# Data Analytics Plan – Phase 1

**JUNE 2025**

EMERALD COAST REGIONAL COUNCIL 2.0

DATA ANALYTICS PLAN
AN ECRC 2.0 INITIATIVE

Contents


Document Introduction ..... 1
Proof of Concept Architecture Overview ..... 1
Technology Solutions ..... 8
MioVision ..... 8
Median Travel Time ..... 8
Intersection HiRes data ..... 8
Intersection TMC data ..... 9
RITIS ..... 9
Region Explorer ..... 9
Massive Data Downloader ..... 9
Congestion Scan ..... 9
Corridor Speed Bins ..... 11
Corridor Time Comparison ..... 11
Trend Map ..... 12
Performance Charts ..... 12
Performance Summaries ..... 12
Bottleneck Ranking ..... 12
User Delay Cost Analysis ..... 12
Dashboard ..... 12
Travel Time Delta Ranking ..... 13
Travel Time Comparison ..... 13
Temporal Comparison Maps ..... 13
Causes of Congestion Graphs ..... 13
On-Premises Data Lake ..... 13
Technical Architecture Overview ..... 13
Structured Data ..... 13
Semi-Structured Data ..... 14
Unstructured Data ..... 20
Cloud Data Lake/Warehouse Solutions ..... 22
Azure ..... 22
Azure Data Lake Storage ..... 22
Azure DataBricks ..... 26
Azure Synapse Analytics ..... 29
Azure Data Factory ..... 33
Azure Ecosystem Flexibility ..... 37
Appendix A: API Tables ..... 39

# Document Introduction

This document is intended to serve as a resource for developing a Data Analytics platform that will connect the technology infrastructure in the Smart Regions Master Plan to the Regional Transportation Management Center (RTMC). This living document will cover existing technology solutions currently deployed by the Region and may change over time as new solutions are introduced to the transportation system and RTMC. It will also detail various "data lake" solutions that can be created in the RTMC to house data from technology solutions. The goal is to make this data available to the region and empower stakeholders to create dashboards for data analysis.

The process for this project started with interviews with private and public stakeholders. Once the project was kicked off, staff moved onto completing tasks en route to our proof of concept. While working on the proof of concept, staff presented at monthly traffic meetings to track progress and gain feedback on progress towards a completed proof of concept.

This document is organized into four sections:

- Proof of Concept Architecture Overview: Details the software solutions used to achieve the result in the proof of concept.
- Technology Solutions: Discusses the client-side technology that allows us to collect data from controllers out in the field.
- On-Premises Data Lake: Evaluates possible software solutions our team can use in the case of an on-premises data lake.
- Cloud Data Lake/Warehouse Solutions: Evaluates possible software solutions our team can use in the case of a cloud-based data lake.

# Proof of Concept Architecture Overview

For Phase I of the project, the team developed a proof-of-concept data lake specifically designed to scale and adapt to ECRC's requirements. This initial version includes automated data pipelines that streamline data flow into and out of the data lake, ensuring continuous and seamless data integration. Additionally, the data lake supports various analytics options—ranging from intuitive dashboards and visualizations to more sophisticated real-time analytics, big data processing, and machine learning applications, helping ECRC make informed, data-driven decisions.

This foundational proof-of-concept also provides a robust system capable of expanding as requirements grow, offering flexibility to scale and adapt according to evolving operational needs.

**Data Pipelines**

Three components are required to build a data pipeline:

1. A data sender
2. A data receiver
3. A common communication protocol

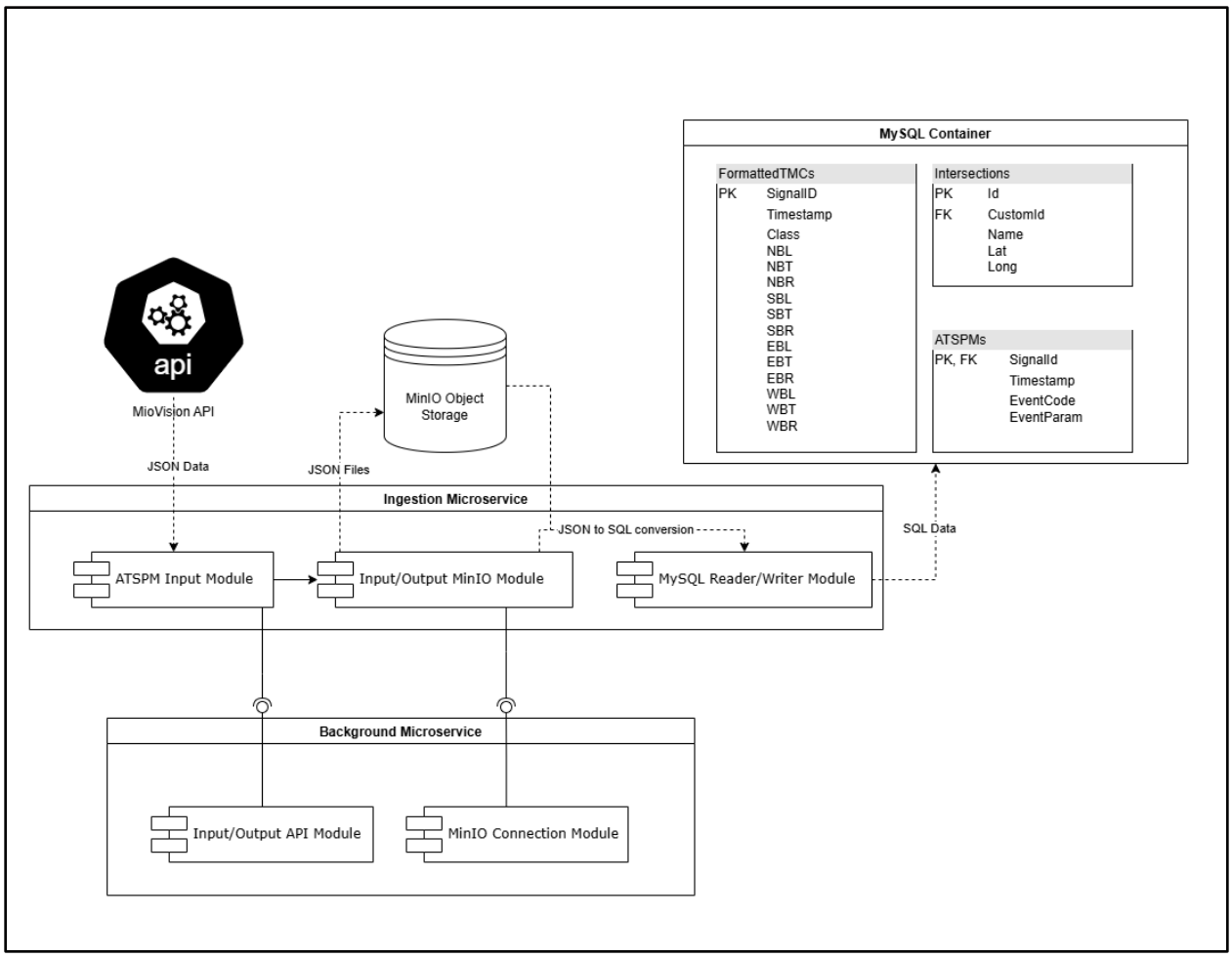
The sender includes cameras and controllers using MioVision software in this proof of concept. The data lake acts as the receiver, collecting and storing this information. An Application Programming Interface (API) was introduced to ensure a common communication protocol.

An API sets clear guidelines for exchanging information between two different systems or devices. It enforces a set of rules telling each system how to request data and how to respond clearly. Using these rules, the MioVision cameras and controllers can send data directly to the data lake without errors.

APIs often use common web methods—like clicking links or submitting forms online—to send and receive information. They usually format data in simple, easy-to-read formats like JSON or XML. This standardization makes the data reliable and easy to handle.

Additionally, APIs have built-in ways to keep communication secure using encryption and user validation, ensuring only approved systems or people can access sensitive data. They also make adding new equipment or changes easier since every new device can use the same clear set of communication rules.

The API is the common communication protocol in this proof of concept. An API serves as the entry and exit point for data within the data lake. Its common web methods allow system configurators to quickly and easily configure current services or add more if needed. APIs are also not storage type specific. As the system requirements expand, the API can currently meet those needs or quickly be expanded to create a new pipeline to include that functionality.




## Power BI Dashboard

This Power BI dashboard visualizes processed Turning Movement Count (TMC) and intersection data from the MioVision API. The dashboard features:

- A geospatial map displaying intersections across Pensacola, including areas like East Milton, Ensley, Brent, Ferry Pass, and West Pensacola
- Time filtering capabilities with 15-minute interval timestamps
- Detailed movement counts broken down by direction at the highlighted intersection

Directional traffic flow metrics include:

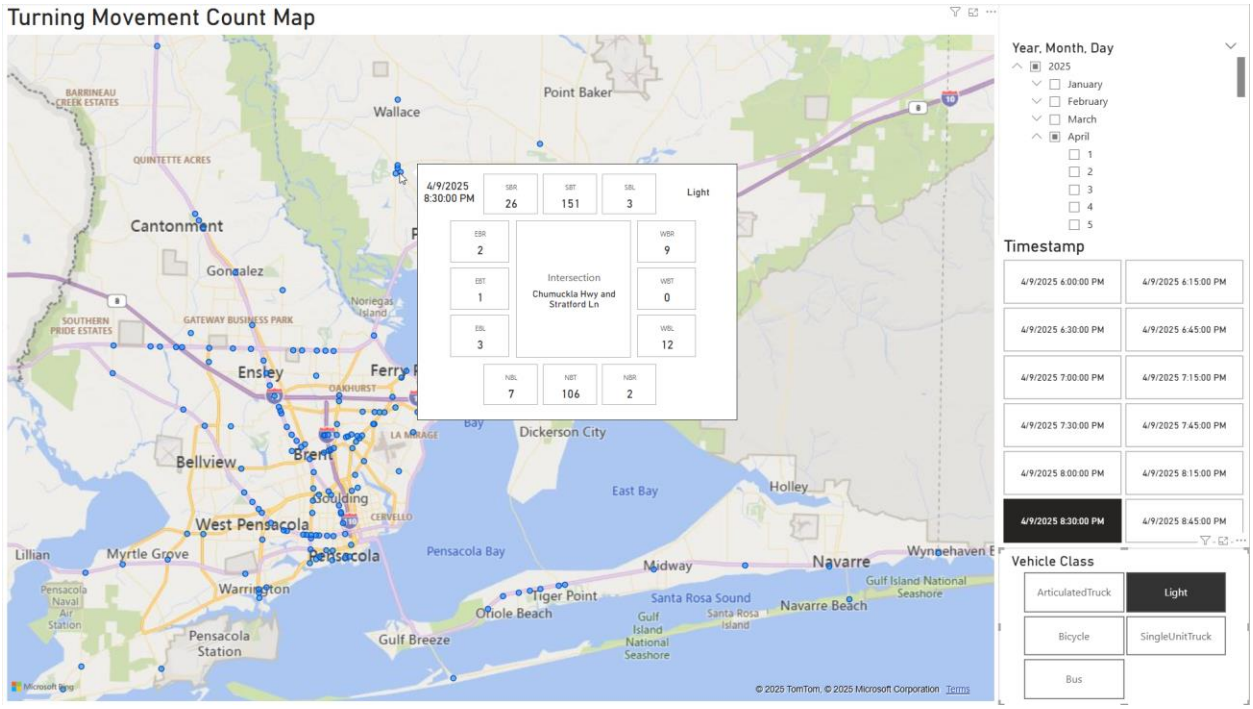
- Northbound movements
- Eastbound movements
- Westbound movements
- Southbound movements

Vehicle types actively being recorded by MioVision Smart Sense Cameras:

- Pedestrians

- Bicycle
- Light (motorcycles, cars & pickup trucks)
- Work van
- Single unit truck
- Articulated Truck
- Bus

The dashboard will enable transportation analysts to filter by timestamp, day, and vehicle class (light to analyze traffic patterns and movement counts at specific intersections. This data visualization tool helps traffic engineers and urban planners make informed decisions about signal timing, road capacity, and infrastructure improvements based on actual traffic volumes by direction and time period.




*Power BI Dashboard using Miovision data*

## Unstructured Data

### MinIO

MinIO is a high-performance, open-source object storage system optimized for cloud-native applications but can also be deployed for on-premises storage. It is widely used for building scalable and efficient storage infrastructure in enterprises. MinIO is the entry point for data coming into the pipeline.

When building this part of the service for proof of concept, an emphasis was put on scalability and flexibility. MinIO's "object storage" framework allows for both. Unlike a database, object storage does not need to know what format the data is in to successfully store a data "object." This allows MinIO to serve as a catch-all for any data entering the pipeline from the API.

## Structured Data

### MySQL

MySQL was chosen for data storage of structured data; SQL, short for Structured Query Language, is a data storage framework that allows large amounts of data to be stored and processed quickly and efficiently. Several popular variations of SQL databases exist, including Microsoft SQL Server, PostgreSQL, and MySQL. After factoring in price and ease of use, MySQL was chosen as the platform for structured data storage.

The main benefits to any SQL provider over a comparable technology like Excel are its integration with SQL and having no limit on the number of rows in a single table. Using SQL, you can iterate over and process millions of rows of data in a short timeframe with very little code. The database structure of MySQL also benefits the implementation by ensuring data can be stored safely and securely.

## Deployment and Maintenance

### Docker

This proof-of-concept data lake's deployment and maintenance process has been simplified using a software deployment process called containerization. Containerization breaks the components of the data lake into smaller chunks known as "containers" that each house their own individual services. Each container serves its own purpose, orchestrated and connected through an internal network that provides secure communication channels between services.

Docker is an industry-standard software that handles the containerization of this data lake. Using Docker, three containers have been created: The API ingestion service that pulls data from the MioVision API, MinIO Object Storage, and MySQL tabular storage.

Containerizing each piece of the data lake has many benefits. Docker allows administrators to update software at will without needing large maintenance windows. Using the Docker framework, each update can be pushed to the application repository by developers, engineers, and other power users and pulled down by others from the repository, allowing for components to be easily shared and grown upon.

There are also benefits in system uptime by using Docker containers. Containers are designed to operate independently from each other, making errors easier to identify and resolve without bringing the whole system down. As a result, containers have a higher tolerance for outages. Using orchestration software such as Docker, services can be told to start up or shut down in certain conditions. Lastly, containers are very easy to scale due to the fact they are virtualized within their host environment. There are no hardware

requirements to run a certain number of containers. They will operate within the framework of the host environment.

### Data Sources

The Advanced Traffic Signal Performance Measures (ATSPM) system collects various data to analyze traffic signal performance and improve overall traffic flow. The data format typically includes:

1. **Signal Phase and Timing Information**: Details about the signal phases, timing intervals, and change intervals for each traffic signal.
2. **Traffic Volume Data**: Counts of vehicles, pedestrians, and cyclists at intersections are often collected in real-time.
3. **Delay Data**: Information on the delay experienced by vehicles at intersections, which helps identify inefficiencies.
4. **Traffic Flow Trends**: Patterns such as peak traffic hours and seasonal variations over time.
5. **Signal Compliance**: Data on how well vehicles comply with signal changes, including instances of red-light running.
6. **Incident Data**: Information about traffic incidents or disruptions affecting signal performance.

This data is typically stored in a structured format, allowing for analysis and reporting, often leveraging databases and visualization tools to assist traffic engineers in making informed decisions.

The exact format that ATSPM data is collected in the format below.

| Column Name | Data Type | Nullable |
|---|---|---|
| SignalID | String | No |
| Timestamp | DateTime | No |
| EventCode | Int | No |
| EventParam | int | No |

1. **SignalID**: This unique identifier specifies which traffic signal is being referenced. It allows for precise tracking of performance and events at specific intersections, facilitating targeted analysis and reporting.
2. **Timestamp**: This data point records the exact date and time (down to the millisecond) when an event occurs. Accurate timestamps are crucial for analyzing traffic patterns, assessing signal timing effectiveness, and correlating events with traffic conditions.

3. **EventCode**: This code represents the specific event being recorded by the traffic signal controller, such as phase changes, malfunction alerts, or manual overrides. It categorizes the nature of the event, enabling traffic engineers to identify trends and issues in signal operation.
4. **EventParam**: This parameter indicates the specific phase of the traffic signal during which the event occurs (e.g., green, yellow, or red). Understanding the phase context helps evaluate signal performance, compliance, and the impact of various phases on traffic flow and safety.

Together, these data points provide a comprehensive view of traffic signal operations, allowing for effective analysis and improvements in traffic management strategies.

Controllers recording these events take second by second measurements, depending on the number of controllers.

| Event Code | Event Descriptor |
|---|---|
| 0 | Phase On |
| 1 | Phase Begin Green |
| 2 | Phase Check |
| 3 | Phase Min Complete |
| 4 | Phase Gap Out |
| 5 | Phase Max Out |
| 6 | Phase Green Termination |
| 7 | Phase Green Termination |
| 8 | Phase Begin Yellow Clearance |
| 9 | Phase End Yellow Clearance |
| 10 | Phase Begin Red Clearance |
| 11 | Phase End Red Clearance |
| 12 | Phase Inactive |
| 21 | Pedestrian Begin Walk |
| 43 | Phase Call Registered |
| 45 | Pedestrian Call Registered |
| 81 | Detector Off |
| 82 | Detector On |

### Technical Summary

There are two types of data storage used in this proof of concept: Structured (database tables) and Unstructured (files). A separate module was used to store each kind of data. To facilitate data entry into the lake, the API ingestion tool was developed. This tool serves as a connection between the MioVision API and the unstructured data module, as well as a connection between the unstructured data module and the structured data module. For the modules to easily operate together, a packaging system was used. Docker is the industry-standard software used to package all these modules to ensure they can communicate with each other efficiently and securely. Lastly, a Power BI dashboard was developed to present the data in a human readable format.

# Technology Solutions

This section describes two platforms leveraged by the region that can be integrated with the RTMC. This is intended to be a living document that is expanded upon as new technologies are incorporated to the region.

## MioVision

MioVision is currently used for FL-AL TPO and has 338 signals with MioVision hardware installed and configured in the MioVision ONE UI/Web application. This web application lets users view real-time data from MioVision hardware, such as Intersection Health Metrics, insights, and analysis. Users can also access live camera feeds and detector data directly from the website.

The MioVision API has many different endpoints related to Alerts, Cameras, Diagnostics, Intersections, Organization information, Priority Requests, TMC information, Median Travel Time, and User information.

### Median Travel Time

The Median Travel time endpoint allows the retrieval of Travel Time between the specified intersections for a 24-hour period, aggregated into 15-minute bins.

By default, results are aggregated into 1-minute bins. This can be changed by adding a binSize parameter (Integer) to the query string, changing the number of minutes. The unit of time can be changed by adding a binUnits parameter (String) to the query string. Valid bin units include: [Minute, Hour, Day].

By default, the 24-hour period ends at the time the request is submitted. The end of the 24-hour period can be specified with the endDateTime parameter.

### Intersection HiRes data

Returns telemetry data from the intersection in the HiRes Data Format. This data can be collected in a maximum of one-hour increments.

### Intersection TMC data

We can retrieve full Turning Movement Count data (Count, Direction of travel, class, and detected movement) for each individual intersection for a two-day period with a variable minute/hour/day binsize with a maximum period of 48 hours. MioVision SmartSense (vehicle detection software running ON can distinguish vehicle Classes into [Pedestrian, Bicycle, Light, WorkVan, SingleUnitTruck, ArticulatedTruck, Bus] buckets, which can also be specified in the query. This data can be manipulated in code as a json OR downloaded as a CSV file via another CSV specific endpoint. For a lane granular retrieval of data, there is a lane specific endpoint that specifies the *lane* of arrival of vehicles (No CSV download for this). There is also a similar endpoint for data relating to pedestrian crossing TMC data. This endpoint retrieves counts, directions of travel, and classes (bike or ped) in similar bins with a max period of 48 hours.

Additional details can be found in Appendix A.

## RITIS

RITIS, or the Regional Integrated Transportation Information System, is a collaborative platform that provides real-time and historical data on transportation systems. It is used by transportation agencies, public safety officials, and other stakeholders to manage and analyze traffic conditions, incidents, and infrastructure performance. RITIS aggregates data from various sources, including traffic sensors, cameras, and GPS, offering tools for data visualization, analysis, and decision support to improve transportation system efficiency and safety. The platform is particularly valuable for regional coordination and incident management. The following feature descriptions and example images will briefly describe the capabilities of the PDA (probe data analytics) Software Suite.

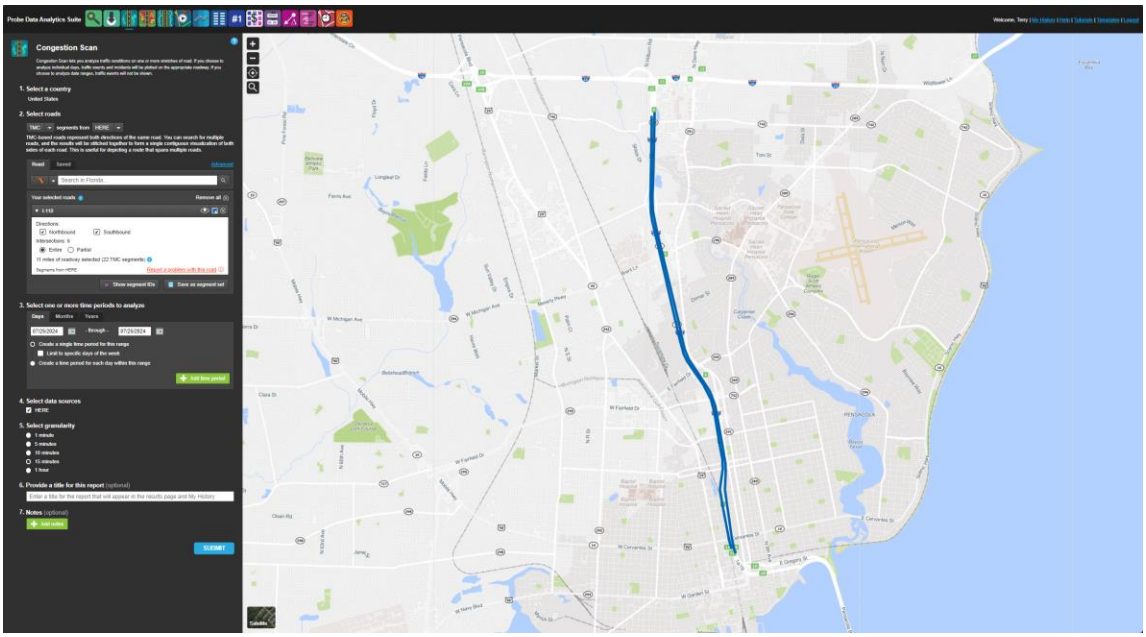
### Region Explorer

The Region Explorer tool shows the relationship between bottlenecks and traffic events on roadways and their impact on traffic. This tool works on both active real-time events and historical events.

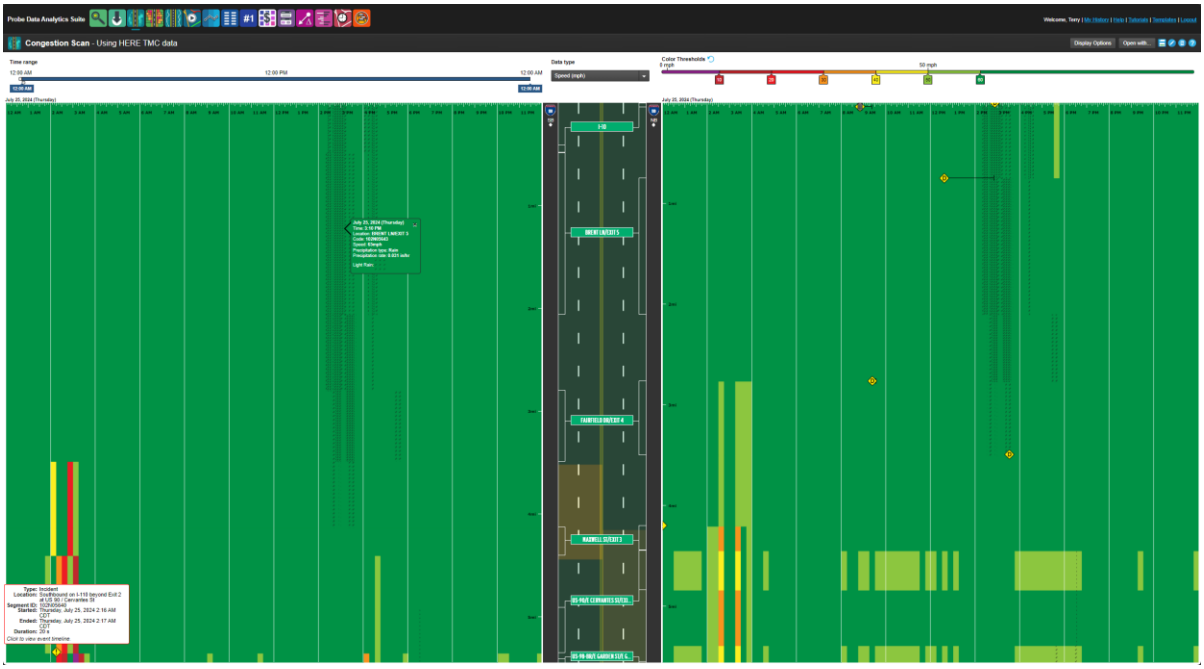
### Massive Data Downloader

The Massive Data Downloader allows users to download CSV probe data in bulk. You can customize your reports by choosing which roads, dates, and time periods to include and how to aggregate the data.

### Congestion Scan

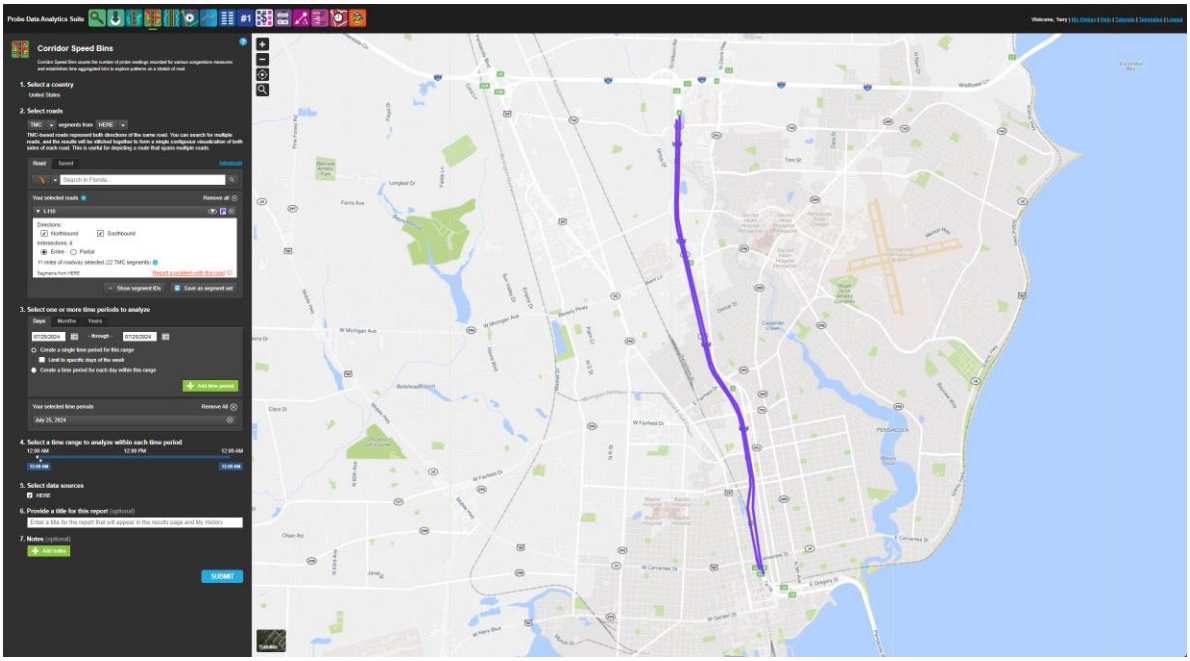



The Congestion Scan tool allows users to analyze real-time and historical traffic conditions along a set stretch of road. This tool enables users to select types of road segments, different probe data providers, date and time for analysis, and the granularity of data and roads. Users can run bulk reports simultaneously by selecting multiple roads and time periods rather than running multiple reports back-to-back.

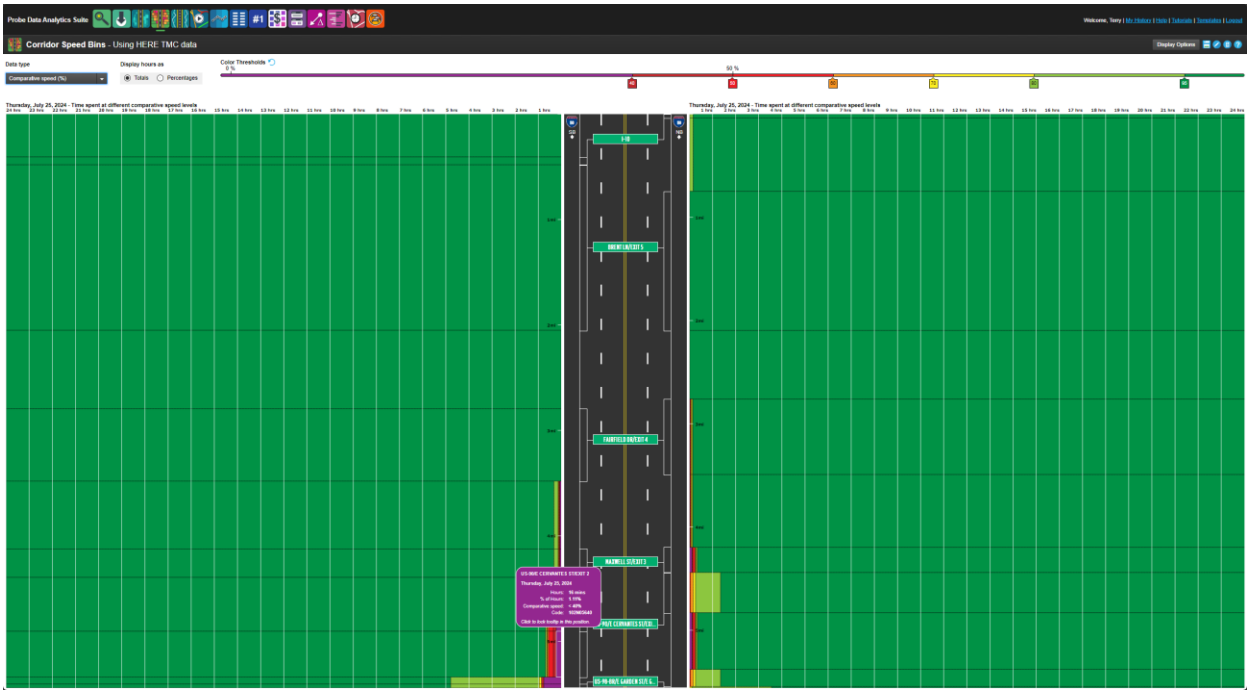



Daily reports allow users to view incident statistics and traffic conditions throughout the day and how they affect traffic along a roadway.

## Corridor Speed Bins




The Corridor Speed Bins tool enables users to count probe readings related to congestion metrics. This data is then aggregated into bins and displayed along the corresponding sections of the road, providing a clear view of congestion patterns.




## Corridor Time Comparison

The Corridor Time Comparison tool visualizes congestion along road corridors by displaying bi-directional line charts for selected road segments. It combines elements of performance

charts and congestion scans, allowing users to compare traffic data across one or two date ranges and up to seven different times of day, totaling 14 possible combinations. Users can analyze various metrics, including speed, congestion, travel time index, buffer index, and planning time index, providing a comprehensive view of traffic patterns.

### Trend Map

The Trend Map tool generates animated maps that visualize metrics based on raw speed data and performance metrics. Each map is accompanied by a chart that displays the distribution of speeds over the corresponding time periods.

### Performance Charts

The Performance Charts tool includes bar, line, plot, and candlestick charts that display aggregate conditions across road sections. These charts can be organized by time period or road direction, with TMCs in the same direction, even on different roads, combined into direction-based charts.

### Performance Summaries

The Performance Summaries tool enables the creation of reports on performance metrics by day of the week, weekdays, and weekends for specific road stretches. Users can select roads, date ranges, data sources, and time periods, then analyze the results using tabs for different time periods and road directions. The summaries include metrics like buffer time and buffer index and can be exported to Excel, saved as screenshots, or viewed in other tools within the Probe Data Analytics Suite.

### Bottleneck Ranking

The Bottleneck Ranking tool helps identify, rank, and explore traffic bottlenecks on roadways. Three metrics—Magnitude of Speed Drops, Severity of Congestion, and Estimated Total Delay—allow users to weigh the traditional base impact metric for better insights. The default table view is sorted by base impact weighted by total delay, but users can re-sort by other metrics.

### User Delay Cost Analysis

The User Delay Cost Analysis tool combines probe speed data with volume data to estimate the cost of delays caused by congestion.

### Dashboard

The Dashboard tool lets you create and manage collections of widgets to monitor roadway performance. Upon first use, you'll name your dashboard and can create multiple dashboards as needed. Available widgets include Speed and Travel Timetable, Ranked Bottleneck Table, Reliability Table, MAP-21, User Delay Cost Table, Ranked Bottleneck Comparison, Event Count, Clearance Time, and Energy Use and Emissions Table.

### Travel Time Delta Ranking

The Travel Time Delta Ranking tool enables users to rank and compare performance changes in corridors between two time periods. It is designed for before-and-after analysis of corridor performance improvement projects but can also be used to analyze changes along multiple roadways between any two time periods.

### Travel Time Comparison

The Travel Time Comparison tool lets users compare travel time distributions between two selected corridors during specified times of day.

### Temporal Comparison Maps

The Temporal Comparison Maps tool allows users to analyze performance metrics for road segments across selected time periods and ranges. You can choose up to 7 time periods and ranges, visualize metrics on maps, and compare deltas between them. The tool offers options to select metrics, compare by dates or time ranges, and choose specific maps or delta comparisons. Hovering over a road segment displays a description and metric value, and you can choose a base period for comparison based on your selected options.

### Causes of Congestion Graphs

The Causes of Congestion Graphs tool helps identify and quantify the impact of different congestion causes based on user delay cost. The primary categories include recurrent, weather, work zone, incident, signals, holiday, multiple causes, and unclassified.

# On-Premises Data Lake

## Technical Architecture Overview

A data lake is a centralized repository that allows you to store all your semi-structured and unstructured data at any scale. You can store your data as-is without having to first structure the data and run different types of analytics—from dashboards and visualizations to big data processing, real-time analytics, and machine learning to guide better decisions.

### Structured Data

Data Warehouses like Tableau and Snowflake are excellent for storing and accessing structured data because they are optimized for performance, scalability, and complex analytical queries. In a data warehouse, structured data is stored in well-defined tables with a strict schema, allowing fast SQL-based queries and deep analysis. These platforms excel at delivering quick insights from structured datasets, particularly for business intelligence (BI) applications like Tableau, where users can visualize and interact with data efficiently.

However, to integrate structured data with the broader range of formats handled by data lakes, structured data may need to be converted to semi-structured formats like JSON, Avro, or Parquet. This conversion allows structured data to coexist with semi-structured (e.g., logs, JSON) and unstructured data (e.g., images, videos) in the flexible environment of a data lake.

Data lakes store vast amounts of data cost-effectively, but they lack the optimized querying performance of data warehouses.

The emergence of data lakehouses bridges the gap between these two worlds. A data lakehouse combines the best aspects of both data warehouses and data lakes—offering the performance and structure of a warehouse while maintaining the flexibility and scalability of a lake. This unified architecture allows for easy access to structured data with SQL queries while simultaneously supporting the semi-structured and unstructured data formats common in data lakes. It essentially merges high-performance analytics with the storage flexibility of a lake, providing a single solution for all data types.

## Semi-Structured Data

Semi-structured data in a data lake is typically stored in a way that maintains its organization while benefiting from the scalability and flexibility of the lake. Data lakes often store structured data in open, standardized file formats optimized for performance and analytics.

File formats:

- CSV (Comma-Separated Values): CSV is a simple, widely used file format for storing tabular data in plain text. Each line in a CSV file represents a row in the table, and data fields in that row are separated by commas (or other delimiters). CSV is popular for data exchange and lightweight data storage, with the following key features:
    - Simplicity: CSV is straightforward, consisting of plain text files where each line corresponds to a record, and each field is separated by a comma (or another delimiter such as a semicolon, tab, or space). This simplicity makes it easy to understand, create, and edit manually.
    - Human-Readable: CSV files are human-readable and can be opened and edited in a basic text editor or spreadsheet software like Microsoft Excel or Google Sheets without the need for specialized software.
    - No Data Types: CSV doesn't inherently support data types or schema. All data is stored as plain text, which may require additional parsing and validation when imported into a database or other software.
    - Schema-Free: CSV files do not enforce any structure or schema, making them flexible but also requiring external handling of data validation and type enforcement.
    - Portability: CSV files are portable and compatible across many different systems, platforms, and applications. They are often used for data exchange between programs, databases, and systems that don’t require complex data structures.
    - No Compression: CSV files are plain text and typically do not support built-in compression, which can result in larger file sizes compared to binary formats like Avro or Parquet.

- Lack of Advanced Features: CSV lacks support for advanced features like schema evolution, complex data types (nested structures), or metadata. This makes CSV less suitable for big data use cases or complex data storage requirements.
- Wide Software Support: CSV is supported by virtually all data-processing tools, including databases, programming languages, and business applications. Its simplicity and ubiquity make it a common format for exporting and importing data between different systems.
- Limited Performance for Large Datasets: CSV stores data in a row-based format and lacks the optimizations seen in columnar formats like Parquet or ORC. For large datasets, CSV can be inefficient in terms of storage and query performance, especially for analytical workloads where only specific columns are needed.
- No Data Validation: CSV does not include built-in mechanisms for validating the consistency or correctness of the data, meaning additional tools or scripts are often required for data cleaning and validation.
- File Size and I/O: Since CSV is a plain text format, it tends to be less efficient in terms of file size and I/O performance, especially when handling large datasets.
- CSV is a highly portable and accessible file format, ideal for lightweight data exchange and simpler use cases, but its lack of advanced features and performance limitations make it less suitable for complex data storage and analysis in big data environments.

- Apache Parquet: Apache Parquet is a columnar storage file format designed to efficiently store and process large-scale data, primarily in distributed systems like Hadoop and Apache Spark. Parquet offers several important features, including:
  - Columnar Storage: Parquet stores data in a columnar format, which optimizes query performance by allowing systems to read only the specific columns needed rather than the entire dataset. This makes Parquet ideal for analytical workloads.
  - Schema Evolution: Parquet supports schema evolution, enabling changes to the schema over time without breaking compatibility with previous versions of the data.
  - Efficient Compression: Parquet uses efficient compression techniques at the column level, such as Snappy, Gzip, or ZSTD. Columnar compression typically provides better compression ratios compared to row-based formats because similar data types are stored together.

- Predicate Pushdown: Parquet allows queries to filter data at the storage level (predicate pushdown), reducing the amount of data read from storage and improving performance in data retrieval operations.
- Splitting Large Files: Parquet stores data in chunks called row groups, enabling efficient reading and parallel processing of specific file sections without reading the entire dataset. This is especially useful for distributed data processing frameworks.
- Data Types: Parquet supports complex data types (like nested structures, arrays, and maps), which makes it highly versatile for various kinds of structured and semi-structured data.
- Interoperability: Parquet is highly interoperable across different big data tools and frameworks like Apache Spark, Hadoop, Hive, and Impala, making it a popular choice for big data processing.
- Portability: Parquet files can be shared and used across different systems without any issues related to schema interpretation or formatting.
- Performance: Parquet's columnar storage model significantly improves read performance for analytical queries, as it reduces I/O by allowing selective reading of columns, resulting in faster data retrieval.
- Ease of Use: Parquet is easy to use, integrate, and validate in modern data platforms, and it is a go-to format for data warehousing, big data analytics, and machine learning workloads.
- Parquet is often compared with other big data formats like Avro and ORC, but its columnar format is particularly well-suited for analytical use cases.

- Apache Avro: Apache Avro is a file format that stores data in a binary format while using JSON to define the data's structure. This format has several key features, including:
  - Schema evolution: Avro supports changing data schemas over time.
  - Portability: Avro files are portable and can be read by different systems without needing external schema references.
  - Block storage: Avro stores data in blocks, allowing specific file sections to be read without reading the entire dataset.
  - Compression: Avro supports compression codecs to optimize storage and performance.
  - Serialization encodings: Avro supports two serialization encodings: binary and JSON. Binary encoding is faster and smaller, while JSON encoding may be better for debugging and web applications.

- Markers: Avro files include markers that can split large data sets into subsets for Apache MapReduce processing.
- Ease of use: Avro is easy to use and validate.
- Avro is a popular file format for big data management, along with Parquet and ORC.

Storage options:

- Hadoop Distributed File System (HDFS) for On-Premises Local Storage: Hadoop Distributed File System (HDFS) is one of the most popular storage solutions for managing large datasets in a distributed environment. It's designed for big data processing in distributed clusters but can also be used for on-premises local storage in enterprise environments. Here are some of the advantages that HDFS provides:
  - Scalability: HDFS can easily scale horizontally by adding more nodes to the cluster. This makes it ideal for on-premises environments where data grows over time. (Nodes are just computers/hardware accessible on a common network).
  - Fault Tolerance: With built-in replication, HDFS ensures that data is highly available, even if individual nodes fail. Data is automatically replicated across different nodes to avoid loss.
  - High Throughput: HDFS is optimized for high-throughput data access, especially for read-heavy workloads. It works well for batch processing of large files in formats like Parquet, Avro, and CSV.
  - Supports Large Files: HDFS is designed to handle large datasets, which makes it perfect for storing and processing big data, including structured, semi-structured, and unstructured data.
  - Cost Efficiency: HDFS runs on commodity hardware, reducing the overall storage cost. It's a good solution for organizations that want to avoid the high cost of proprietary storage solutions.
  - Parallel Processing: When combined with processing engines like Apache Hadoop MapReduce or Apache Spark, HDFS allows for efficient parallel processing of large datasets.
  - Schema Flexibility: HDFS can store data in a variety of formats (CSV, Parquet, Avro), making it versatile for different types of workloads and applications.
- MinIO for On-Premises Object Storage: MinIO is a high-performance, open-source object storage system that is optimized for cloud-native applications but can also be deployed for on-premises storage. It is widely used for building scalable and efficient storage infrastructure in enterprises. Below are some of the key advantages of MinIO:

- Scalability: MinIO can scale seamlessly, both vertically and horizontally, by adding more drives or nodes. This makes it an ideal choice for growing on-premises storage needs.
- Fault Tolerance: With erasure coding and bitrot detection, MinIO provides built-in data protection, ensuring data integrity and availability even during hardware failures.
- High Performance: MinIO is optimized for high-throughput, low-latency data access, making it well-suited for workloads like machine learning, analytics, and large-scale content delivery.
- Supports Large Objects: MinIO can store large objects (up to 5TB per object) and is designed to handle massive datasets, making it ideal for big data, media files, and backups.
- Cost Efficiency: MinIO can be deployed on commodity hardware, enabling cost-effective storage solutions for organizations that need scalable, high-performance object storage without incurring significant hardware costs.
- S3 Compatibility: MinIO offers full compatibility with Amazon S3 API, making it easy to integrate with a wide range of applications and tools that support S3-based storage
- Multi-Cloud and Hybrid Deployments: MinIO can be deployed across private, public, and hybrid cloud environments, providing flexibility and enabling seamless data mobility between different storage systems.
- Security: MinIO provides enterprise-grade security features such as encryption, identity management, and policy-based access control, ensuring that stored data remains secure and compliant with regulatory requirements.

Accessing Structured Data from S3: The process of retrieving structured data from s3 buckets is a multistep process requiring an S3 connector and distributed SQL query engine. There are a few different technologies that facilitate users to make a direct connection to the structured s3 data. The most used software is Hive-Metastore (part of the Apache Hadoop ecosystem), in tandem with Trino, Dbeaver, and PostgreSQL. Since structured data is compressed into storage file types that efficiently store data, we need to extract the associated data and metadata.

- Trino: Trino is a high-performance, distributed SQL query engine that queries large datasets across various data sources. It allows users to run SQL queries on data stored in different systems (such as data lakes, databases, and object storage systems) without moving or copying the data into a single location. Key aspects of Trino include:
  - Distributed Query Processing: Trino processes queries in parallel across a distributed cluster of machines, which allows it to handle large-scale datasets with high performance.

- Data Source Flexibility: Trino can query data from multiple types of sources such as Hadoop Distributed File System (HDFS), Amazon S3, Google Cloud Storage, traditional databases (like MySQL and PostgreSQL), and NoSQL systems like Cassandra and MongoDB. It does this without importing or duplicating data into a separate data warehouse.
- SQL Compatibility: Trino supports standard SQL, allowing users to write complex queries to join, aggregate, and manipulate data from different sources in a familiar language.
- Federated Queries: With Trino, you can run federated queries, meaning you can query data from multiple data sources in a single SQL query. For example, you can join a table from an S3 data lake with another table from MySQL without moving the data.
- Schema-on-Read: Trino uses a schema-on-read approach, meaning it reads and interprets the data schema at the time of the query rather than requiring predefined schemas, which makes it well-suited for working with semi-structured and unstructured data.
- Open-Source: Trino is an open-source project with a large, active community. It evolved from a previous project called Presto, and it is maintained and developed independently.
- Integration with Big Data Tools: Trino integrates well with other big data tools like Apache Hive, Apache Kafka, Apache HBase, and Apache Iceberg, making it a key component in modern data architectures.
- High Performance: Trino is designed for interactive querying, making it faster than traditional batch-processing systems for running ad-hoc queries over big data. It is particularly useful for analytics and business intelligence applications where low-latency querying is important.

- Hive Metastore + Trino: The Hive Metastore is a centralized repository for storing metadata about tables, databases, and data structures. Initially designed for Apache Hive, it plays a crucial role in modern data platforms by allowing various data processing engines to access shared metadata. In the context of Trino, a high-performance distributed SQL query engine, the Hive Metastore manages schema information and enables efficient data querying across different storage systems. Here are some of the key advantages of using Hive Metastore with Trino:
  - Centralized Metadata Management: Hive Metastore acts as a single source of truth for metadata, enabling multiple file systems like Hive Trino and Spark to access consistent information about the underlying data. This eliminates the need for each system to maintain its own metadata.

- Schema Management: Hive Metastore stores schema details, making it easier for Trino to query structured and semi-structured data stored in the above formats (Parquet, ORC, and Avro) without manually defining schemas.
- Interoperability: Trino can query data from various storage systems using the Hive Metastore, which allows access to data from multiple different s3 locations/platforms.
- Partitioned data handling: Hive Metastore tracks partitioned data, one of the main performance benefits of using Trino. Trino can efficiently scan only relevant portioned data, reducing the processing time and load during queries and improving execution speed.
- Scalability: As datasets grow, the Hive Metastore supports large-scale data environments, allowing Trino to work with distributed data across clusters while maintaining performance and scale.
- Compatibility: Hive Metastore's compatibility with different file formats and storage systems ensures that Trino can integrate with various data lakes, warehouses, and other big data ecosystems.
- Ease of Data Management: With Hive Metastore, managing table versions, schema evolution, and data organization becomes straightforward, providing Trino users with an organized and efficient way to query and manage large datasets.
- Security and Access Control: When used with Trino, Hive Metastore supports fine-grained access control and security features, ensuring that metadata and the underlying data are protected and accessed securely based on user permissions.

## Unstructured Data

Unstructured data refers to information that does not adhere to a predefined data model or format, making it challenging to organize, analyze, and process using traditional relational databases. Unlike structured data, which is neatly organized into rows and columns in tables, unstructured data lacks a consistent structure or schema. Here are some of the characteristics of unstructured data.

Lack of Predefined Schema:

- Unstructured data does not fit into traditional table formats or relational database schemas. It lacks a specific structure, making it challenging to categorize and query using conventional methods.

Variety of Formats:

- It can come in many different formats, including:
  - Text Files: Documents, emails, and social media posts.

  - Multimedia: Images, videos, and audio recordings.
  - Semi-Structured Data: JSON, XML, and log files that have some organizational properties but do not fit neatly into a table.
  - Web Content: HTML pages, blogs, and web crawls.
- Rich in Information:
  - Despite its lack of structure, unstructured data often contains valuable insights and information. For example, customer feedback in social media posts or multimedia content in videos can provide rich, qualitative data.
- Complexity in Processing:
  - Analyzing unstructured data typically requires advanced techniques such as natural language processing (NLP), image recognition, and machine learning. These techniques help extract meaningful patterns and insights from the raw data.

Examples of Unstructured Data:

- Textual Content: Emails, news articles, and blog posts.
- Multimedia: Photographs, video recordings, and audio files.
- Social Media: Tweets, Facebook posts, and user comments.
- Documents: PDFs, Word documents, and presentations.
- Web Data: HTML content from websites and web crawls.

MinIO is an S3-compatible object storage solution designed for on-premises deployments, making it ideal for managing large volumes of unstructured data such as images, videos, and logs. Here's how MinIO handles unstructured data:

1. Object-Based Storage: Unstructured data is stored as objects in buckets, providing a flexible way to manage diverse data types. Each object includes the data, metadata, and a unique identifier.
2. Scalability and Performance: MinIO scales horizontally by adding more storage nodes and supports high-speed operations, ensuring efficient management of petabytes of unstructured data. It uses erasure coding for data redundancy and durability.
3. S3-Compatible API: MinIO uses the Amazon S3 API, allowing existing S3-based applications to interact seamlessly with on-prem storage. This compatibility makes integration straightforward.
4. Data Security: MinIO offers end-to-end encryption and fine-grained access control to protect unstructured data. It also supports object locking in regulatory compliance.
5. Multi-Tenancy and Hybrid Cloud: MinIO supports multi-tenant environments and can integrate with cloud storage for hybrid cloud setups, providing flexible data management options.

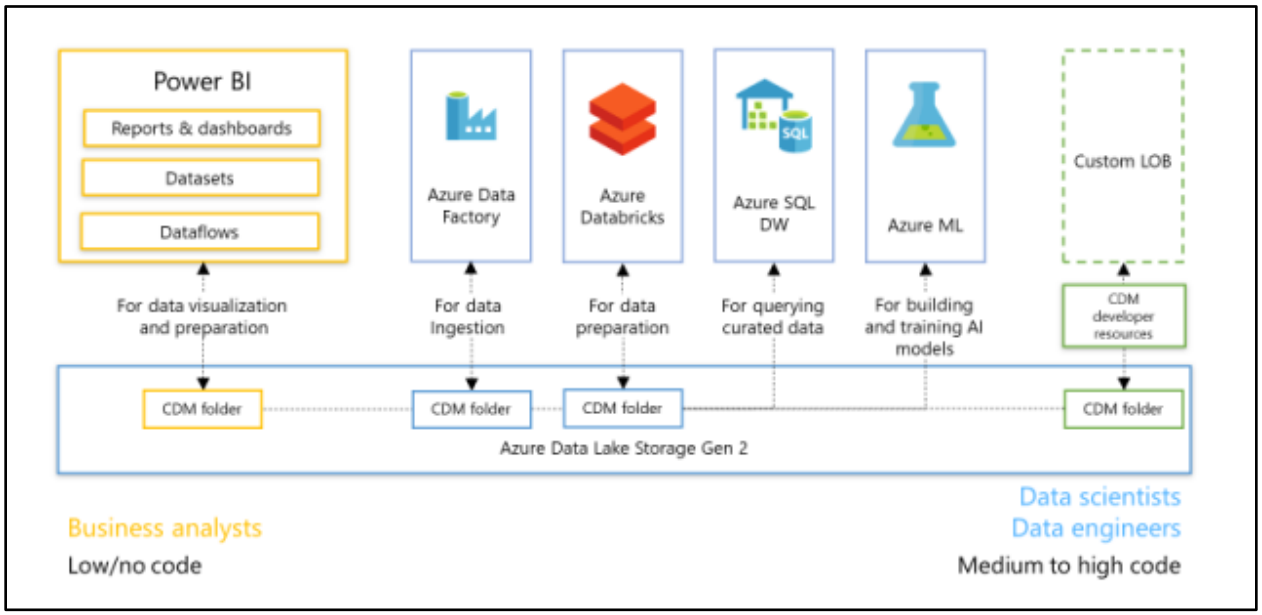
# Cloud Data Lake/Warehouse Solutions

## Azure

Due to an on-premises data lake solution's potential storage and computing requirements, some organizations opt to build their solutions through a cloud provider such as Microsoft Azure, Amazon Web Services, and Google Cloud. These platforms can abstract away many management duties of an on-premises data lake and allow endless scalability in exchange for monthly hosting fees. After consulting with stakeholders, Azure was focused on for this proof of concept. Every cloud provider offers similar capabilities, so the lessons learned from this can be applied to other providers.

Several key technologies work together to build a comprehensive data lake solution on Azure to provide robust data storage, processing, and analytics capabilities. These include Azure Data Lake Storage for scalable and secure data storage, Azure Databricks for data engineering and advanced analytics, Azure Synapse Analytics for integrated data exploration and machine learning, Azure Data Factory for orchestrating data workflows, and Microsoft Fabric, which unifies all these elements into a cohesive platform. Below is a detailed description of each technology:

### Azure Data Lake Storage




Azure Data Lake Storage (ADLS) is a scalable, secure storage solution optimized for big data analytics, built on top of Azure Blob Storage. It integrates with Hadoop ecosystems by offering HDFS-compatible access, allowing seamless use with tools like Apache Spark, Azure Synapse Analytics, and Azure Databricks. Here are some of the key features of ADLS:

- Hierarchical Namespace

  - Directory and File Structures: Unlike flat object storage, ADLS provides a hierarchical namespace, meaning you can create directories, subdirectories, and files. This structure enables more efficient data management, easier navigation, and better performance for operations such as renaming, moving, or deleting files.
  - Performance Benefits: Hierarchical namespace allows for better organization, which minimizes the overhead of listing files and improves data retrieval times for large datasets.

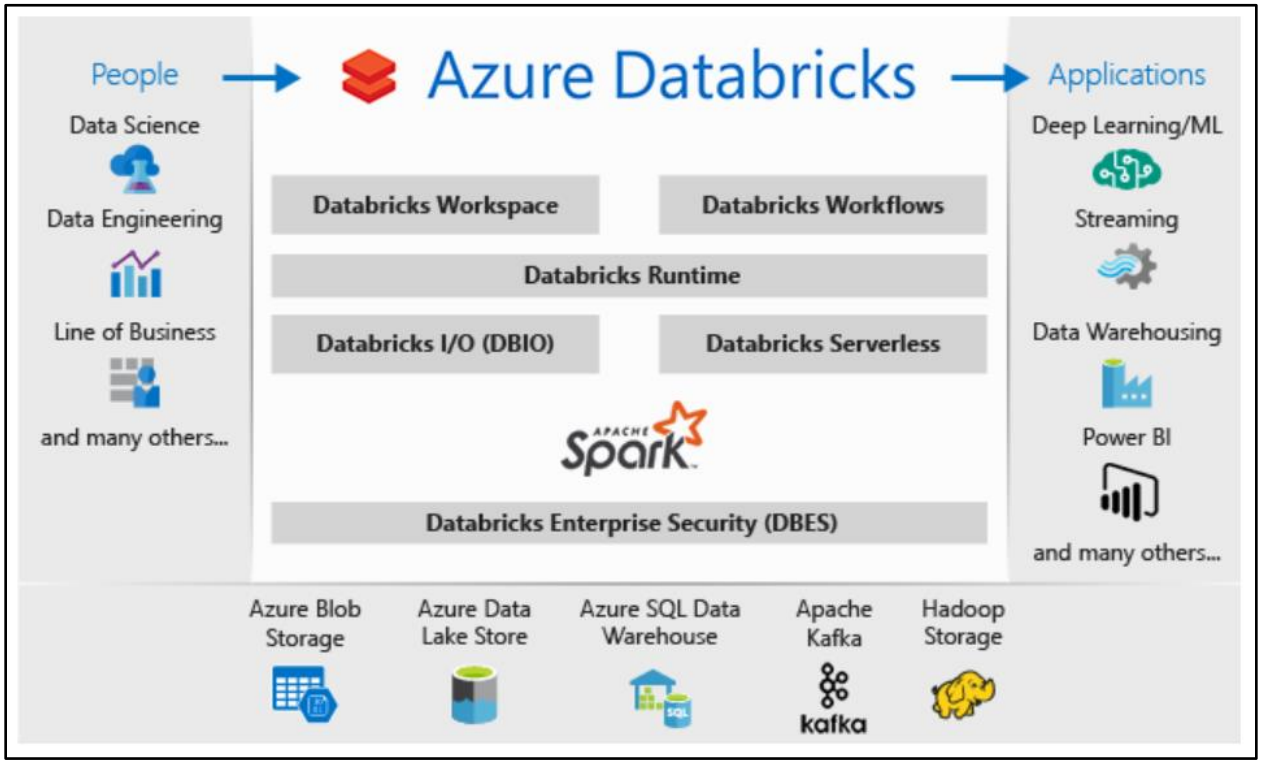
- Massive Scalability
  - Scalable to Exabytes: ADLS is built to scale infinitely, supporting vast amounts of data across structured, semi-structured, and unstructured formats (e.g., CSV, JSON, AVRO, images, videos).
  - Designed for Big Data Workloads: It's optimized to support petabyte-scale data lakes and IoT workloads, allowing real-time analytics, batch processing, and long-term storage.
  - Elasticity: The system automatically scales as data grows, providing seamless scaling without manual intervention.
- Fine-Grained Access Control
  - Role-Based Access Control (RBAC): Integrates with Azure Active Directory (AAD) to allow administrators to assign role-based permissions to specific users or groups. This enables better governance and security.
  - Access Control Lists (ACLs): ACLs enable detailed access controls at the file and directory level. You can set read, write, and execute permissions for specific users or groups, providing fine-tuned access management.
  - Multi-layered Security: Combines AAD, RBAC, and ACLs to provide enterprise-level security over sensitive datasets.
- Tiered Storage
  - Hot Tier: Optimized for frequently accessed data, providing low-latency access at a higher cost.
  - Cool Tier: Designed for infrequently accessed data, balancing performance and lower storage costs.
  - Archive Tier: Best suited for long-term, rarely accessed data, with the lowest cost but longer retrieval times. Data stored in this tier can be rehydrated when needed.
  - Automatic Lifecycle Management: You can configure policies to automatically transition data between these tiers based on usage patterns, optimizing storage costs.

- Data Encryption
    - Automatic Encryption at Rest: All data stored in ADLS is encrypted by default using AES-256 encryption, ensuring data is always secure.
    - Customer-Managed Keys (CMK): In addition to Azure-managed encryption keys, you can manage your own encryption keys via Azure Key Vault, giving you more control over encryption and compliance.
    - Encryption in Transit: ADLS supports TLS (Transport Layer Security) to encrypt data in transit, ensuring secure communication between your application and storage.
- Hadoop Integration
    - HDFS-Compatible API: ADLS offers native Hadoop Distributed File System (HDFS) APIs, allowing Hadoop-based applications like Apache Spark, Hive, and Pig to directly read and write to the data lake without code changes.
    - Seamless Data Access: This HDFS compatibility extends to any analytics tool that operates on top of Hadoop ecosystems, enabling easy access to data stored in ADLS for data processing tasks.
    - No Application Rewrites: Applications that previously used HDFS can easily transition to ADLS without requiring modifications to the application logic.
- Performance Optimization
    - High Throughput for Large Data Volumes: ADLS is optimized for high-bandwidth workloads like analytics and IoT, providing the ability to process large amounts of data in parallel with low latency.
    - Fast Data Ingestion: ADLS supports batch and streaming data ingestion, making it suitable for real-time analytics pipelines and high-velocity data processing.
    - Efficient Operations on Big Data: The hierarchical namespace reduces the overhead associated with operations such as renaming, listing, and deleting files, resulting in faster overall performance, especially for big data scenarios.
- Hybrid Cloud and On-Premises Integration
    - Azure Stack Integration: ADLS can be extended to on-premises environments via Azure Stack, allowing you to build hybrid architectures that span both on-premises and cloud environments.
    - Azure Arc: ADLS supports Azure Arc, enabling you to manage and secure data across hybrid and multi-cloud environments. This allows you to extend data lake capabilities to other public clouds or on-premises environments.

- Seamless Data Migration: Tools like Azure Data Factory and third-party migration tools can be used to move data from on-premises systems, legacy storage solutions, or other cloud platforms into ADLS.

- Analytics
  - Azure Synapse Analytics: ADLS is deeply integrated with Azure Synapse Analytics, which allows you to query and analyze massive datasets stored in ADLS using SQL, Spark, or Data Explorer pools. Synapse can read data directly from ADLS, facilitating fast and scalable analytics.
  - Azure Databricks: Databricks provides a unified analytics platform for running ETL jobs, machine learning, and real-time analytics on data stored in ADLS. This allows for large-scale data processing in a Spark-based environment.
  - Azure Data Factory: Enables orchestration of data pipelines into and out of ADLS. Data Factory can move, transform, and process data, making ADLS an essential part of ETL (Extract, Transform, Load) workflows.
- Monitoring and Management
  - Azure Monitor and Log Analytics: Provides monitoring tools to track the performance of your data lake, including operations, resource utilization, and performance metrics. Log Analytics can query and analyze logs, offering insights into how data is accessed and processed.
  - Azure Storage Explorer: A graphical tool for browsing, managing, and interacting with data stored in ADLS. It simplifies tasks such as file uploads, downloads, and access management.

## Azure DataBricks




Azure Databricks is a unified analytics platform that integrates with Azure to provide a scalable environment for big data engineering, machine learning, and real-time analytics. Built on Apache Spark, Databricks simplifies collaboration between data engineers, data scientists, and business analysts through a unified workspace. Below is an in-depth look at its key technological aspects:

- Apache Spark Foundation
    - Built on Apache Spark: Azure Databricks is designed around Apache Spark, a powerful distributed processing engine. It provides out-of-the-box support for Spark APIs in languages like Python, Scala, SQL, and R, making it ideal for high-performance data processing, ETL, and analytics tasks.
    - Unified Data Processing: Azure Databricks allows batch and real-time data processing, enabling use cases like data streaming, machine learning, and interactive analytics.
    - Cluster Management: Databricks automates the setup and configuration of Spark clusters, allowing dynamic scaling based on workload requirements, which improves efficiency and reduces cost.
- Collaborative Workspace
    - Notebooks: Databricks provides interactive notebooks where data scientists, engineers, and analysts can collaborate. These notebooks support multiple

languages (Python, Scala, SQL, R) within the same environment and allow live sharing of code, data visualizations, and outputs.

- Version Control Integration: Databricks integrates with GitHub, Azure DevOps, and other version control systems, enabling collaborative coding and seamless version management.
- Real-Time Collaboration: Multiple users can collaborate in real-time within a single notebook, making it easier to collaborate on projects and streamline communication across teams.

- Data Engineering
  - ETL and Data Pipelines: Databricks is optimized for Extract, Transform, and Load (ETL) processes, focusing on building scalable and reliable data pipelines. It offers native integration with Azure Data Lake Storage (ADLS) and Azure Blob Storage, allowing seamless data ingestion, transformation, and export.
  - Delta Lake: A key feature of Databricks is Delta Lake, an open-source storage layer that brings ACID (Atomicity, Consistency, Isolation, Durability) transactions to Spark, making data lakes more reliable. Delta Lake also provides features like schema enforcement, time travel (data versioning), and data indexing for fast reads.
  - Job Scheduling: Databricks offers built-in tools for job scheduling and monitoring, enabling automated execution of data engineering pipelines and integration with tools like Azure Data Factory for orchestrating workflows.
- Scalability and Performance
  - Dynamic Autoscaling: Databricks clusters can automatically scale up or down based on workload demands, ensuring that computational resources are used efficiently. This feature helps reduce costs by allocating resources only when needed.
  - High-Performance Optimizations: Databricks provides optimizations like caching, optimized execution plans, and vectorized query execution, which speed up batch and streaming workloads.
  - Parallelism and Distributed Processing: Leveraging Spark's distributed nature, Databricks can handle large datasets across multiple nodes, enabling parallel processing and improving performance for data-intensive tasks.
- Data Integration
  - Azure Data Services Integration: Databricks is fully integrated with Azure data services, including Azure Data Lake Storage, Azure Synapse Analytics, Azure SQL Database, and Cosmos DB. This makes it easy to ingest, process, and analyze data from various sources.

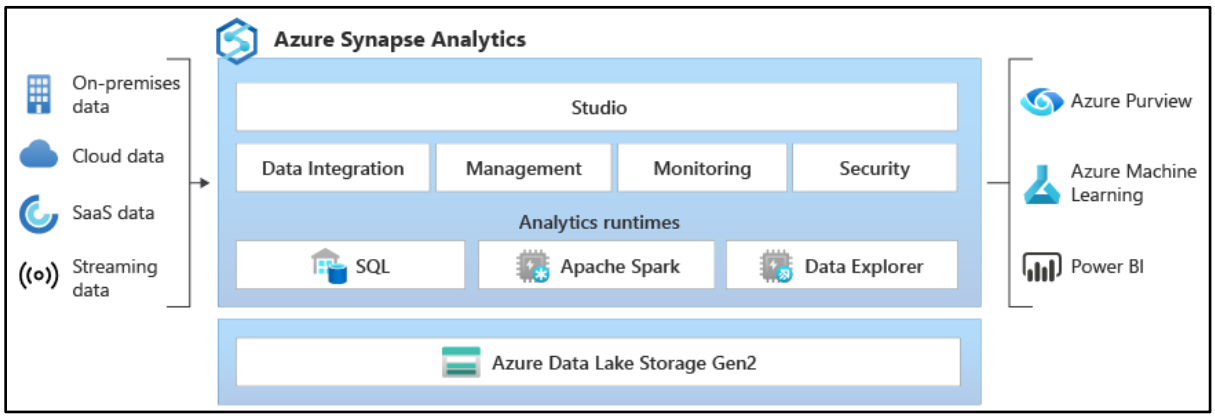
- Structured Streaming: Databricks supports real-time data processing via structured streaming, allowing for the creation of streaming data pipelines and real-time analytics use cases. It also provides connectors for event hubs and IoT devices.
- Data Ingestion: Supports multiple data ingestion methods, such as batch ingestion, streaming ingestion, and integration with data lakes and databases, making it versatile for various data sources.

- Security and Governance
  - Azure Active Directory Integration: Databricks integrates with Azure Active Directory (AAD) for single sign-on (SSO) and role-based access control (RBAC), ensuring that only authorized users can access data and compute resources.
  - Data Encryption: All data processed in Databricks is encrypted at rest and in transit, using AES-256 encryption for storage and TLS for data in transit, ensuring data security throughout its lifecycle.
  - Compliance and Certifications: Databricks adheres to various compliance standards such as GDPR, HIPAA, SOC 2, and ISO certifications, making it suitable for use cases with strict regulatory requirements.
  - Audit Logging: Databricks provides audit logging capabilities, allowing administrators to track user actions, cluster configurations, and data access to ensure compliance and governance.
- Delta Lake
  - ACID Transactions: Delta Lake enables ACID transactions, ensuring reliable and consistent data operations in data lakes. This brings transactional guarantees to big data storage, improving reliability and eliminating issues like data corruption.
  - Schema Enforcement and Evolution: Delta Lake enforces data schema, ensuring that data adheres to expected structures and allows schema evolution to accommodate changes in data over time.
  - Time Travel: Delta Lake's time travel feature allows users to query historical data versions, enabling access to previous states of the data or perform rollback operations when necessary.
  - Performance Optimizations: Delta Lake supports data compaction and indexing to improve performance when reading or querying large datasets. This reduces query latency and improves overall system efficiency.
- Monitoring and Management
  - Cluster Monitoring: Azure Databricks provides built-in tools for monitoring cluster performance, resource usage, and workload execution. You can track metrics like CPU and memory utilization in real-time and adjust as needed.

- Job Execution Monitoring: Databricks allows you to monitor the status of scheduled jobs, including details on job runs, success/failure rates, and execution times. Alerts and notifications can be configured to respond to issues in real-time.
- Logs and Audit Trails: Databricks generates detailed logs of all operations, making tracking and troubleshooting issues easy. It integrates with Azure Monitor and Log Analytics for advanced monitoring and logging.

- Pricing and Cost Management
  - Pay-As-You-Go: Databricks uses a consumption-based pricing model, where you pay for the resources (compute, storage) your clusters use. It offers flexibility in pricing based on workload and usage patterns.
  - Cluster Cost Management: Databricks includes tools for monitoring cluster usage and costs, allowing administrators to track resource usage and optimize clusters to reduce costs. Autoscaling and auto-termination features help minimize waste.
  - Reserved Pricing: Discounts are available for reserved instance pricing, allowing organizations to commit to certain workloads for extended periods in exchange for lower rates.

## Azure Synapse Analytics




Azure Synapse Analytics is a comprehensive analytics service that combines big data and data warehousing into a unified environment. It enables users to ingest, prepare, manage, and analyze data for immediate business intelligence and machine learning needs. Below is an in-depth look at the key technological aspects of Azure Synapse Analytics:

- Unified Analytics Platform
  - Integration of Data Warehousing and Big Data: Azure Synapse combines traditional data warehousing with big data analytics, allowing you to analyze relational and non-relational data from a single platform. You can query

structured and unstructured data across data lakes and data warehouses using a unified SQL experience.

  - Synapse Studio: This unified development environment enables users to build end-to-end analytics solutions in one interface. Synapse Studio integrates data exploration, preparation, orchestration, and business intelligence.

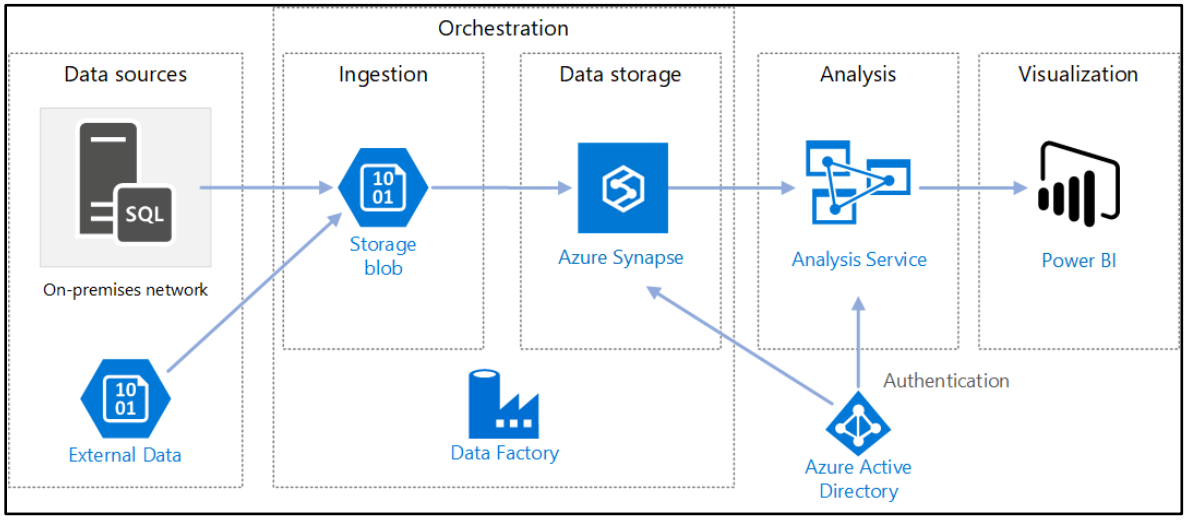
- Data Integration and Orchestration
  - Azure Data Factory Integration: Azure Synapse has built-in data integration features from Azure Data Factory. It provides over 90 pre-built connectors to seamlessly ingest data from on-premises systems, cloud platforms, and various applications. This makes it easy to move, transform, and orchestrate data pipelines.
  - Data Pipelines and Orchestration: Synapse offers a powerful orchestration engine for ETL/ELT workflows. It allows you to automate the movement and transformation of data across various services and storage types, making it easier to manage complex data pipelines.
- Data Lake Integration
  - Deep Integration with Azure Data Lake Storage (ADLS): Synapse is tightly integrated with Azure Data Lake, allowing users to query and analyze large datasets stored in Azure Data Lake Storage Gen2 directly from Synapse. This eliminates the need to move or copy data between systems, reducing complexity and costs.
  - SQL On-Demand (Serverless SQL): Synapse provides a serverless option that enables users to run SQL queries on data in data lakes without providing resources upfront. It is ideal for ad-hoc queries or exploration of large datasets without needing a dedicated data warehouse.
- Analytics Engines
  - Dedicated SQL Pools (Formerly SQL Data Warehouse): Synapse offers a dedicated SQL pool for running high-performance data warehousing workloads. This allows for the parallel processing of queries across distributed compute nodes, ensuring fast query performance even for large datasets.
  - Serverless SQL Pools: The serverless SQL pools enable on-demand querying of data in data lakes or external data sources. Users only pay for the resources used, making it cost-efficient for occasional or exploratory queries.
  - Apache Spark Integration: Synapse includes fully managed Apache Spark pools, allowing users to leverage the power of distributed data processing for big data analytics and machine learning tasks. Spark clusters can be dynamically allocated based on workload and integrate seamlessly with other Synapse features like data orchestration and monitoring.

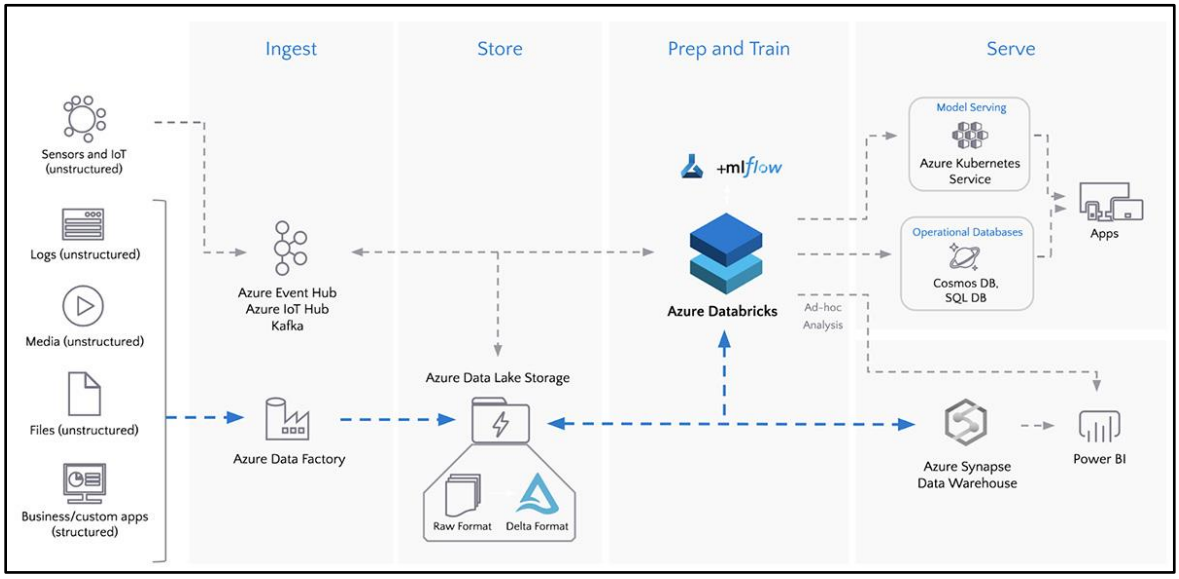
- Data Exploration and Querying
    - SQL Querying Across Data Lakes and Warehouses: With Synapse, you can use SQL to query data stored in both data lakes and data warehouses from a single interface. This makes it easy for users familiar with SQL to query large, distributed datasets.
    - PolyBase: Synapse uses PolyBase to allow querying of external data sources without importing data into Synapse. Without moving the data, you can connect and query data from different storage systems, including Azure Blob Storage, Hadoop, and even SQL Server.
- Data Security and Compliance
    - Comprehensive Security Features: Synapse provides advanced security features, including data encryption (in transit and at rest), network isolation, and data masking to protect sensitive information. It integrates with Azure Active Directory (AAD) for role-based access control (RBAC) to manage permissions.
    - Data Encryption: Azure Synapse encrypts all data at rest using AES-256 encryption. Users can manage their encryption keys via Azure Key Vault or use Azure-managed keys.
    - Virtual Networks (VNET) and Private Link: Synapse supports the use of Virtual Networks for enhanced security, enabling private communication between Synapse and other Azure resources. With Private Link, you can securely access Synapse over a private network.
    - Audit and Compliance: Azure Synapse is compliant with various industry standards like GDPR, HIPAA, and ISO. It provides logging and auditing features to track data access and changes, ensuring regulatory requirements are met.
- Synapse SQL
    - T-SQL Support: Azure Synapse supports Transact-SQL (T-SQL), which makes it familiar for SQL Server users to run complex queries, create stored procedures, and interact with databases.
    - Query Optimization and Indexing: Synapse uses advanced query optimization techniques to enhance query performance, including distributed query plans and the use of indexed views and materialized views. This makes it efficient for querying large datasets with complex join and aggregation operations.
- Real-Time Analytics and Streaming
    - Event Hubs and IoT Integration: Synapse supports integration with Azure Event Hubs, IoT Hub, and Azure Stream Analytics for real-time data ingestion and analytics. This allows real-time processing of streaming data from various sources, such as IoT devices, social media feeds, and telemetry systems.

  - Real-Time Data Processing with Apache Spark: Apache Spark in Synapse allows for real-time data processing and stream analytics. Spark Streaming is natively supported for handling high-velocity data streams and integrating real-time analytics into existing data pipelines.

- Scalability and Performance
  - Massively Parallel Processing (MPP): Azure Synapse uses MPP architecture, which splits large datasets across multiple nodes and runs queries in parallel. This ensures fast query execution and scalability for large data workloads.
  - Elasticity and Auto-Scaling: Synapse allows for auto-scaling based on workload demand. Users can scale resources up or down dynamically, depending on the volume of data and complexity of queries. This helps optimize cost and performance.
  - Caching and Data Partitioning: Synapse provides intelligent caching and data partitioning to improve query performance. It automatically partitions large datasets and uses adaptive query processing to handle complex workloads efficiently.
- Business Intelligence and Reporting
  - Power BI Integration: Azure Synapse integrates directly with Power BI, allowing users to create and share real-time dashboards and reports from Synapse data. Power BI reports can be embedded within Synapse Studio, providing an end-to-end data exploration and visualization experience.
  - Data Visualization in Synapse Studio: Synapse Studio includes built-in data visualization tools, allowing users to explore data, create charts, and generate reports without needing external BI tools. This supports interactive data analysis for business users.
- Monitoring and Management
  - Azure Monitor Integration: Synapse integrates with Azure Monitor and Log Analytics to comprehensively monitor workloads, resource utilization, and query performance. You can set up alerts and notifications for operational issues, making it easier to manage complex workloads.
  - Workload Management: Synapse provides workload management features that allow users to prioritize specific workloads and allocate resources accordingly. This ensures critical queries or data processing tasks get the required resources during peak times.

## Azure Data Factory




**Azure Data Factory (ADF)** is a cloud-based data integration service that allows you to create, orchestrate, and automate data movement and transformation workflows across on-premises and cloud environments. ADF is an essential tool for building Extract, Transform, Load (ETL) and Extract, Load, Transform (ELT) workflows for big data and analytics. Below is an in-depth look at the key technological aspects of Azure Data Factory:

- Data Integration and Orchestration
    - Data Movement: Azure Data Factory enables the movement of data from over 90+ supported data sources, including on-premises databases, cloud-based services (e.g., Azure SQL Database, Azure Data Lake, and Cosmos DB), and Software-as-a-Service (SaaS) applications like Salesforce, Google Analytics, and Dynamics 365.

- Data Transformation: ADF supports data transformation via code-free data flows (data mapping and wrangling) or by running custom code in languages such as SQL, Python, or Spark within ADF pipelines. It can scale to process large amounts of data efficiently.
- ETL/ELT Pipelines: ADF supports traditional ETL and ELT architectures. For ETL, data is extracted, transformed, and loaded into a destination data store. For ELT, data is loaded directly into the destination, and transformations happen afterward using the processing power of the destination (such as Azure Synapse Analytics or SQL databases).

- Mapping Data Flows
  - Code-Free Data Transformation: Mapping Data Flows in ADF allows users to visually design data transformation workflows without writing code. This enables data engineers to build complex data transformations, including joins, aggregations, filtering, sorting, and more.
  - Auto-Scaling: Mapping Data Flows automatically scale based on the data volume, ensuring efficient processing for small and large datasets. You don't have to manage infrastructure; ADF takes care of provisioning, scaling, and resource allocation.
  - Transformation Capabilities: With Mapping Data Flows, you can design data pipelines that involve data type conversion, derived column creation, and normalization/denormalization of data. It also supports schema drift handling for flexible data models.
- Wrangling Data Flows
  - Power Query for Data Wrangling: ADF integrates Power Query, the data transformation technology behind Power BI, to provide a code-free data wrangling experience. Wrangling Data Flows allow users to clean and reshape their data using an intuitive, Excel-like interface before loading it into a destination.
  - Data Profiling: Wrangling Data Flows provides data profiling capabilities, which allow users to understand the shape, quality, and distribution of data before performing transformations. This enables better data preparation and cleaning.
- Pipelines and Activities
  - Pipeline Design: A pipeline in ADF is a logical grouping of activities that perform a unit of work. Activities could include data movement, transformation, control flow, and external processing steps.
  - Control Flow: ADF provides control flow activities like branching, conditional execution, looping (ForEach), and error handling to orchestrate complex

workflows. These control flow activities allow users to design robust and flexible data pipelines.

  - Linked Services: Linked services define the connection information for ADF to communicate with external data sources and compute resources, such as Azure Blob Storage, on-premises SQL databases, or machine learning services. ADF supports various linked services for hybrid cloud and multi-cloud architectures.

- Data Movement and Copy Activity
  - Copy Activity: The Copy Activity in ADF moves data from a source to a destination. It can efficiently handle high-volume data transfers, supporting batch and incremental data loads.
  - Parallelism and Performance: ADF's data movement engine supports high-throughput, parallel data movement across multiple compute nodes, improving the performance of large-scale data transfers. It also allows data movement to be throttled and monitored for fine-tuned performance control.
  - On-Premises Integration: ADF supports moving data from on-premises systems to the cloud using a self-hosted integration runtime. This enables hybrid data environments and seamless data movement between on-premises systems and cloud-based services.
- Integration Runtimes
  - Azure Integration Runtime: Azure provides a fully managed compute infrastructure to execute activities like data movement, transformation, and ADF orchestration. This runtime is elastic and scales automatically.
  - Self-Hosted Integration Runtime: For on-premises data sources or network-restricted environments, ADF provides a self-hosted integration runtime that allows you to securely integrate with local resources without exposing them to the public internet. This is ideal for hybrid cloud environments.
  - SSIS Integration Runtime: ADF provides an Azure-hosted environment for running SQL Server Integration Services (SSIS) packages. This allows organizations to lift-and-shift existing SSIS ETL workloads into Azure without requiring significant modifications.
- Data Transformation with Compute Services
  - Azure Databricks Integration: ADF can orchestrate data transformation using Azure Databricks, a powerful Spark-based analytics platform. Users can trigger Databricks notebooks, jobs, or activities directly within ADF pipelines, allowing for seamless integration of big data processing.
  - Azure HDInsight and Spark Integration: ADF supports HDInsight, a managed Hadoop and Spark service, for large-scale data processing. Users can

orchestrate Apache Spark, Hadoop MapReduce, Hive, and other HDInsight jobs within ADF pipelines.

  - Azure Functions and Custom Code: ADF can trigger custom code through Azure Functions, which allows for advanced processing capabilities beyond what is natively available in ADF. This extends ADF's flexibility for handling specialized data transformation tasks.

- Scheduling and Monitoring
  - Trigger-Based Pipelines: ADF supports multiple triggering mechanisms, including time-based triggers (schedules) and event-based triggers (e.g., when new data arrives in a blob storage container). This allows for both scheduled and real-time data pipeline execution.
  - Data Pipeline Monitoring: ADF provides real-time monitoring tools that allow users to track the status of their pipelines, view historical runs, and identify performance bottlenecks. ADF also integrates with Azure Monitor for advanced monitoring, alerting, and diagnostics.
  - Error Handling and Retry Logic: ADF provides built-in error handling and retry mechanisms. Pipelines can be configured to automatically retry failed activities or branch based on error conditions, ensuring robustness and resilience in workflows.
- Data Governance and Security
  - Azure Role-Based Access Control (RBAC): ADF integrates with Azure Active Directory for role-based access control, ensuring users have appropriate permissions to perform activities on datasets and pipelines. This enhances data governance and security.
  - Managed Identity Support: ADF supports managed identities for secure authentication to Azure services, reducing the need to manage credentials. This allows secure communication between ADF and Azure resources without embedding secrets.
  - Data Encryption: Data in ADF is encrypted at rest and in transit using AES-256 encryption for data storage and TLS for secure communication. This ensures that data remains protected throughout its lifecycle.
  - Auditing and Logging: ADF provides comprehensive logging and auditing features, which track the execution of pipelines, data movement, and transformation activities. This supports compliance with industry standards and allows for troubleshooting and auditing workflows.
- Hybrid and Multi-Cloud Support
  - On-Premises Integration: ADF enables hybrid cloud scenarios by supporting on-premises data sources through the self-hosted integration runtime. This

allows for seamless data movement between on-premises environments and the cloud.

- Multi-Cloud Data Integration: ADF supports multi-cloud environments, enabling integration with services from other cloud providers such as AWS, Google Cloud, and others. This makes ADF a versatile tool for organizations operating in a multi-cloud environment.

## Azure Ecosystem Flexibility

The technologies within the Azure ecosystem—Azure Data Lake Storage, Azure Synapse Analytics, Azure Databricks, Azure Data Factory, and Power BI—can be used together as part of a seamless data pipeline or independently, depending on an organization's specific needs.

**Using the Technologies Together:**

When integrated, these tools create a unified data architecture that handles all stages of the data lifecycle:

1. **Azure Data Lake Storage (ADLS)**: Acts as the foundation for storing raw data, whether structured or unstructured. It is a scalable data lake that supports advanced analytics, machine learning, and big data workloads.
2. **Azure Data Factory (ADF)**: Orchestrates data movement into and out of ADLS. ADF can extract data from on-premises or cloud-based systems, transform it, and load it into Azure Synapse Analytics or Databricks for further processing.
3. **Azure Synapse Analytics**: Once data is in ADLS, Synapse provides the analytics layer, offering a unified workspace for querying, transforming, and analyzing large datasets. It seamlessly connects with ADLS to perform real-time analytics, data exploration, and complex SQL-based queries over structured or unstructured data.
4. **Azure Databricks**: Adds a powerful data engineering and machine learning layer. Databricks can directly read from and write to ADLS, allowing data engineers and scientists to build machine learning models, perform advanced data transformations, or preprocess data before loading it into Synapse for further querying.

Together, these tools form a data lakehouse architecture that supports the entire data lifecycle—from raw data storage to advanced analytics and reporting—within a single, connected ecosystem.

**Using the Technologies Independently:**

Each tool within the Azure ecosystem is powerful enough to be used independently, depending on specific use cases:

- **Azure Data Lake Storage (ADLS)**: Can be used standalone to store vast amounts of raw data for archiving or basic big data analytics without any advanced processing.

- **Azure Synapse Analytics**: It can function as a robust, standalone data warehouse or data analytics tool for querying, transforming, and analyzing data from various sources without needing a data lake.
- **Azure Databricks**: Can be used independently as a big data and machine learning platform. Databricks are often leveraged for data engineering tasks, machine learning workflows, or to run Spark jobs for high-performance data processing.
- **Azure Data Factory (ADF)**: Works independently as an integration and orchestration service to automate data pipelines between multiple systems, even if the final destination isn't within Azure services.

# Appendix A: API Tables

## Alerts

/v1/alerts

| Column Name | Data Type | Comments |
|---|---|---|
| alertID | String | ID |
| alertStartDateTime | DateTime | |
| intersectionId | String | |
| Type | String | |

## Cameras

/v1/intersections/{intersectionsId}/cameras

| Column Name | | Data Type | Comments |
|---|---|---|---|
| Cameras | | Camera Object List | |
| | Id | Guid | |
| | Type | String | |
| | Label | String | |
| | streamUrl | String | |

/v1/intersections/{intersectionId}/cameras/{cameraId}/snapshot

| Column Name | Data Type | Comments |
|---|---|---|
| Snapshot of current camera condition | JPEG | |

## Diagnostics

/v1/intersections/{intersectionId}/communicationDevices/diagnostics

| Column Name | | Data Type | Comments |
|---|---|---|---|
| additionalProp1 | | Diagnostic object 1 | |
| | Name | String | |
| | State | String | |
| | Description | String | |
| additionalProp2 | | Diagnostic object 1 | |
| | Name | String | |
| | State | String | |
| | Description | String | |
| | | | |
| additionalProp3 | | Diagnostic object 1 | |
| | Name | String | |
| | State | String | |

| Column Name | | Data Type | Comments |
|---|---|---|---|
| | Description | String | |

/v1/intersections/{intersectionId}/detectionDevices/diagnostics

| Column Name | | Data Type | Comments |
|---|---|---|---|
| additionalProp1 | | Diagnostic object 1 | |
| | Name | String | |
| | State | String | |
| | Description | String | |
| additionalProp2 | | Diagnostic object 1 | |
| | Name | String | |
| | State | String | |
| | Description | String | |
| additionalProp3 | | Diagnostic object 1 | |
| | Name | String | |
| | State | String | |
| | Description | String | |

## Intersections

/v1/intersections

| Column Name | | Data Type | Comments |
|---|---|---|---|
| Intersections | | List of Intersection objects | |
| | Id | Guid | |
| | Lat | Double | |
| | Long | Double | |
| | Name | String | |
| | customID | string | |

/v1/intersections/hardware

| Column Name | | | Data Type | Comments |
|---|---|---|---|---|
| Intersections | | | List of Intersection hardware objects | |
| | Id | | Guid | |
| | Devices | | List of device objects | |
| | | Id | Guid | |
| | | Serial | String | |
| | | Label | String | |

| Column Name | | | | Data Type | Comments |
|---|---|---|---|---|---|
| | Connectivity | | | Single Connectivity object | |
| | | Connected | | Bool | |
| | | Timestamp | | DateTime | |
| | | disconnectReason | | String | |
| | detectionDevices | | | List of DetectionDevices hardware info | |
| | | Id | | Guid | |
| | | Serial | | String | |
| | | Label | | String | |
| | | Connectivity | | Single connectivity object | |
| | | | Connected | Bool | |
| | | | timestamp | dateTime | |
| | | | disconnectedReason | String | |
| | Peripherals | | | List of peripheral objects | |
| | | Name | | String | |
| | | Type | | String | |

/v1/intersections/{intersectionsId}

| Column Name | Data Type | Comments |
|---|---|---|
| Id | Guid | |
| Lat | Double | |
| Long | Double | |
| Name | String | |
| customID | String | |

/v1/intersections/{intersectionsId}/hardware/detectionConfiguration

| Column Name | Data Type | Comments |
|---|---|---|
| LastUpdated | dateTime | |
| Hash | String | |
| Configuration | String | |

/v1/intersections/{intersectionId}/hiresdata

Notes: Microsoft Word - 317857-text.native.1353009594.docx (purdue.edu)

| Column Name | Data Type | Comments |
|---|---|---|
| Timestamp | dateTime | |
| eventCode | Int | |
| eventParam | Int | |

GET

/v1/intersections/{intersectionId}/notes – get notes

| Column Name | Data Type | Comments |
|---|---|---|
| Content | String | |

PUT

/v1/intersections/{intersectionId}/notes – update notes

| Column Name | Data Type | Comments |
|---|---|---|
| Content | String | |

## Organizations

/v1/organizations/{organizationId}

| Column Name | Data Type | Comments |
|---|---|---|
| Id | Guid | |
| Name | String | |

## Priority Requests

/v1/intersections/{intersectionId}/priorityCapabilities

| Column Name | Data Type | Comments |
|---|---|---|
| OutputMode | String | |
| prioritySupported | Bool | |

GET

/v1/intersections/{intersectionId}/priorityCapabilities

| Column Name | | Data Type | Comments |
|---|---|---|---|
| priorityRequests | | List of priority request objects | |
| | Id | String | |
| | timeToLive | Int | |
| | msgId | String | |
| | obuId | String | |
| | tripId | String | |

| Column Name | | | Data Type | Comments |
|---|---|---|---|---|
| | Type | | String | |
| | preemptionData | | Object | |
| | | preemtpionNumber | Int | |
| | PriorityData | | Object | |
| | | requestId | Int | |
| | | vehicleId | String | |
| | | vehicleClass | int | |
| | | vehicleClassLevel | int | |
| | | serviceStrategy | int | |

POST

/v1/intersections/{intersectionId/priorityRequests

| Column Name | | | Data Type | Comments |
|---|---|---|---|---|
| priorityRequests | | | List of priority request objects | |
| | Id | | String | |
| | timeToLive | | Int | |
| | msgId | | String | |
| | obuId | | String | |
| | tripId | | String | |
| | Type | | String | |
| | preemptionData | | Object | |
| | | preemtpionNumber | Int | |
| | PriorityData | | Object | |
| | | requestId | Int | |
| | | vehicleId | String | |
| | | vehicleClass | int | |
| | | vehicleClassLevel | int | |
| | | serviceStrategy | int | |

DELETE

/v1/intersections/{intersectionId}/priorityRequests/{priorityRequestId}

[Ends specified priority request {priorityRequestId} at specified intersection {intersectionId}]

GET

/v1/intersections/{intersectionId}/priorityRequests/{priorityRequestId}

| Column Name | | Data Type | Comments |
|---|---|---|---|
| Id | | String | |
| timeToLive | | Int | |
| msgId | | String | |
| obuId | | String | |
| tripId | | String | |
| Type | | String | |
| preemptionData | | Object | |
| | preemptionNumber | Int | |
| priorityData | | Object | |
| | requestId | Int | |
| | vehicleId | String | |
| | vehicleClass | Int | |
| | vehicleClassLevel | Int | |
| | serviceStrategy | string | |
| startedTimestamp | | DateTime | |
| lastUpdatedTimestamp | | DateTime | |
| ttlRemaining | | Int | |
| endedTimestamp | | DateTime | |
| Status | | String | |
| Error | | Object | |
| | Message | String | |
| | Timestamp | DateTime | |
| | Error | String | |
| | Status | Int | |
| | Path | String | |

PUT – update the timeToLive of a given priority request at a certain intersection.

/v1/intersections/{intersectionId}/priorityRequests/{priorityRequestId}

{body}

| Column Name | Data Type | Comments |
|---|---|---|
| timeToLive | Int | |

## Turning Movement Counts

/v1/intersections/{intersectionId}/tmc

| Column Name | Data Type | Comments |
|---|---|---|
| Timestamp | DateTime | |
| Class | String | |
| Entrance | String | |

| Column Name | Data Type | Comments |
|---|---|---|
| Exit | String | |
| Qty | Int | |

/v1/intersections/{intersectionId}/tmc/crosswalk

| Column Name | Data Type | Comments |
|---|---|---|
| Timestamp | DateTime | |
| Class | String | |
| crosswalkSide | String | |
| Direction | String | |
| qty | int | |

/v1/intersections/{intersectionId}/tmc/csv

The return value is tmc data in a CSV file format rather than an object list.

/v1/intersections/{intersectionId}/tmc/lanes

| Column Name | Data Type | Comments |
|---|---|---|
| Timestamp | DateTime | |
| Class | String | |
| Entrance | String | |
| Exit | String | |
| laneId | String | |
| Qty | int | |

## Median Travel Times

/v1/medianTravelTimes

| Column Name | | Data Type | Comments |
|---|---|---|---|
| travelTime | | List of travelTime Objects | |
| | startTime | DateTime | |
| | endTime | DateTime | |
| | Median | Float | |
| Data Confidence | | String | |

## Users

| Column Name | Data Type | Comments |
|---|---|---|
| externalUserId | String | |
| organizationName | String | |