafka-vs-apache-pulsar/**)
  - Confluent (**https://www.confluent.io/kafka-vs-pulsar/**)
  - StreamNative (**https://streamnative.io/blog/apache-pulsar-vs-apache-kafka-2022-benchmark**)
  - Macrometa (https://www.macrometa.com/event-stream-processing/kafka-alternatives)

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

Redpanda is a Kafka-compatible streaming data platform that claims to be optimized for lower latency and thus higher throughput. Consequently, comparison web sites that cover many or all of these factors exist that compare Redpanda to Kafka. Examples of these sites include:

- https://www.kai-waehner.de/blog/2022/11/16/when-to-choose-redpanda-instead-of-apache-kafka/
- https://www.infoworld.com/article/3660628/review-redpanda-gives-kafka-a-run-for-its-money.html
- https://techwithadrian.medium.com/a-closer-look-at-redpanda-37015edb0841
- https://medium.com/event-driven-utopia/real-time-streaming-for-mortals-how-redpanda-and-materialize-making-it-a-reality-18ac0bdc6f43
- https://sourceforge.net/software/compare/Apache-Kafka-vs-Redpanda/

Each of these review articles comes to different conclusions based on the specific needs of the system being designed and deployed. The TDEI team will need to carefully consider the pro's and con's of the alternatives.

- **Is system feasibility fully established?**
  Yes, we believe the hard part would be to select the system components that are the best fit for our system and team constraints.

- Do experiments or modeling and simulation validate performance predictions of system capability?
  There are current system deployments that allow us to make rough predictions. We are in the process of prototyping with some of these tools to assess how well they may fit with our project.

- Does the technology address a need or introduce an innovation in the field of transportation?
  We do not know of other implementations in the field directly, although ride hail companies or micromobility companies may be working with similar message brokerage. Uber is likely to be using similar message brokerage. That infrastructure not open nor does it produce shareable data.

Tech Readiness Level 4 **Components validated in laboratory environment**

- Are end-user requirements documented?
  Yes, due to the numerous industry operators in this space, multiple stakeholder needs have been observed and documented, although not necessarily in the transportation or accessibility domains.

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

- Does a plausible draft integration plan exist, and is component compatibility demonstrated?
  Please see the sample integration plan in Section **2.3.2** for a draft integration modelled after Azure-cloud-based deployments of message brokerage technology.

- Were individual components successfully tested in a laboratory environment (a fully controlled test environment where a limited number of critical functions are tested)?
  We have recently tested the publishing and listening to event streams in our laboratory towards the development of a microservices architecture for the integration server. We are in the process of implementing further prototype integrations.

Tech Readiness Level 5 **Integrated components demonstrated in a laboratory environment**

- Are external and internal system interfaces documented?
  External system interfaces and documented. Some are actual turnkey system deployments that allow us to use them off the shelf until we have the capacity to tune them to our specific TDEI needs. Please see evaluation criteria and discussion of Kafka and Pulsar above.

- Are target and minimum operational requirements developed?
  Yes, these will be aligned or exceed the requirements identified in the Systems Requirements documents for the TDEI.

- Is component integration demonstrated in a laboratoryenvironment (i.e., fully controlled setting)?
  Integration of these components has been demonstrated both in the controlled environment of our lab and in the wild.

Based on the Technology Readiness Framework introduced by the *FHWA Technology Readiness Level Guidebook,* we followed the procedure above and conclude that message brokerage technologies are at readiness level 5.

3.5.3 TRA: Using APIs and API Gateway within the Context of Microservices Architecture

Table 12 Technology readiness assessment (TRA) for 2.2.1 Microservices Architecture: Enabling Data Collection, Aggregation, Integration and Transformation with APIs and API Gateways

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.
***Tech Readiness Level 1* Basic principles and research** • Do basic scientific principles support the concept? Yes, we are learning from the experiences of hundreds of organizations in other industries and their successful ongoing work within their own industries in developing and implementing open data standards. The shift to open APIs presented a big, enabling shift for these industries towards data sharing and data sustainability. • Has the technology development methodology or approach been developed? Yes, the technology has been developed and is currently on offer both as an architectural paradigm, through open-source projects, as well as available as pre-programmed templates for purchase through large cloud infrastructure and data vendors like Microsoft Azure, Amazon AWS, etc.
***Tech Readiness Level 2* Application formulated** • Are potential system applications identified? Yes, the primary considerations will go to evaluating the benefits/detriments of using the following applications. These considerations come directly from Microsoft's Azure documentation, designed to assist developers in choosing a gateway technology. The following material is taken directly from that site.[58] "Reverse proxy server. Nginx and HAProxy are popular reverse proxy servers that support features such as load balancing, SSL, and layer 7 routing. They are both free, open-source products, with paid editions that provide additional features and support options. Nginx and HAProxy are both mature products with rich feature sets and high performance. You can extend them with third-party modules or by writing custom scripts in Lua. Nginx also supports a JavaScript-based scripting module referred to as NGINX JavaScript. This module was formally named nginScript.

[58] https://learn.microsoft.com/en-us/azure/architecture/microservices/design/gateway

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

Service mesh ingress controller. Service meshes to consider include Linkerd or Istio. It is important to consider the features that are provided by the ingress controller for that service mesh. For example, the Istio ingress controller supports layer 7 routing, HTTP redirects, retries, and other features.

Application Gateway. Application Gateway is used to manage load balancing services that can perform layer-7 routing and SSL termination. It may also provide a web application firewall (WAF)."

The TDEI development team has not yet identified open-source turnkey solutions for Application Gateways, but Azure supplies its own, and one can use open-source reverse proxy solutions and then add functionality. Additional functionality that could be offloaded to an application gateway, and the additional off the shelf microservices to support them:

- SSL termination
- Authentication
- IP allow/block list
- Client rate limiting (throttling)
- Logging and monitoring
- Response caching
- Web application firewall
- GZIP compression
- Servicing static content

API Management. There are plenty of open-source examples of organizations publishing APIs to external and internal customers. These examples provide features that can be useful to the TDEI development team for managing our public-facing APIs. For example, the team can benefit from examples in rate limiting, IP restrictions, and authentication using identity providers.

API Management doesn't perform any load balancing, so it should be used in conjunction with a load balancer such as the Application Gateways described above. There are at least 20 open-source, used API Management tools to choose from, all described in this article.[59] We will have to choose from among these options.

- Are system components and the user interface at least partly described?

[59] https://appinventiv.com/blog/open-source-api-management-tools/

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

Yes, all architectural components are fully described. In particular, the architectural paradigm we intend to use is frequently used in industry. It is described below this table.

- Do preliminary analyses or experiments confirm that the application might meet the user need?
  Yes, Microservices architecture and APIs have become omnipresent in software and mobile app development. This model is helping developers to address multiple diverse stakeholders needs, scale and extend their applications without having to refactor or rebuild code each time. The architecture has been transformative in cloud application development. The TDEI small team can specifically benefit from the modular nature of building with APIs and API Layers because different teams (some might be outside our organization) would be able to develop to the API specification. There is also the additional benefit of protecting the data assets and interfaces via well managed APIs and API Gateways.

***Tech Readiness Level 3 Proof* of concept**

- Are system performance metrics established?

  System performance metrics are not necessarily established, but there are performance features that are generally established as implementation tradeoffs are being considered. For any of the selected open-source solutions integrated together, the TDEI will have to evaluate:

Features.

For instance, the options listed above for Application Gateways all support layer 7 routing, but support for other features will vary. Depending on the features that we need, we may identify different solutions that fit the needs, or we may have to deploy more than one application gateway, depending on the API and microservices underlying them.

Ease of Deployment.

Microsoft supplies considerable documentation on its Azure architecture web site in support of microservices deployment.[60] That web site describes several ways to deploy gateways, allowing the TDEI team to select the approach that the team can use to ease deployment. Microsoft offers four separate approaches to deployment.

[60] https://learn.microsoft.com/en-us/azure/architecture/microservices/design/gateway

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

The first is "Nginx and HAProxy will typically run-in containers inside the cluster but can also be deployed to dedicated VMs outside of the cluster. This isolates the gateway from the rest of the workload but incurs higher management overhead."

A second alternative recommended by Microsoft is to "deploy Nginx or HAProxy to Kubernetes as a ReplicaSet or DaemonSet that specifies the Nginx or HAProxy container image. We would then use a ConfigMap to store the configuration file for the proxy, and mount the ConfigMap as a volume, and create a service of type LoadBalancer to expose the gateway through a Load Balancer."

A third Microsoft alternative is to "create an Ingress Controller. An Ingress Controller is a Kubernetes resource that deploys a load balancer or reverse proxy server. Several implementations exist, including Nginx and HAProxy. A separate resource called an Ingress defines settings for the Ingress Controller, such as routing rules and TLS certificates. That way, we will not need to manage complex configuration files that are specific to a particular proxy server technology. "

A fourth option is to deploy fully managed services such as Azure Application Gateway and API Management.

Ease of Management.

In the instructional material Microsoft provides on microservice gateways,[61] it is recommended that when services are updated or new services are added, the gateway routing rules may need to be updated. The TDEI will have to consider how updates to services will be managed. The TDEI expects to apply these same management approaches for SSL certificates, IP allow lists, and other aspects of configuration.

- Is the system feasibility fully established?
  Yes, we believe the hard part would be to select the system components that are the best fit for our system and team constraints, ensure that it is secure and open source.

- Do experiments or modeling and simulation validate performance predictions of system capability?
  There are current system deployments that allow us to make rough predictions. We are in the process of prototyping with some of these tools to assess how well they may fit with our project.

[61] https://learn.microsoft.com/en-us/azure/architecture/microservices/design/gateway

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.
• Does the technology address a need or introduce an innovation in the field of transportation? We do not know of other implementations in the field directly, although ride hail companies or micromobility companies may be working with similar infrastructure, though it is not open, nor does it necessarily produce shareable data.
Tech Readiness Level 4 **Components validated in laboratory environment** • Are end-user requirements documented? Yes, due to the numerous industry operators in this space, multiple stakeholder needs have been observed and documented, although not necessarily in the accessibility domain. We recently joined many transit organizations in co-authoring the Mobility Data Interoperability Principles, and all entities a**cknowledged** the need for standardization and standard practices, of which API publication is one. • Does a plausible draft integration plan exist, and is component compatibility demonstrated? Please see figures below for a draft integration plan modelled after often-used deployments of these enabling technologies. • Were individual components successfully tested in a laboratory environment (a fully controlled test environment where a limited number of critical functions are tested)? Individual components are tested and in use daily in industry. In our laboratory, we have been deploying and implementing the OpenSidewalks API since 2018. We are in the process of implementing prototype integrations.
Tech Readiness Level 5 **Integrated components demonstrated in a laboratory environment** • Are external and internal system interfaces documented? External system interfaces and documented. Some are actual turnkey system deployments that allow us to use them off the shelf until we have the capacity to tune them to our specific TDEI needs. • Are target and minimum operational requirements developed? Yes, these will be aligned or exceed the requirements identified in the Systems Requirements documents for the TDEI. • Is component integration demonstrated in a laboratory environment (i.e., fully controlled setting)? Integration of these components has been demonstrated both in controlled environments and in the wild.

Process: In the TRL tables, each row assesses the technology under consideration per a specific Technology Readiness Level. We will specify the TRL Name and Description, and then state its requirements along with our response to the requirements regarding the technology under evaluation.

We believe these technologies (API and API Gateways) are at readiness level 5.

Additional Notes: Assessing the use of APIs and API gateways required us to investigate actual architectures that currently use these technologies in other data service system. We identified one architectural paradigm that is used in the field and fits well into fits well for the TDEI use case.

Figure 5 shows how the integrated API architecture defines the process for running and exposing TDEI APIs. Specifically, the initial layer has the API Portal, through which data consumers and producers register, sign up for community messaging, access TDEI documentation, and have access to monitoring data about the TDEI system. The API Portal is the mechanism by which data producers, consumers, TDEI data tenants (internal and external) and all third-party data stakeholders can access the TDEI framework for API analysis, API documentation, and to ensure that the data interoperates with web/mobile applications. Through the development of the portal, TDEI will define how we expose data to internal, partner, and third-party developers. Protected by the portal layer, is the API Gateway layer where implementations include API security, data validation, data caching, and service orchestration.

These architectural methods are widely used in the field. The diagram fits into our overall architecture (refer to Figure 3 in the component integration; this figure is a detailed view of the boxes labeled “API Gateway,” “Internal API Management” and “APIs for users, applications, analytics” found in both of the “API Layer” panels). The API layer in the TDEI lends agility and scale to the overall architecture and the ability to modify as new use cases and TDEI stakeholders become TDEI data tenants.

The ability to decouple the analytics services and APIs from the data lake means that we can serve parallel data schemas with very different input types and continue to scale linearly with the data (where the data processing runs separately from the data storage and distribution/ dissemination). In this way, we can serve requests by application developers, for example, that are real-time and allow dynamic access and are able to support many more users and downstream applications. These practices are currently hailed as industry’s best standards for organizations that are going to scale out the data to large numbers of users with multiple downstream applications (see Arcadia Data testament, for example). The example provided allows the downstream data consumption to be fully distributed and allows the TDEI to support users making native queries on complex data sources (for instance, only sidewalks of a particular kind in a very large region), allow for user concurrency and scalability regardless of any latencies in data processing.

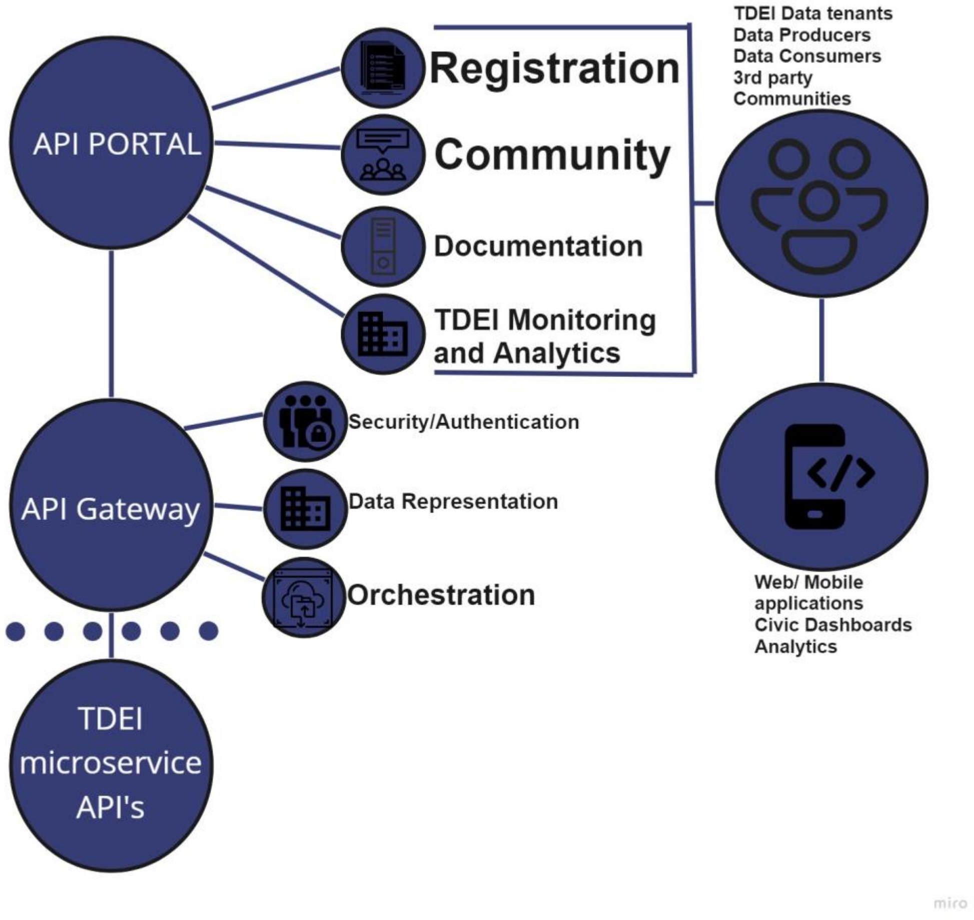


Figure 5 Architecture diagram demonstrating a widely used paradigm for API Gateways and APIs for microservices integration into a complete microservice architecture that promotes cyber safety and data interoperability.

Based on the Technology Readiness Framework introduced by the *FHWA Technology Readiness Level Guidebook,* we followed the procedure above and conclude that use of APIs and API Gateways technologies are at readiness level 5.

4 Risk Assessment

Here we identify specific risks (e.g., performance data gaps, utilization of standards, etc.) associated with the specific technologies, which provides a basis for quantifying those risks through formal risk assessments.

4.1 Assessing Risk

Describe how the risk assessment was performed and criteria for defining high, medium, and low impact. With the proliferation of services and containers, orchestrating and managing large groups of containers quickly became one of the critical challenges.

Table 13. Risk assessment for each enabling technology

Risk ID	Enabling Technology	Risk Description	Impact Level
1	2.2.3 Application Programming Interfaces	Even with use of APIs, there remain design choices that can be made in the deployment of APIs that can make the use of the underlying services difficult to integrate into the full system deployment. These challenges include: • API resource parameters and defining them within APIs • API resource design workflow • Designing and implementing API resource relations • API actions • API versioning • API response pagination and metadata, filtering, sorting, search, long-running operations, concurrency control, conditional requests & caching, error handling, bulk operations, file uploads • API documenting • API security	Medium
2	2.2.3 Application Programming Interfaces	Exposing an API means being exposed to cyber threats. Providing a reliable API endpoint requires safeguarding against application-level threats.	High
3	2.2.3 Application Programming Interfaces	Exposing an API means requirement to adequately route traffic and balance load. Providing a reliable API endpoint requires safeguarding against this issue.	High
4	2.2.4 Intermediary API Gateway Layers Help Integrate APIs	The API gateway is a potential bottleneck or single point of failure in the system.	High

Risk ID	**Enabling Technology**	**Risk Description**	**Impact Level**
5	2.2.1.2 Key Enabling Technology Components for Microservices: Containers and Orchestration Managers	Designing disparate microservices creates challenges in communication among these services. We intend to have multiple tenants in the TDEI environment, each potentially running some portion of their own microservices that suit their data environment. The risk has to do with how these services interact with each other and with the TDEI system.	High
6	2.2.1.2 Key Enabling Technology Components for Microservices: Containers and Orchestration Managers	Lossless Data Delivery: Data updates and batch upload events initiated by the multiple tenants of the TDEI create challenges for messaging and streaming update events, particularly if we are to adopt the acceptable best practice design of stateless services. State of datasets does exist, particularly for transportation and municipal agencies, where changes and updates will occur spontaneously rather than on a release schedule. The services need to be aware of these changes, and though an API call is often an effective way of initially establishing state for a given service, it's not good for updates if they need to be polled constantly (this is to say that the microservices operating within the tenant agencies will have to send notifications to the TDEI when they have updates, rather than the TDEI data infrastructure constantly 'asking' the tenant microservices whether they have updates).	High
7	2.2.2 Message Streaming and Brokering: Enable Integration of the Data Interoperability Platform	Flexible topic routing	Medium
8	2.2.2 Message Streaming and Brokering: Enable Integration of the Data Interoperability Platform	Message ordering may get missed	Low
9	2.2.2 Message Streaming and Brokering: Enable Integration of the Data Interoperability Platform	Stability of the orchestration servers create a high-risk vulnerability	High
10	2.2.2 Message Orchestration in cloud	Unstable life cycle in cloud	High

4.2 Mitigating Risk

For each of the "High" impact risks identified in Section 4, we assess the likelihood of the risk occurring (High, Medium, or Low) and describe the mitigation strategy that will be used to reduce the risk. These risks will be tracked on an ongoing basis in the Risk Register.

Table 14. High-impact risk mitigation plans

• **Risk ID**	• **Risk Probability**	• **Mitigation Plan**
• 1	• Medium	JSON:API specification is used to mitigate concerns around API scalability, extensibility, and standardization practices. For more information on the JSON:API specification, please see https://jsonapi.org/. Additionally, for guidance on other microservice APIs in the TDEI system that support the interchangeable data infrastructure, but are not directly serving transportation data, we will reduce development risks through proper API **design**, taking guidance from the RESTful API Design Guide published by the Bank of Belgium found at: https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki. This resource provides guidance and mechanisms for API building supporting decision making about many of the issues named in this risk. Finally, for handling API registration and Security- we will be using Intermediary API Gateway Layers technology mentioned in 2.2.4 which addresses these concerns.

• Risk ID	• Risk Probability	• Mitigation Plan
• 2	• High	APIs will be hardened for cyber-security concerns using API Gateways. The gateway provides the enforcement capability that only trusted messages (authentication and authorization) can pass through– a control mechanism that requires that API callers have the appropriate identity, authentication, and security clearance. The Gateway prevents cyber-attacks by inspecting the messages passing through it. The Gateway provides API firewalling to only allow legitimate messages to enter an organization. **API Firewalling** helps to mitigate against application-level threats, such as cross-site scripting, SQL injection, command injection, cross-site request forgery, etc. The Gateway will detect and block threats. Additionally, **messages can be checked** to see if they might contain **viruses**. The Gateway provides multiple ways for API consumers to authenticate and get access to API resources. The Gateway can support one of the many open standards that means to determine the validity of an API Consumer (i.e., OAuth, JWT tokens, API Key, HTTP Basic/Digest, SAML, etc.) or it can use non-standard means to locate credentials in headers or payload of the message.
• 3	• High	APIs will be hardened for risks of traffic routing and load balancing using API Gateways. As the Gateway sits in the line of traffic, it provides basic load balancing and route trafficking capabilities (Round Robin, Weighted Round Robin, random, etc.) for traffic entering the organization. The Gateway provides various mechanisms for managing the rate of flow into an organization. It can protect the TDEI backend against severe traffic spikes and denial of service attacks. As it sits in the flow of traffic it can provide traffic throttling and smoothing. IP addresses can be white or blacklisted. Additionally, the Gateway provides various failure patterns, like a circuit breaker or retry policies, to help protect the organization from cascading failures.

• Risk ID	• Risk Probability	• Mitigation Plan
• 4	• High	To avoid having the API Gateway as the deployment bottleneck, TDEI will deploy at least two replicas of the gateways for high availability. If needed, we could scale out the replicas further, depending on the load. The gateway will be additionally run on a dedicated set of nodes to garner the following benefits: **Isolation.** All inbound traffic goes to a fixed set of nodes, which can be isolated from backend services. **Stable configuration.** If the gateway is misconfigured, the entire application will still be available. • **Performance.** Specific VM configurations for the gateway might be better for performance reasons.
• 5	• High	• To mitigate missing updates if data is suddenly published too fast for ingestion, and to maintain the "Best Available Data," it is necessary to couple state-establishing API calls with messaging or event streaming so that services can broadcast changes in state and other interested parties (i.e., APIs within the TDEI infrastructure) can listen for those changes and adjust accordingly. We can use a general-purpose message broker, but we believe that in our case, an event streaming platform, might be a good fit to enable future integration with other sensors and city IoT infrastructure, for example.
• 6	• High	• Given current event streaming architecture and large data volumes, achieving lossless delivery for data pipelines is cost prohibitive in cluster implementations (AWS EC2 or Azure). Accounting for this, an acceptable amount of data loss, while balancing cost. We must be able to monitor the daily data loss rate. Metrics are gathered for dropped messages so we can act if needed. • A mitigating pipeline will produce messages asynchronously without blocking applications. In case a message cannot be delivered after retries, it will be dropped by the producer to ensure the availability of the event log queue.

• **Risk ID**	• **Risk Probability**	• **Mitigation Plan**
• 7	• Medium	• Flexible Message routing: Most of the applications in Netflix use our Java client library to produce to event streaming pipeline. On each instance of those applications, there are multiple producers, with each producing to a cluster for sink level isolation. The producers have flexible topic routing and sink configuration which are driven via dynamic configuration that can be changed at runtime without having to restart the application process. This makes it possible for things like redirecting traffic and migrating topics across event streaming clusters. For non-Java applications, they can choose to send events to REST endpoints which relay messages to fronting clusters.
• 8	• Medium	• Message ordering: For greater flexibility, the producers do not use keyed messages. Approximate message ordering is re-established in the batch processing layer (in platforms like Hive / Elasticsearch) or routing layer for streaming consumers.
• 9	• High	• Orchestration Server stability: The TDEI will put the stability of the event streaming clusters at a high priority because they are the gateway for message injection. Therefore, we will not allow client applications to directly consume from them to make sure they have predictable load.
• 10	• High	• Unpredictable cloud life cycle: In the cloud, instances have an unpredictable life-cycle and can be terminated at any time due to hardware issues. Transient networking issues are expected. These are not problems for stateless services but pose a big challenge for a stateful service requiring brokers and a single controller for coordination. The TDEI will weigh the benefits and detractors of using these brokers.

Appendix A. Acronyms

This appendix includes a list of acronyms used in the document.

Acronym	Definition
ADA	Americans with Disabilities Act
AI	Artificial intelligence
AMQP	Advanced Message Queuing Protocol
API	Application program interface
ARNOLD	All Road Network of Linear Referenced Data
ASP	Application Service Provider
ATTRI	Accessible Transportation Technologies Research Initiative
BAA	Broad Area Announcement
CD	Continuous deployment
CI	Continuous integration
ConOps	Concept of Operations
DOI	Digital Object Identifier
DOT	Department of transportation
ET	Enabling technology
ETL	Extract, Transform, Load
ETRA	Enabling Technology Readiness Assessment
FHWA	Federal Highway Administration
GIS	Geographic information systems
GTFS	General Transit Feed Specification
GTFS-Flex	General Transit Feed Specification for flexible route services
GTFS-Pathways	General Transit Feed Specification for pathways through transit facilities
HTTP	Hypertext Transfer Protocol
ICAP	Internet Content Adaptation Protocol
IES-City	IoT-enabled smart city
IoT	Internet of things
ITS	Intelligent transportation system
LEP	Limited English Proficiency
LiDAR	Light Detection and Ranging
ML	Machine learning
.NET	The brand name of a proprietary software framework developed by Microsoft Corporation
NIST	National Institute of Standards and Technology
Npm module	any file or directory in the node_modules directory that can be loaded by the Node.js require() function

Acronym	Definition
OASC	Open and Agile Smart Cities
OSM	OpenStreetMap
PID	Personal identifier
QC	Quality control
REST	Representational State Transfer
RPC	Remote Procedures Call
SSL	Secure sockets layer
SyRS	System Requirements Specification
Taskar Center or TCAT	Taskar Center for Accessible Technology at the University of Washington
TCP	Transmission Control Protocol
TDEI	Transportation Data Equity Initiative
TRA	Technology Readiness Assessment
TRAC	Washington State Transportation Center at the University of Washington
TRL	Technology Readiness Level
U.S.	United States
U.S. DOT	United State Department of Transportation
UW	University of Washington
WAF	Web application firewall

Appendix B. References

This document utilizes information and processes defined in the following reports and web site. Reports are noted first, with websites noted below the reports.

Reports

- Accessible Transportation Technologies Research Initiative (ATTRI) Performance Metrics and Evaluation, Final Evaluation Framework Report, FHWA-JPO-20-784, https://rosap.ntl.bts.gov/view/dot/50748/.
- FHWA Technology Readiness Level Guidebook *https://www.fhwa.dot.gov/publications/research/ear/17047/17047.pdf*
- IES-City Framework. Available online: https://pages.nist.gov/smartcitiesarchitecture/ (accessed on 30 July 2021).
- Karpenko, A.; Kinnunen, T.; Madhikermi, M.; Robert, J.; Främling, K.; Dave, B.; Nurminen, A. Data Exchange Interoperability in IoT Ecosystem for Smart Parking and EV Charging. Sensors 2018, 18, 4404. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6308793/ (accessed on 30 July 2021)
- The Key to Mastering Kafka Streams and ksqlDB by Mitch Seymor, ISBN-10: 1492062499, March 16, 2021
- Kincaid, Jason (November 10, 2009). "Google's Go: A New Programming Language That's Python Meets C++". TechCrunch. Retrieved January 18, 2010.
- National Smart City Strategic Program. Available online: https://www.smartcities.kr/about/about.do#en_intercep (accessed on 30 July 2021).
- OASC. A Guide to SynchroniCity. Available online but requires registration through: https://www.smartcitiesworld.net/news/news/oasc-launches-guide-to-synchronicity-5089
- Phase 1 Concept of Operations (ConOps), University of Washington ITS4US Deployment Project, FHWA-JPO-21-861, https://rosap.ntl.bts.gov/view/dot/58675
- Phase 1 System Requirements Specification (SyRS), University of Washington ITS4US Deployment Project, FHWA-JPO-21-884, https://rosap.ntl.bts.gov/view/dot/60129
- Zhang, Yuxiang, Sachin Mehta, and Anat Caspi. "Collecting Sidewalk Network Data at Scale for Accessible Pedestrian Travel." *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, 2021

Websites

- What is a Microservices Architecture? Google. https://cloud.google.com/learn/what-is-microservices-architecture (extracted April 5, 2023)
- Microservice Architecture Style. Microsoft. https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices (extracted April 5, 2023)

- Microservices vs. Monolithic Architecture. Atlassian. https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (extracted April 5, 2023)
- IBM Cloud: What is an API? https://www.ibm.com/au-en/topics/api (extracted March 29, 2023)
- Mobility Data Interoperability Principles, https://www.interoperablemobility.org/ (extracted April 3, 2023)
- The Bezos API Mandate: Amazon’s Manifesto for Externalization, January 19, 2021: https://nordicapis.com/the-bezos-api-mandate-amazons-manifesto-for-externalization/
- National Bank of Belgium, REST-API Design Guide: https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki (extracted July 6, 2022)
- IBM Cloud: What are Microservices? = https://www.ibm.com/topics/microservices (extracted March 29, 2023)
- Learn Microsoft Azure Documentation: End-to-end computer vision at the edge for manufacturing: https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/ai/end-to-end-smart-factory (extracted April 5, 2023)
- Adam Drake, Enough with Microservices, https://adamdrake.com/enough-with-the-microservices.html (extracted April 3, 2023)
- IView Labs – 9 Key Points to Decide on Microservices Architecture - https://iviewlabs.medium.com/9-key-points-to-decide-on-microservices-architecture-c390d9827db7 (extracted April 5, 2023)
- Microservice Architecture – What are microservices? - https://microservices.io/ (extracted April 5, 2023)
- SHIFT Commerce's Journey: Deconstructing Monolithic Applications into Services - https://blog.heroku.com/monolithic-applications-into-services (extracted March 29, 2023)
- Clarion Technologies, 5 Best Technologies to Build Microservices Architecture, by Vinugayathri, https://www.clariontech.com/blog/5-best-technologies-to-build-microservices-architecture (extracted July 6, 2022)
- JetBrains The State of Developer Ecosystem 2022, Developer Survey Responses https://www.jetbrains.com/lp/devecosystem-2021/microservices/#:~:text=Primary%20languages%20among%20microservices%20developers&text=The%203%20most%20popular%20languages,%2C%20and%20Python%20(25%25). (extracted April 5, 2023)
- "Java Platform, Enterprise Edition (Java EE)". *Oracle Technology Network,* Oracle, https://www.oracle.com/java/technologies/java-ee-glance.html (extracted April 5, 2023)
- StackOverflow, Why use CDI in Java EE, https://stackoverflow.com/questions/13047807/why-use-cdi-in-java-ee (extracted March 29, 2023)
- Spring Software, https://spring.io/ (extracted April 5, 2023)

- Dropwizard is a Java framework for developing ops-friendly, high-performance, RESTful web services, https://www.dropwizard.io/en/latest/ (extracted April 5, 2023)
- Restlet Framework: https://restlet.talend.com/ (extracted April 5, 2023)
- Spark - A micro framework for creating web applications in Kotlin and Java 8 with minimal effort, https://sparkjava.com/ (extracted April 3, 2023)
- Surf Corporation: Microservices with Go: Why they are made for each other and what results you can get. https://surf.dev/why-golang-with-microservices/ (extracted April 3, 2023)
- New York Times, Gizmo GitHub Site: https://github.com/NYTimes/gizmo (extracted March 29, 2023)
- Go Kit, A toolkit for microservices, https://gokit.io/ (extracted April 5, 2023)
- Go-Micro, GitHub site, https://github.com/go-micro/go-micro (extracted April 5, 2023)
- Koding: Kite, GitHub site, https://github.com/koding/kite (extracted April 5, 2023)
- Tech Target, App Architecture, How viable is it to create a microservice in Python? https://www.techtarget.com/searchapparchitecture/tip/How-viable-is-it-to-create-microservices-in-Python (extracted April 5, 2023)
- DZone, Is Python Effective for Microservices Architecture? https://dzone.com/articles/is-python-effective-for-microservice-architecture (extracted April 5, 2023)
- Analytics Vidhya, An Easy Introduction to Flask Framework for beginners, by Ashray Saini, September 5, 2022, https://www.analyticsvidhya.com/blog/2021/10/easy-introduction-to-flask-framework-for-beginners/ (extracted April 5, 2023)
- Flask Development, One Drop At a Time, https://flask.palletsprojects.com/en/2.2.x/
- Phrase Corporation, An I18n Walkthrough for Falcon Web Apps in Python, https://phrase.com/blog/posts/falcon-python-i18n/ (extracted April 5, 2023)
- Bottle: Python Web Framework, https://bottlepy.org/docs/dev/ (extracted April 5, 2023)
- Nameko 2.12.0 Documentation, https://nameko.readthedocs.io/en/stable/what_is_nameko.html (extracted April 5, 2023)
- CherryPy – A Minimalist Python Web Framework, https://docs.cherrypy.dev/en/latest/ (extracted April 7, 2023)
- Wikipedia, Node.js, https://en.wikipedia.org/wiki/Node.js (extracted April 7, 2023)
- Mozilla Developer Platform: Express/Node Introduction, https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_nodejs/Introduction (extracted April 7, 2023)
- Node: About Node.js, https://nodejs.org/en/about (extracted April 7, 2023)
- Express 4.18.1, Fast, unopinionated, minimalist, web framework for Node.js, https://expressjs.com/ (extracted April 7, 2023)
- Sails: The MVC framework for Node.js, https://sailsjs.com/ (extracted April 7, 2023)
- Section: Introduction to hapi.js Framework, https://www.section.io/engineering-education/introduction-to-hapi/ (extracted April 7, 2023)

- Jevlix: ASP.NET vs ASP.NET Core. By Vitaliy Ilyukha, https://jelvix.com/blog/asp-net-vs-asp-net-core (extracted April 7, 2023)
- Microsoft .NET: Microservices with .NET, https://dotnet.microsoft.com/en-us/apps/aspnet/microservices# (extracted April 10, 2023)
- Learn Microsoft Azure Documentation: Use API gateways in microservices, https://learn.microsoft.com/en-us/azure/architecture/microservices/design/gateway (extracted April 5, 2023)
- Cloud Infrastructure Services: Kafka vs Pulsar – What's the Difference? (Pros and Cons), by Kamil Wisniowski, September 6, 2022, https://cloudinfrastructureservices.co.uk/kafka-vs-pulsar-whats-the-difference/
- Digitalis, Apache Kafka vs Apache Pulsar https://digitalis.io/blog/kafka/apache-kafka-vs-apache-pulsar/ (extracted April 7, 2023)
- Confluent: Kafka vs. Pulsar vs. RabbitMQ: Performance, Architecture, and Features Compared, https://www.confluent.io/kafka-vs-pulsar/ (extracted April 7, 2023)
- StreamNative, April 7, 2022, Apache Pulsar vs. Apache Kafka 2022 Benchmark https://streamnative.io/blog/apache-pulsar-vs-apache-kafka-2022-benchmark (extracted April 7, 2023)
- Macrometa: Kafka Alternatives, Chapter 4 of Event Stream Processing, https://www.macrometa.com/event-stream-processing/kafka-alternatives (extracted April 7, 2023)
- Ki Waegner: When to choose Redpanda instead of Apache Kafka? https://www.kai-waehner.de/blog/2022/11/16/when-to-choose-redpanda-instead-of-apache-kafka/ (extracted April 7, 2023)
- Infoworld, Review: Redpanda gives Kafka a run for its money, by Martin Heller, May 25, 2022, https://www.infoworld.com/article/3660628/review-redpanda-gives-kafka-a-run-for-its-money.html (extracted April 7, 2023)
- Tech with Adrian Bednarz, October 27, 2022, A closer look at Redpanda, https://techwithadrian.medium.com/a-closer-look-at-redpanda-37015edb0841 (extracted April 7, 2023)
- Medium.Com: Real-Time Streaming for Mortals: How Redpanda and Materialize Making It a Reality, October 20, 2021, https://medium.com/event-driven-utopia/real-time-streaming-for-mortals-how-redpanda-and-materialize-making-it-a-reality-18ac0bdc6f43 (extracted April 7, 2023)
- Source Forge: Apache Kafka vs. Redpanda Comparison Chart, https://sourceforge.net/software/compare/Apache-Kafka-vs-Redpanda/ (extracted April 7, 2023)
- Appinventiv: 20 Open-Source API Management Platforms to Add to Your Tech Stack, September 7, 2022, https://appinventiv.com/blog/open-source-api-management-tools/ (extracted April 7, 2023)

U.S. Department of Transportation
ITS Joint Program Office-HOIT
1200 New Jersey Avenue, SE
Washington, DC 20590

Toll-Free “Help Line” 866-367-7487
www.its.dot.gov

FHWA-JPO-21-889

U.S. Department of Transportation