Understanding the data playbook

Article • 02/07/2025

The data playbook provides enterprise software engineers with solutions which contain code developed to solve real-world problems. Everything in the playbook is developed with, and validated by, some of Microsoft's largest and most influential customers and partners.

Learn more about data solutions

Here is a brief introductory video about our flagship solution - Medallion architecture using a data lake also known as Modern Data Warehouse (MDW).
https://learn-video.azurefd.net/vod/player?id=05fd2288-425e-415b-8328-34db9b1cd6b8&locale=en-us&embedUrl=%2Fdata-engineering%2Fplaybook%2Funderstanding-data-playbook

The **solutions** presented in the playbook employ good engineering practices to accelerate real-world application development. Common themes include:

- Improving application design and developer productivity by sharing code and knowledge developed by experts for Microsoft customers.
- Using automation to make repetitive tasks faster, more reliable, and auditable
- Making application deployments and operations secure and observable.
- Securing applications by following security best practices at all stages of the engineering lifecycle.
- Writing portable solutions that run in multiple locations, including edge devices, on-premises data centers, the Microsoft cloud, and competitor's clouds.

Identifying integrated solutions

Data solutions span multiple Microsoft products and services and focus on creating integrated end-to-end solutions often using a range of open-source software libraries.

Proven with real customers

All code linked from playbook solutions and capabilities was created working with our customers to develop production solutions. This documentation and code is generalized to remove confidential details.

For further information

- Engineering Fundamentals
- Azure Architecture Center: Data Checklist

Feedback

Was this page helpful? Yes No

Understanding data solutions

Article • 02/07/2025

A solution is an opinionated engineering approach to solve a business problem. It provides guidance, insights, and best practices on how to develop a complete functional solution to address an end-to-end business scenario. Multiple customers successfully applied and validated all the listed solutions.

Here is a list of available solutions:

- Medallion architecture using a data lake: Implementation of Medallion architecture with a data lake, is a versatile pattern for building scalable analytical data pipelines in a cloud-first environment. It is also referred to as Modern Data Warehouse (MDW) solution in this playbook.

Implementing the Medallion architecture using a data lake

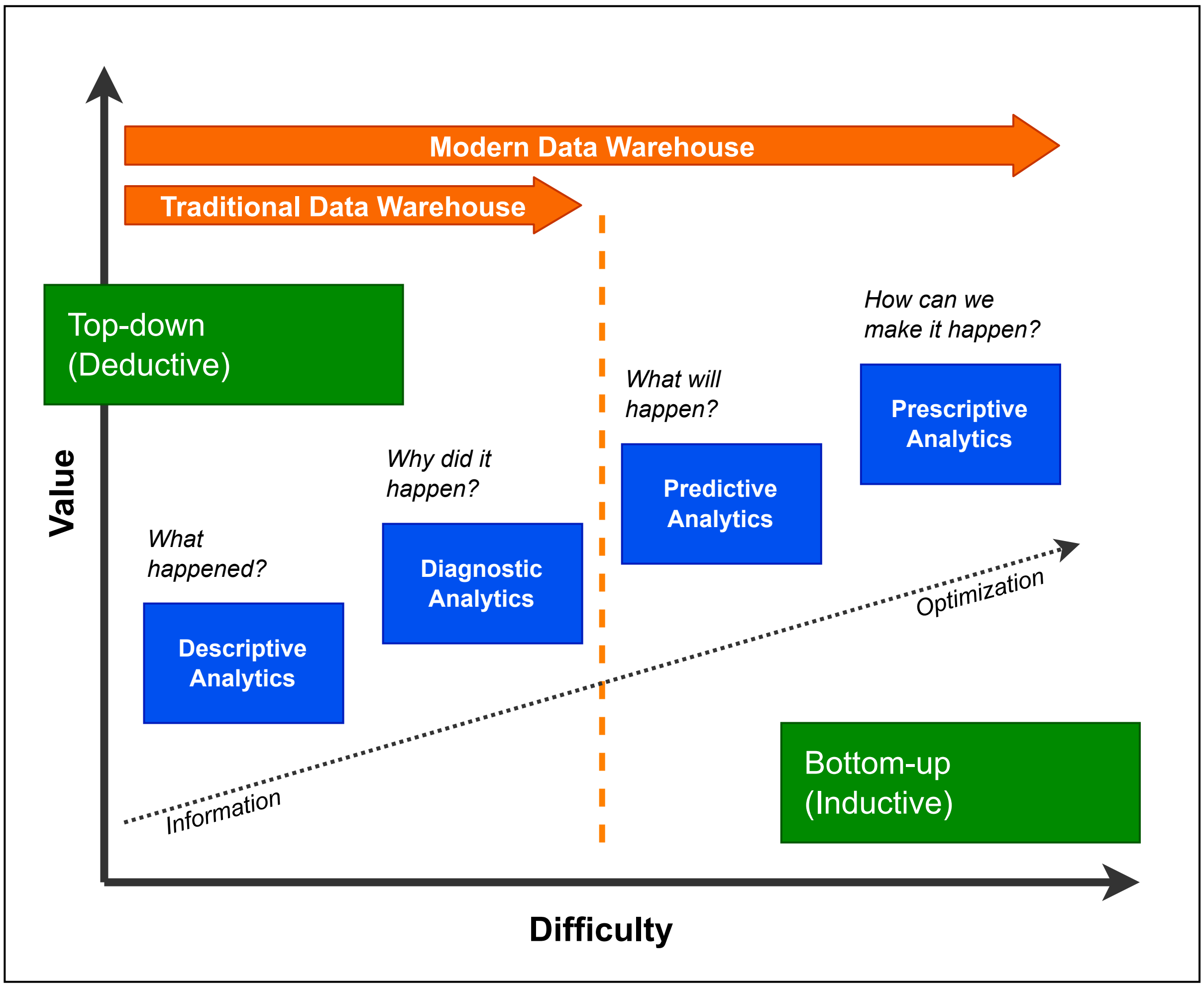
Article • 02/07/2025

This solution showcases the implementation of DataOps best practices to create a Modern Data Warehouse (MDW) using the Medallion architecture and a data lake. The MDW is a popular architectural pattern for building analytical data pipelines in a cloud-first environment. It serves as a foundation to support advanced analytical workloads, such as machine learning (ML), in addition to traditional ones, like business intelligence (BI).

Learn about the traditional data warehouse vs the Modern Data Warehouse

The modern data warehouse unlocks advanced capabilities related to analytics that would otherwise be difficult to achieve with traditional data warehousing architectures. In a traditional data warehouse, data pipelines, and the relevant dimensional model (star schema) are built based on known reporting requirements. So, analytics requirements on a traditional data warehouse can be achieved using a top-down (deductive) approach. For advanced analytical requirements in machine learning use cases, the reporting outputs are initially unknown. As a result, data scientists need to undergo an iterative exploratory analysis phase to uncover insights in raw datasets and their relevance to the outcome. This approach can be described as a bottom-up (inductive) method.

The following diagram shows the different types of analytics that can be done using both traditional and modern data warehouses:



Compared to a traditional RDBMS data warehouse, a *data lake* is the primary means of data storage in an MDW. Data lakes support storage of both structured and unstructured datasets, which is required for advanced analytics use cases. Data lake also enables schema-on-read access, which is crucial for exploratory analysis. The RDBMS data warehouse is still an important component of the MDW architecture but is now used as a serving layer to enable traditional business intelligence reporting.

The mechanism of loading data is also different. While extract-transform-load (ETL) (SQL Server Integration Services (SSIS) is an example) is preferred in traditional data warehousing, extract-load-transform (ELT) is preferred in MDW. In MDW with ELT, data is first ingested into the data lake as-is and then transformed.

Learn about the Modern Data Warehouse architecture

The following are the four stages in an MDW architecture:

1. **Ingest**: Different data sources are ingested and persisted into the data lake.
2. **Transform**: Data is validated and transformed into a predetermined schema.

3. **Model**: Data is then modeled into a form optimized for consumption (for example, Star schema).
4. **Serve**: Data is exposed for consumption. It includes enabling visualization and analysis by end users.

The following *functional components* of the architecture enable these four stages:

- **Storage**: The primary role of the storage component is to act as the main storage for both the data lake and the relevant serving layers. For details, see Understanding data lake section.
- **Compute**: Includes compute for the ingest, transform, and serve stages. Common data computing frameworks include: Apache Spark (available through Azure Synapse Spark pools or Azure Databricks), Azure Data Factory (ADF) data flows, and Azure Synapse SQL dedicated pools (particularly for the serving layer).
- **Orchestrator**: Responsible for end-to-end orchestration of the data pipeline. Azure Data Factory and Azure Synapse data pipelines are common data orchestrators.

The *nonfunctional components* that need to be considered are:

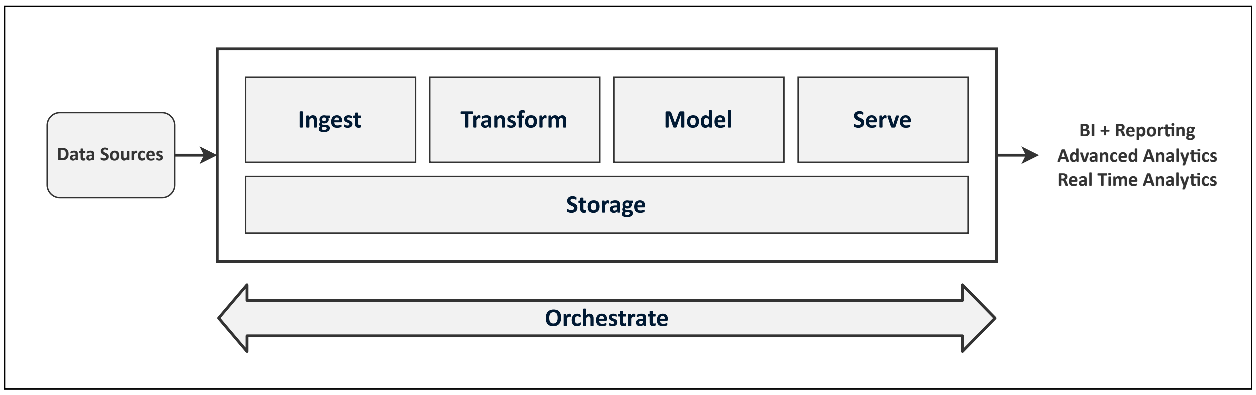
- **Security**: Includes platform, application, and data security.

- **Data governance**: Ensures datasets are governed and cataloged along with their captured lineage.

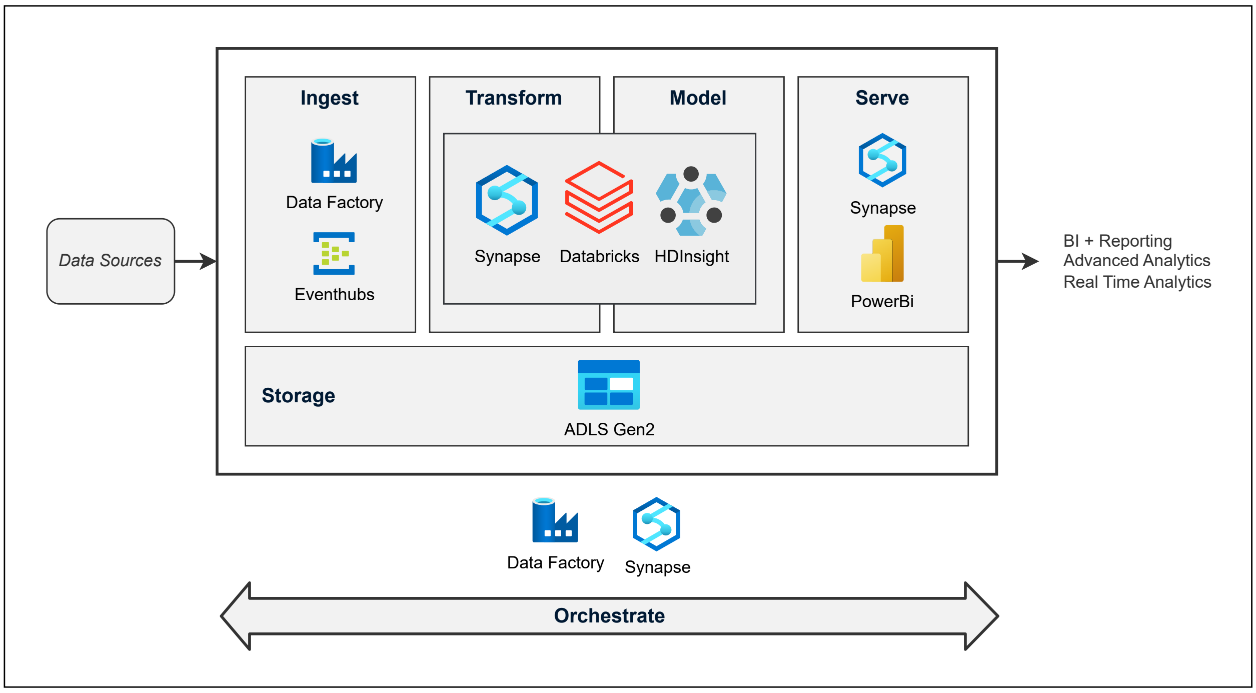
- **Operability (DevOps)**: Refers to DevOps practices to ensure efficient operations of the data system, including CI/CD, automated testing, and monitoring.

Learn about the MDW logical architecture

The diagram shows the logical MDW architecture along with its functional components:

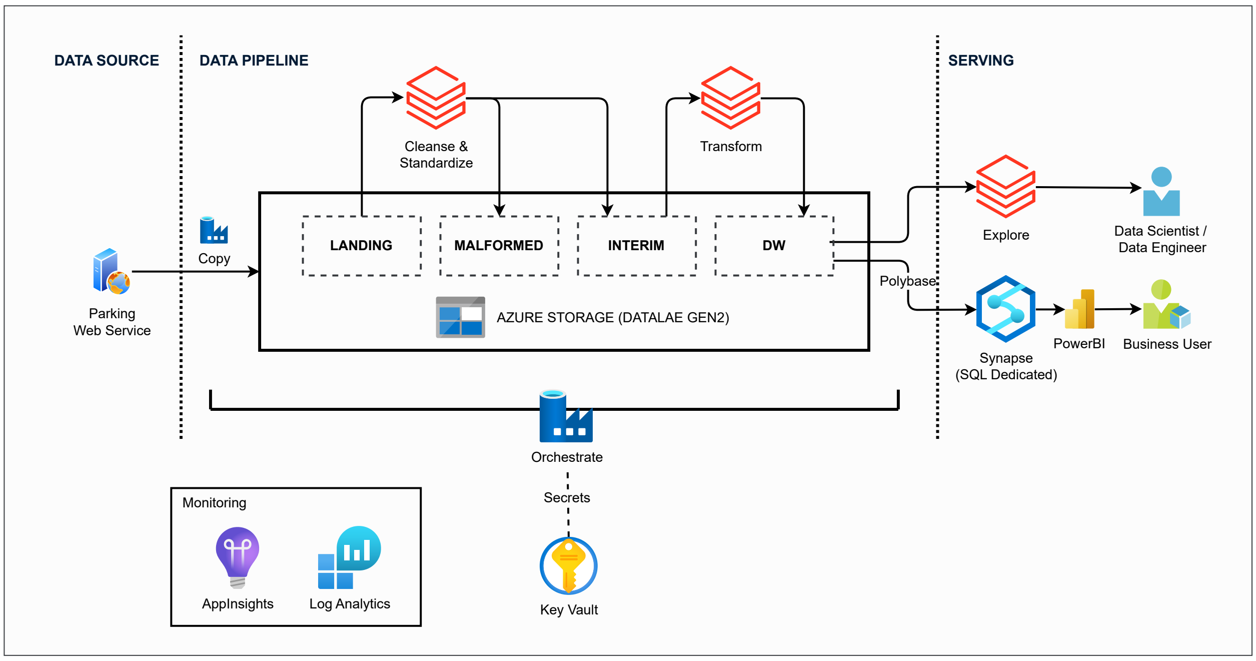


The diagram shows the logical MDW architecture with corresponding Azure services. The list of Azure services is nonexhaustive.

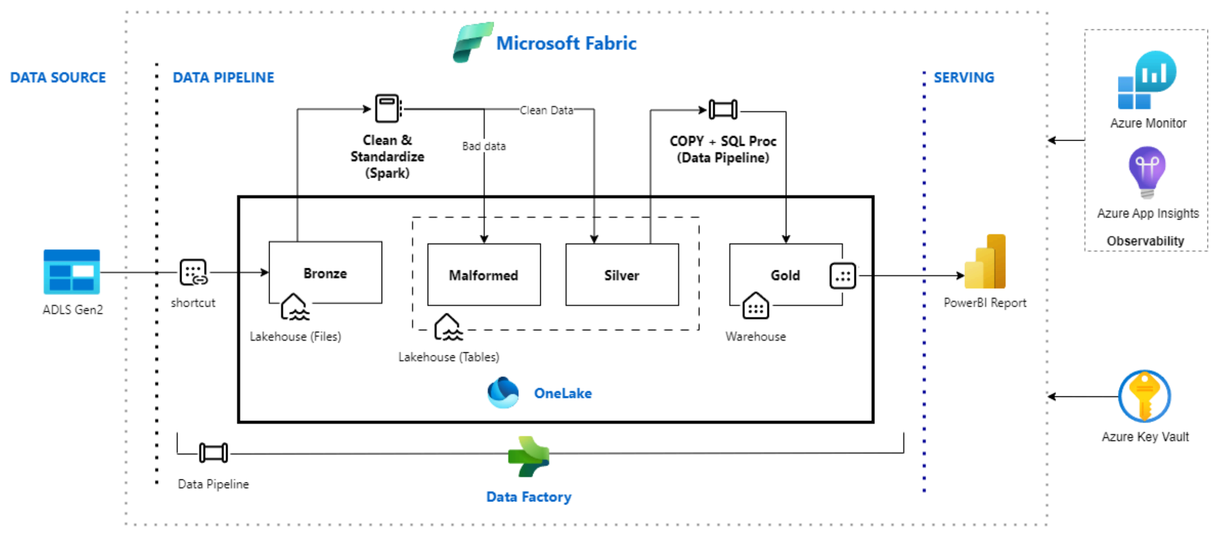


Here are a few samples of implementing an MDW architecture:

- Using Azure Databricks and Azure Data Factory:



- Using Microsoft Fabric:



Understand the limitations

The MDW architectural principles are versatile in building analytical data pipelines in a cloud-first environment. However, they don't offer comprehensive guidance in the following areas:

- Enterprise-wide data platform architecture.
- Enterprise data governance.
- Federated data architectures (Example: Data Mesh).
- Data sharing.
- On-premises data workloads.
- Transactional data workloads.
- Detailed guidance in pure streaming and event-driven data pipeline architectures.

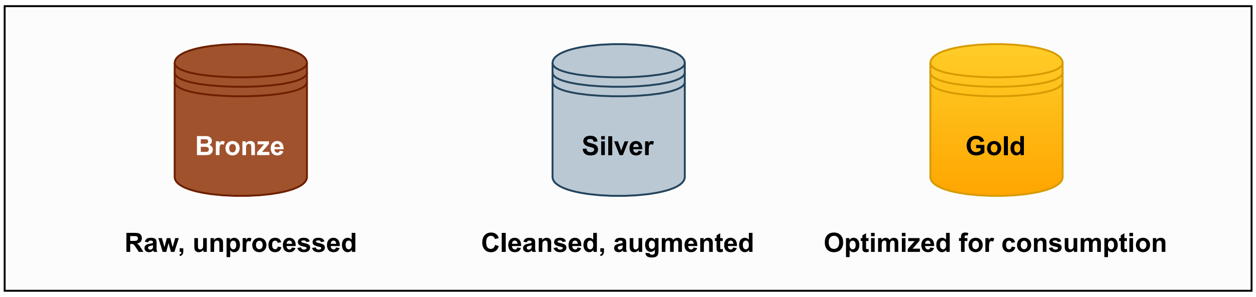
Learn Modern Data Warehouse best practices

The following section elaborates more on the specific stages and components within the MDW architecture along with key considerations and best practices.

Understanding data lake

A primary component of MDW architecture is a data lake storage that acts as the source of truth for different datasets. It is recommended to use Azure Data Lake Storage (ADLS) Gen2 for the data lake storage.

It's a best practice to logically divide the data lake into multiple zones corresponding to increasing levels of data quality. Data lake zones typically map directly to different data ingestion, transformation, and serving outputs of your data pipeline activities. At least three zones (Bronze/Silver/Gold or Raw/Enriched/Curated) are recommended:



Raw layer - Datasets are kept as similar to the source dataset as possible with little to no transformations applied. Raw datasets give the ability to replay data pipelines if there are production issues. It also means that data pipelines should be designed to be replayable and idempotent.

Enriched - Datasets have data validation applied and standardized to a common type, format, and schema. It's common to use `parquet` or `delta` as the storage format for this layer.

Curated - Datasets are optimized for consumption in the form of Data Product. It's common to have these datasets with dimensional modeling applied and eventually loaded into a data warehouse. This data lake zone forms part of the Serving layer in the MDW architecture. The common storage formats for this layer are `parquet` and `delta`.

For detailed information, see CAF: Data Lake Zones and Containers.

Implementing data ingestion

First, identify which data sources need to be ingested. For each of the identified data sources, determine:

- Location.
- Source system (FTP, storage account, SAP, SQL Server, etc.).
- Data format, if applicable (CSV, Parquet, Avro) and corresponding closest matching destination format.
- Expected volume and velocity (frequency, interval).

Second, identify the ingestion mechanism, for example, batch or streaming. The ingestion mechanism would determine appropriate technologies to be used. Azure Data Factory or Azure Synapse data pipelines are common services for ingesting batch datasets. Both provide an integration runtime (IR) for ingesting datasets on different networking environments (on-premises to cloud). While Azure Event Hubs and Azure

IoT Hub are common ingestion points for streaming when paired with a streaming service such as Azure Steam Analytics.

Generally, the ingestion pipeline requires a one-time historical load and a recurring delta load. For the latter, determine how to identify data deltas for each run. Two commonly used methods include change data capture and utilizing a dataset attribute to identify modified records. An external state store may be needed to keep track of the last loaded dataset in the form of a watermark table.

Consider using a metadata-driven approach for large-scale ingestion use cases such as loading multiple datasets. For more information, see ADF Metadata-driven copy jobs.

At times, data preprocessing is required after data ingestion and before data transformation. In cases where a large number of huge files (in gigabytes or more) are being ingested, preprocessing steps can get complicated. An industry-specific example is the processing of data in ROS format. Robot Operating System (ROS) format requires the extraction of bagged files and metadata generation for each bag file to make it available for further processing.

Azure Batch can be a great compute option for such scenarios.

Performing data transformation

Following data ingestion into the Raw zone of a data lake, the data needs to be validated and then transformed into a standard format and schema in the Enriched zone. Data validations should be performed at this point. It's best practice to maintain a malformed record store to track records that failed validations to help with debugging issues.

Common services used at this stage include Azure Synapse Spark pools, Azure Databricks, and data flows (ADF/Synapse).

Learn about data modeling

Data modeling goes hand-in-hand with data transformation. Here, data modeling refers to both the standardized data model in the enriched zone and the consumer-optimized data model in the curated zone. While related, both data models serve fundamentally different purposes. The goal of the enriched zone data model is to provide a common data model without a specific business use case in mind. The priority is completeness, data standardization, validity, and uniformity across disparate sources from the raw zone. On the other hand, datasets in the curated zone are designed to be easily consumable. The steps may include data filtering, aggregations, and use case-specific transformations depending on the specific consumer of the dataset. The curated zone therefore may contain many derivative data products produced from datasets in the enriched zone.

How to serve data

The serving layer of the architecture functions primarily to serve the data to downstream consumers. The curated zone in the data lake forms part of the serving layer. Other components such as a data warehouse, an API, or dashboard are also commonly used. Depending on the number of consumers and their requirements, the same datasets may use multiple serving mechanisms.

Common services used at serve stage include Azure Synapse dedicated SQL pool, Azure SQL, and Microsoft Power BI.

Explore MDW technical samples

The following are technical samples showcasing concrete implementations of the Modern Data Warehouse pattern:

- MDW repo: Parking Sensor (Microsoft Fabric)
- MDW repo: Parking Sensor (Azure Databricks and Azure Data Factory)

For more information

- Azure Architecture Center: DataOps for the Modern Data Warehouse
- Databricks: ETL using Azure Databricks (Tutorial)

Understanding Vector Databases

Article • 05/10/2024

In mathematics and physics, a vector is a term that refers colloquially to quantities. Usually those quantities can't be expressed by a single number (a scalar). Imagine an arrow connecting an initial point A with a terminal point B. This arrow represents a vector, and it's what you need to "carry" point A to point B. In simple words, a vector is like an arrow. You can use the arrow to learn where things are in a space, and the arrow shows both the distance and the direction.

A vector space is a set of vectors that can be played with each other. You can add or multiply them, but need to follow some certain rules.

What is a vector's dimension?

To better describe a 'vector', we need to understand what does dimension mean. A dimension is a way to describe how many directions something can be measured. In physics and mathematics, the dimension of a mathematical space (or object) is informally defined as the minimum number of coordinates needed to specify any point within it. A line has a dimension of one (1D) because only one coordinate is needed to specify a point on it – for example, the point at 5 on a number line. A surface, such as the boundary of a cylinder or sphere, has a dimension of two (2D). Because two coordinates are needed to specify a point on the surface. For example, both a latitude and longitude are required to locate a point on the surface of a sphere. In a cube, a cylinder or a sphere is three-dimensional (3D). Because three coordinates are needed to locate a point within these spaces.

What is a high-dimensional vector?

A high-dimensional vector is like a long arrow. The arrow points in many different directions at once (hard to imagine). Each direction represents a different feature or aspect of something.

Let's say we have a vector that represents a picture of a dog. In a low-dimensional vector, we might only have a few directions representing basic features like color and type. But in a high-dimensional vector, we could have hundreds or even thousands of directions. Those directions represent features. Like the shape of the dog ears, the color of the eye, the texture of its fur and so on.

So, a high-dimensional vector is like a detailed description of something. The description contains different aspects or features and was packed into one long arrow (a heavy arrow as well). These high-dimensional vectors are used in things like machine learning and data analysis to capture complex information of something and relationships between things.

What is embedding?

In natural language processing, words are represented as vectors in a high-dimensional space, where each dimension of the space corresponds to a specific aspect or feature of the word's meaning. For example, dimensions might represent things like context, semantics, or syntax.

The procedure to generate the vectors from words is called embeddings. Embeddings capture semantic relationships between words or documents by mapping them to continuous vector representations. In such a way, similar words or documents are closer together in the vector space.

Introduction to vector search

Co-location of similar words

In the vector space, the co-location of similar words refers to the phenomenon: words with similar meanings or semantic relationships are represented by vectors that are close to each other. This means that in the vector space, words are represented as high dimensional vectors. Similar words tend to have vectors that are positioned nearby.

Let's consider a simplified two-dimensional vector space where words are represented by points. In this space, for example, the word "joy" is represented by point (2, 3). The word "happy" is represented by the point (3, 4). Since they're similar in meaning, their vectors are positioned close to each other in the vector space. The word “sad” is represented by point (-5, -8), as it has an opposite meaning of joy, and their vectors are positioned far from each other.

The co-location of similar words in the vector space is crucial for various natural language processing tasks. Machine learning models use the information to capture and understand the semantic relationships between words. This information can be used in sentiment analysis, contextual information, language translation, and more.

A query will also be ‘translated’ into high dimensional vectors and find a match in the vector space. It's possible to get the results in different media format or even languages than the query content itself.

How vector search works

In vector query execution, the search engine looks for similar vectors to find the best candidates to return in search results. Depending on how you indexed the vector content, the search for relevant matches is either exhaustive, or constrained to near neighbors for faster processing. Once candidates are found, the results are scored using similarity metrics based on the strength of the match.

There are some popular algorithms used in vector search:

"KNN"(K-Nearest Neighbors): is a commonly used classification and regression algorithm. Suppose a classification task receives a training dataset and a new input instance. The task finds the K instances in the training dataset that are closest to the instance. If most of these K instances belong to a certain class, then the input instance is classified as this class.

"ANN"(Approximate Nearest Neighbors): is an algorithm for finding approximate nearest neighbors in large datasets. Unlike the KNN algorithm, which finds the exact nearest neighbors, the ANN algorithm only finds approximate nearest neighbors.

Both KNN and ANN are algorithms based on distance measurements to find the nearest neighbors. KNN finds the exact nearest neighbors, while ANN finds the approximate nearest neighbors. ANN can significantly improve search efficiency then KNN when dealing with massive data, especially in high-dimensional space searches.

"HNSW" (Hierarchical Navigable Small World): During indexing, HNSW creates extra data structures for faster search, organizing data points into a hierarchical graph structure. HNSW has several configuration parameters that can be tuned to achieve the objectives for your search application. Like throughput, latency, and recall. For example, at query time, you can specify options for exhaustive search, even if the vector field is indexed for HNSW.

During query execution, HNSW enables fast neighbor queries by navigating through the graph. This approach strikes a balance between search accuracy and computational efficiency. HNSW is recommended for most scenarios because of its efficiency when searching over larger data sets. In some vector databases, HNSW (Hierarchical Navigable Small World) does an Approximate Nearest Neighbor (ANN) search. Such as Azure AI Search.

How does index help improve the vector search efficiency

An index is a data structure that improves the speed of data retrieval operations on a database table. It works by creating a copy of a subset of the data. The copy allows the database to find the location of the desired rows in the table more quickly. Using an index in vector search has several benefits: It reduces the number of vectors that need to be compared to the query vector. The query is more efficient. Also it greatly reduces memory requirements and enhances accessibility when contrasted with processing searches via raw embeddings

A couple of popular indexing methods:

"Flat" index refers to a type of index structure that stores vectors in a flat, unstructured format. Flat index doesn't have any precomputed structures, which means each query needs to go through every single vector in the database to find its nearest neighbors. Tree-based or hierarchical indexes, are faster in searching but sacrifice accuracy. Flat indexes are typically used as a baseline to evaluate the performance of more complex index structures.

“IVF” stands for Inverted File. It's a type of indexing method frequently used in large-scale vector search and retrieval. In an IVF system, the dataset is first partitioned into several clusters. For each cluster, an inverted file (a list) is created that keeps track of the vectors that belong to this cluster. The advantage is during a search, you only need to search within the relevant clusters instead of the entire dataset. IVF index can significantly reduce the computational cost and improve search efficiency.

IVF is often combined with other techniques such as Product Quantization (PQ) to further improve search performance, especially in high-dimensional spaces.

"VPT" stands for Vantage-Point Tree. It's an index structure for performing efficient nearest neighbor search in a metric space. The Vantage-Point Tree is built by selecting a vantage point and partitioning the data into two parts: points that are nearer to the vantage point than a threshold, and points are not. This process is then recursively applied to both parts.

When doing a search, the VPT excludes large portions of the data from consideration, making the search process faster than a simple linear search. It's effective in low-dimensional spaces. However in high-dimensional spaces, the VPT can suffer from the curse of dimensionality, which makes it less efficient than other methods like KD-trees or Ball trees.

Introduction to hybrid search

Why use hybrid search

Hybrid query is a combination of full text or keyword search that contains both searchable plain text content and generated embeddings. For query purposes, hybrid search is:

- A single query request that includes both search and vectors query parameters
- Executing in parallel
- With merged results in the query response, scored using Reciprocal Rank Fusion (RRF)

Hybrid query combines the strengths of vector search and keyword search. The advantage of vector search is finding information that's conceptually similar to your search query, even if there are no keyword matches in the inverted index. The advantage of keyword or full-text search is precision, with the ability to apply semantic ranking that improves the quality of the initial results. Some scenarios - such as querying over product codes, highly specialized jargon, dates, and people's names - can do better with keyword search because it can identify exact matches.

Benchmark testing on real-world and benchmark datasets indicates that hybrid retrieval with semantic ranking offers significant benefits in search relevance.

Reciprocal Rank Fusion (RRF)

Reciprocal Rank Fusion (RRF) is an algorithm that evaluates the search scores from multiple, previously ranked results to produce a unified result set. RRF is based on the concept of reciprocal rank, which is the inverse of the rank of the first relevant document in a list of search results. The goal of the technique is to take into account the position of the items in the original rankings, and give higher importance to items that are ranked higher in multiple lists. RRF can help improve the overall quality and reliability of the final ranking, making it more useful for the task of fusing multiple ordered search results.

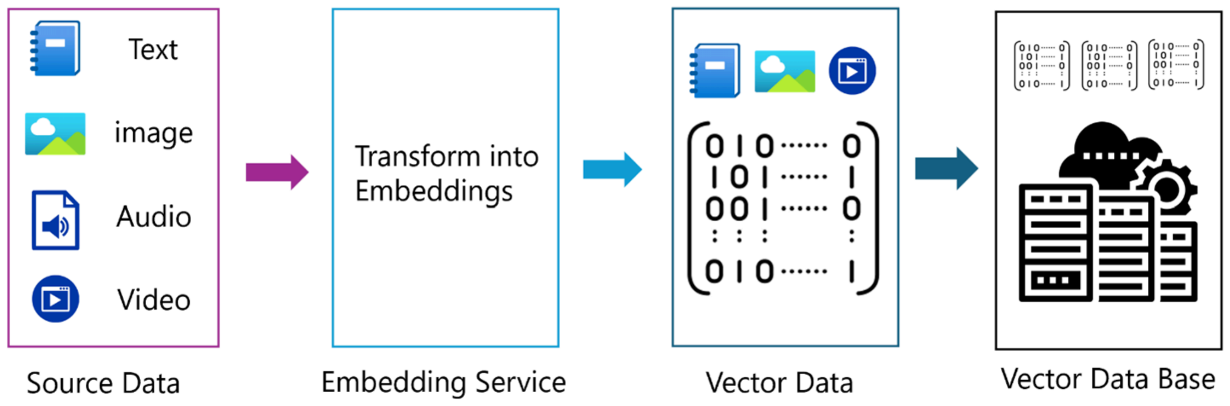
Vector database VS traditional database

A vector database is a type of database that is designed to store, manage, and index massive quantities of high-dimensional vector data efficiently. In traditional relational databases, data points are represented with rows and columns. In vector database, data points are represented by vectors with a fixed number of dimensions, clustered based on similarity.

Vector databases are important in different fields. Vector databases can efficiently store, organize, and search high-dimensional data points (also called vectors). These databases handle data where each entry is like a point in a multi-dimensional space. These vector databases are crucial for tasks like machine learning and natural language processing. Think of them as special tools that help computers find similar things. Vector databases offer an intuitive way to find similar objects, answer complex questions, and understand the hidden context of complex data.

Large language models (LLMs) have enabled many new and exciting applications to be built. But a known shortcoming of LLMs is that a trained language model doesn't have knowledge of recent events, or knowledge available only in proprietary documents. Because the model did not get to train on such knowledge. To tackle this problem, you can use retrieval augmented generation or RAG. And a key component of RAG is a vector database.

Proprietary or recent data is first stored in this vector database. Then, when there's a query that concerns that information, that query is sent to the vector database, which then retrieves the related text data. And finally, this retrieved text can be included in the prompt to the LLM to give it context with which to answer your question. Vector databases preceded this recent generative AI explosion. But vector databases have long been a broad part of semantic search applications. These applications search on the meaning of words or phrases rather than keyword search that looks for exact matches. In recommender systems, vector databases have been used to find related items to recommend to a user.



This process diagram depicts the transformation of various file formats, including text, images, audio, and video, into vector representations via an Embedding Service. Later, these vectors are stored in a Vector Database for efficient management and accessibility.

To learn more about RAG and vector indexing, refer to the documentation:

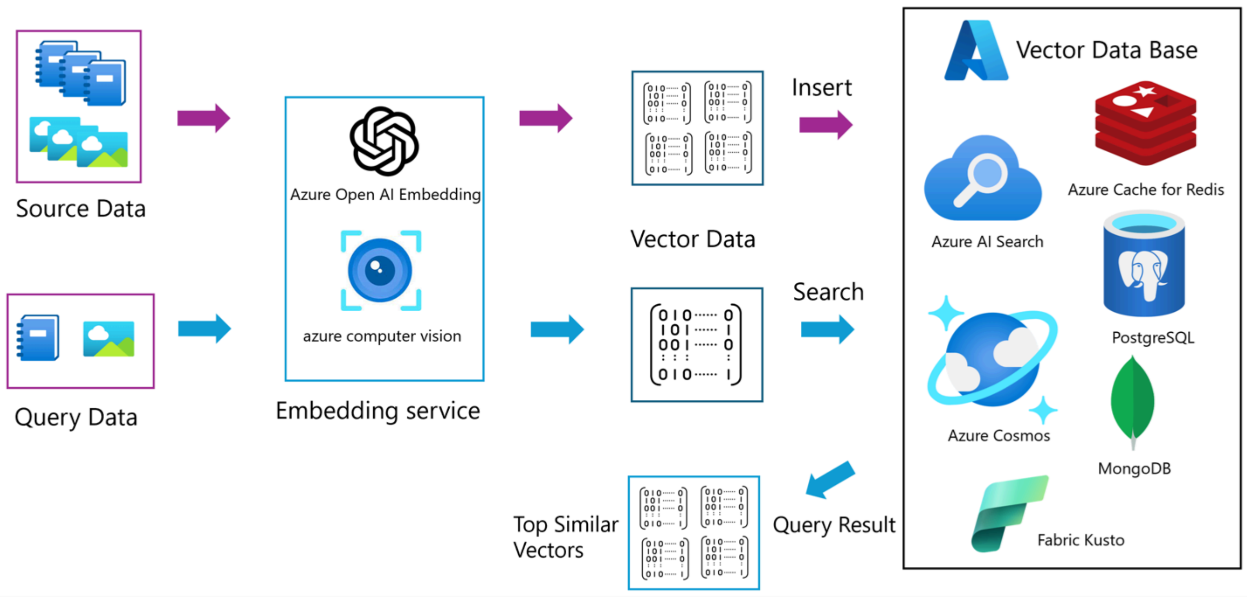
- RAG and generative AI - Azure AI Search | Microsoft Learn
- Retrieval augmented generation and indexes

Traditional databases, organize data into tables, rows, and columns. They use Structured Query Language (SQL) to manage and manipulate the stored data. They're good at handling structured data, with well-defined schemas that facilitate data organization and querying.

The choice between a vector database and a traditional database should be informed by a couple of factors: your specific use case, data types, performance requirements, and scalability needs. Vector databases are great for finding similar things and helping with machine learning. On the other hand, traditional databases are better at handling structured data and making sure transactions go smoothly.

What's covered in this playbook

The playbook contains collection of samples that demonstrates how to use different vector database tools in Azure to store embeddings and construct complex queries from text, documents and images. Each sample contains IaC scripts to spin up vector storage on Azure.



This flow chart outlines the process that is implemented by the sample code, where source data and query data are processed by Azure Open AI and Azure computer vision Embedding to produce vector data. The vector data is then inserted into a vector database, such as Azure Cache for Redis, Azure AI Search, PostgreSQL, Azure Cosmos, and MongoDB. A search operation within this database identifies the top similar vectors, leading to the final query result.

The services are commonly used in all samples of the playbook.

- Azure OpenAI Embedding Service: An embedding is a special format of data representation that can be easily used by machine learning models and algorithms. The embedding is an information dense representation of the semantic meaning of a piece of text.

- Azure Computer Vision Multi-modal embeddings: Multi-modal embedding is the process of generating a numerical representation of an image that captures its features and characteristics in a vector format. These vectors encode the content and context of an image in a way that is compatible with text search over the same vector space.

Our playbook includes a collection of samples that showcase the using of various vector database tools on Azure. These tools empower us to store vector embeddings and construct intricate queries from diverse data types, including text, documents, and images. Here are the key components of our work:

- Azure AI Search: By augmenting our data with semantic and vector search features, we enhance retrieval-augmented generation (RAG) using large language models (LLMs).
- Azure Database for PostgreSQL: In this context, we store both application data and vector embeddings within a scalable PostgreSQL offering. The native support for vector search enables efficient querying.
- Azure Cache for Redis: While Redis is commonly known for caching, we use it to manage vector embeddings efficiently. Its in-memory data structure store allows for rapid retrieval and manipulation of vectors.
- Azure CosmosDb for PostgreSQL: Our exploration extends to Cosmos DB, where we seamlessly integrate data and vectors. The native vector search capabilities enhance our ability to extract meaningful insights.
- Azure Cosmos DB for MongoDB vCore: This MongoDB-compatible service enables us to store application data and vector embeddings together. With native support for vector search, we achieve efficient retrieval and analysis.
- Fabric Real-Time Analytics (Kusto): This folder includes the notebooks to demonstrate vector search capabilities of Fabric Real-Time Analytics(Kusto) for text, documents and images.

We've developed Infrastructure as Code (IaC) scripts to create vector storage on Azure. These scripts streamline the setup process, ensuring consistency and scalability across our vector databases.

Further reading

To further learn more about building LLM apps, refer to the links:

LLMOps with Prompt flow

Chat with your data - Solution accelerator

Feedback

Was this page helpful? Yes No