

Big data architecture style

08/30/2018 • 10 minutes to read • Contributors 

In this article

[When to use this architecture](#)

[Benefits](#)

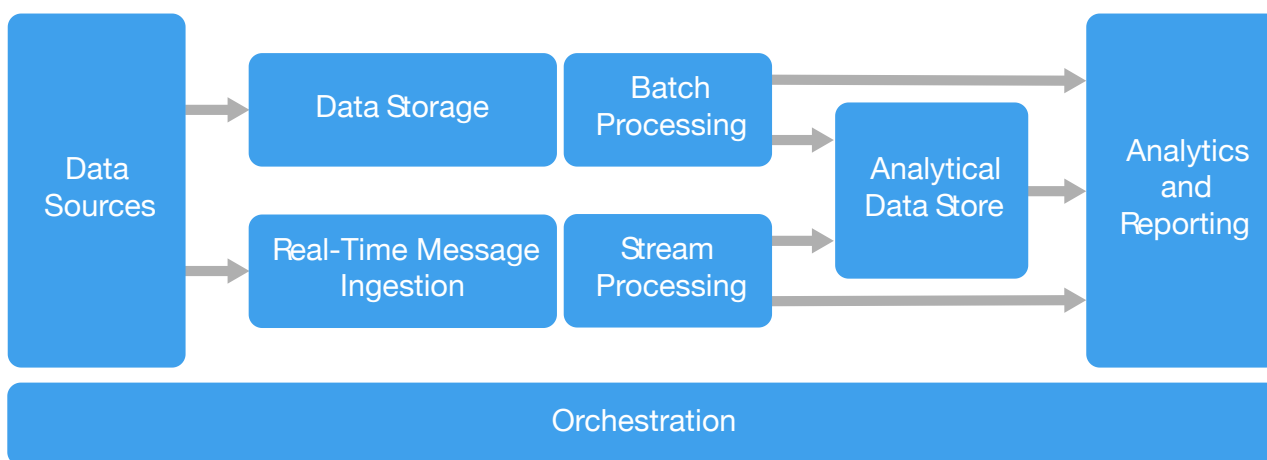
[Challenges](#)

[Best practices](#)

[IoT architecture](#)

[Next steps](#)

A big data architecture is designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems.



Big data solutions typically involve one or more of the following types of workload:

- Batch processing of big data sources at rest.
- Real-time processing of big data in motion.
- Interactive exploration of big data.
- Predictive analytics and machine learning.

Most big data architectures include some or all of the following components:

- **Data sources:** All big data solutions start with one or more data sources. Examples include:
 - Application data stores, such as relational databases.
 - Static files produced by applications, such as web server log files.
 - Real-time data sources, such as IoT devices.
- **Data storage:** Data for batch processing operations is typically stored in a distributed file store that can hold high volumes of large files in various formats. This kind of store is often called a *data lake*. Options for implementing this storage include Azure Data Lake Store or blob containers in Azure Storage.
- **Batch processing:** Because the data sets are so large, often a big data solution must process data files using long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually these jobs involve reading source files, processing them, and writing the output to new files. Options include running U-SQL jobs in Azure Data Lake Analytics, using Hive, Pig, or custom Map/Reduce jobs in an HDInsight Hadoop cluster, or using Java, Scala, or Python programs in an HDInsight Spark cluster.
- **Real-time message ingestion:** If the solution includes real-time sources, the architecture must include a way to capture and store real-time messages for stream processing. This might be a simple data store, where incoming

messages are dropped into a folder for processing. However, many solutions need a message ingestion store to act as a buffer for messages, and to support scale-out processing, reliable delivery, and other message queuing semantics. Options include Azure Event Hubs, Azure IoT Hubs, and Kafka.

- **Stream processing:** After capturing real-time messages, the solution must process them by filtering, aggregating, and otherwise preparing the data for analysis. The processed stream data is then written to an output sink. Azure Stream Analytics provides a managed stream processing service based on perpetually running SQL queries that operate on unbounded streams. You can also use open source Apache streaming technologies like Storm and Spark Streaming in an HDInsight cluster.
- **Analytical data store:** Many big data solutions prepare data for analysis and then serve the processed data in a structured format that can be queried using analytical tools. The analytical data store used to serve these queries can be a Kimball-style relational data warehouse, as seen in most traditional business intelligence (BI) solutions. Alternatively, the data could be presented through a low-latency NoSQL technology such as HBase, or an interactive Hive database that provides a metadata abstraction over data files in the distributed data store. Azure SQL Data Warehouse provides a managed service for large-scale, cloud-based data warehousing. HDInsight supports Interactive Hive, HBase, and Spark SQL, which can also be used to serve data for analysis.
- **Analysis and reporting:** The goal of most big data solutions is to provide insights into the data through analysis and reporting. To empower users to analyze the data, the architecture may include a data modeling layer, such as a multidimensional OLAP cube or tabular data model in Azure Analysis Services. It might also support self-service BI, using the modeling and visualization technologies in Microsoft Power BI or Microsoft Excel. Analysis and reporting can also take the form of interactive data exploration by data scientists or data analysts. For these scenarios, many Azure services support analytical notebooks, such as Jupyter, enabling these users to leverage their existing skills with Python or R. For large-scale data exploration, you can use Microsoft R Server, either standalone or with Spark.
- **Orchestration:** Most big data solutions consist of repeated data processing operations, encapsulated in workflows, that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report or dashboard. To automate these workflows, you can use an orchestration technology such Azure Data Factory or Apache Oozie and Sqoop.

Azure includes many services that can be used in a big data architecture. They fall roughly into two categories:

- Managed services, including Azure Data Lake Store, Azure Data Lake Analytics, Azure Data Warehouse, Azure Stream Analytics, Azure Event Hub, Azure IoT Hub, and Azure Data Factory.
- Open source technologies based on the Apache Hadoop platform, including HDFS, HBase, Hive, Pig, Spark, Storm, Oozie, Sqoop, and Kafka. These technologies are available on Azure in the Azure HDInsight service.

These options are not mutually exclusive, and many solutions combine open source technologies with Azure services.

When to use this architecture

Consider this architecture style when you need to:

- Store and process data in volumes too large for a traditional database.
- Transform unstructured data for analysis and reporting.
- Capture, process, and analyze unbounded streams of data in real time, or with low latency.
- Use Azure Machine Learning or Microsoft Cognitive Services.

Benefits

- **Technology choices.** You can mix and match Azure managed services and Apache technologies in HDInsight clusters, to capitalize on existing skills or technology investments.

- **Performance through parallelism.** Big data solutions take advantage of parallelism, enabling high-performance solutions that scale to large volumes of data.
- **Elastic scale.** All of the components in the big data architecture support scale-out provisioning, so that you can adjust your solution to small or large workloads, and pay only for the resources that you use.
- **Interoperability with existing solutions.** The components of the big data architecture are also used for IoT processing and enterprise BI solutions, enabling you to create an integrated solution across data workloads.

Challenges

- **Complexity.** Big data solutions can be extremely complex, with numerous components to handle data ingestion from multiple data sources. It can be challenging to build, test, and troubleshoot big data processes. Moreover, there may be a large number of configuration settings across multiple systems that must be used in order to optimize performance.
- **Skillset.** Many big data technologies are highly specialized, and use frameworks and languages that are not typical of more general application architectures. On the other hand, big data technologies are evolving new APIs that build on more established languages. For example, the U-SQL language in Azure Data Lake Analytics is based on a combination of Transact-SQL and C#. Similarly, SQL-based APIs are available for Hive, HBase, and Spark.
- **Technology maturity.** Many of the technologies used in big data are evolving. While core Hadoop technologies such as Hive and Pig have stabilized, emerging technologies such as Spark introduce extensive changes and enhancements with each new release. Managed services such as Azure Data Lake Analytics and Azure Data Factory are relatively young, compared with other Azure services, and will likely evolve over time.
- **Security.** Big data solutions usually rely on storing all static data in a centralized data lake. Securing access to this data can be challenging, especially when the data must be ingested and consumed by multiple applications and platforms.

Best practices

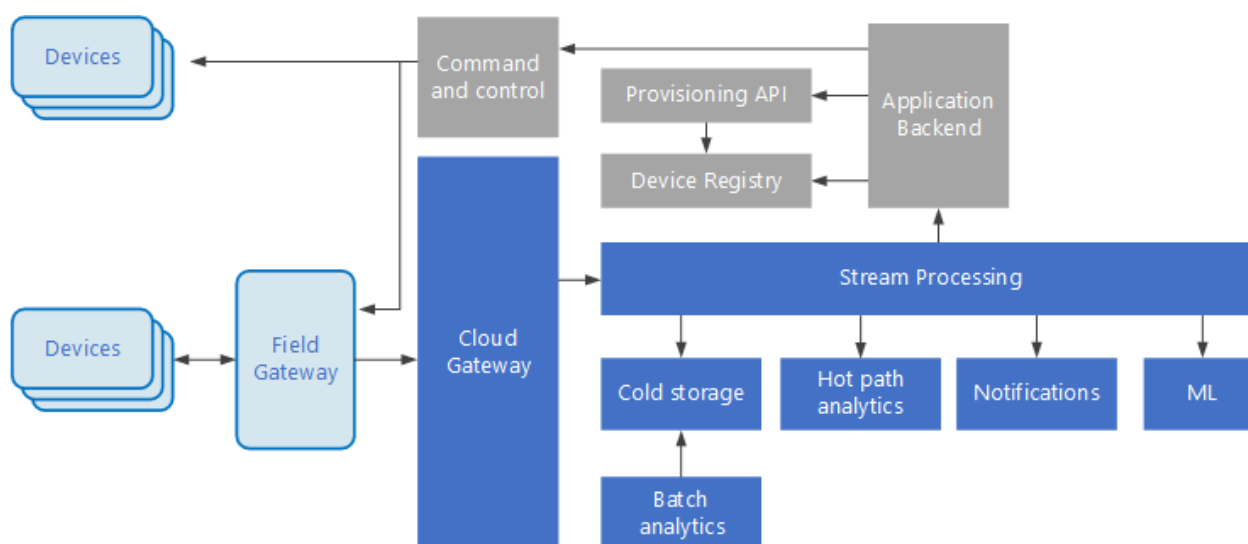
- **Leverage parallelism.** Most big data processing technologies distribute the workload across multiple processing units. This requires that static data files are created and stored in a splittable format. Distributed file systems such as HDFS can optimize read and write performance, and the actual processing is performed by multiple cluster nodes in parallel, which reduces overall job times.
- **Partition data.** Batch processing usually happens on a recurring schedule — for example, weekly or monthly. Partition data files, and data structures such as tables, based on temporal periods that match the processing schedule. That simplifies data ingestion and job scheduling, and makes it easier to troubleshoot failures. Also, partitioning tables that are used in Hive, U-SQL, or SQL queries can significantly improve query performance.
- **Apply schema-on-read semantics.** Using a data lake lets you to combine storage for files in multiple formats, whether structured, semi-structured, or unstructured. Use *schema-on-read* semantics, which project a schema onto the data when the data is processing, not when the data is stored. This builds flexibility into the solution, and prevents bottlenecks during data ingestion caused by data validation and type checking.
- **Process data in-place.** Traditional BI solutions often use an extract, transform, and load (ETL) process to move data into a data warehouse. With larger volumes data, and a greater variety of formats, big data solutions generally use variations of ETL, such as transform, extract, and load (TEL). With this approach, the data is processed within the distributed data store, transforming it to the required structure, before moving the transformed data into an analytical data store.
- **Balance utilization and time costs.** For batch processing jobs, it's important to consider two factors: The per-unit cost of the compute nodes, and the per-minute cost of using those nodes to complete the job. For example, a batch job may take eight hours with four cluster nodes. However, it might turn out that the job uses all four nodes only during the first two hours, and after that, only two nodes are required. In that case, running the entire job on two nodes would increase the total job time, but would not double it, so the total cost would be less. In

some business scenarios, a longer processing time may be preferable to the higher cost of using underutilized cluster resources.

- **Separate cluster resources.** When deploying HDInsight clusters, you will normally achieve better performance by provisioning separate cluster resources for each type of workload. For example, although Spark clusters include Hive, if you need to perform extensive processing with both Hive and Spark, you should consider deploying separate dedicated Spark and Hadoop clusters. Similarly, if you are using HBase and Storm for low latency stream processing and Hive for batch processing, consider separate clusters for Storm, HBase, and Hadoop.
- **Orchestrate data ingestion.** In some cases, existing business applications may write data files for batch processing directly into Azure storage blob containers, where they can be consumed by HDInsight or Azure Data Lake Analytics. However, you will often need to orchestrate the ingestion of data from on-premises or external data sources into the data lake. Use an orchestration workflow or pipeline, such as those supported by Azure Data Factory or Oozie, to achieve this in a predictable and centrally manageable fashion.
- **Scrub sensitive data early.** The data ingestion workflow should scrub sensitive data early in the process, to avoid storing it in the data lake.

IoT architecture

Internet of Things (IoT) is a specialized subset of big data solutions. The following diagram shows a possible logical architecture for IoT. The diagram emphasizes the event-streaming components of the architecture.



The **cloud gateway** ingests device events at the cloud boundary, using a reliable, low latency messaging system.

Devices might send events directly to the cloud gateway, or through a **field gateway**. A field gateway is a specialized device or software, usually colocated with the devices, that receives events and forwards them to the cloud gateway. The field gateway might also preprocess the raw device events, performing functions such as filtering, aggregation, or protocol transformation.

After ingestion, events go through one or more **stream processors** that can route the data (for example, to storage) or perform analytics and other processing.

The following are some common types of processing. (This list is certainly not exhaustive.)

- Writing event data to cold storage, for archiving or batch analytics.
- Hot path analytics, analyzing the event stream in (near) real time, to detect anomalies, recognize patterns over rolling time windows, or trigger alerts when a specific condition occurs in the stream.
- Handling special types of non-telemetry messages from devices, such as notifications and alarms.
- Machine learning.

The boxes that are shaded gray show components of an IoT system that are not directly related to event streaming, but are included here for completeness.

- The **device registry** is a database of the provisioned devices, including the device IDs and usually device metadata, such as location.
- The **provisioning API** is a common external interface for provisioning and registering new devices.
- Some IoT solutions allow **command and control messages** to be sent to devices.

This section has presented a very high-level view of IoT, and there are many subtleties and challenges to consider. For a more detailed reference architecture and discussion, see the [Microsoft Azure IoT Reference Architecture](#) (PDF download).

Next steps

- Learn more about [big data architectures](#).