Working with CSV and JSON files for data solutions

02/12/2018 • 4 minutes to read • Contributors

In this article

About CSV format
About JSON format
When to use CSV or JSON formats
Working with CSV and JSON data in Azure
Challenges

CSV and JSON are likely the most common formats used for ingesting, exchanging, and storing unstructured or semi-structured data.

About CSV format

CSV (comma-separated values) files are commonly used to exchange tabular data between systems in plain text. They typically contain a header row that provides column names for the data, but are otherwise considered semi-structured. This is due to the fact that CSVs cannot naturally represent hierarchical or relational data. Data relationships are typically handled with multiple CSV files, where foreign keys are stored in columns of one or more files, but the relationships between those files are not expressed by the format itself. Files in CSV format may use other delimiters besides commas, such as tabs or spaces.

Despite their limitations, CSV files are a popular choice for data exchange, because they are supported by a wide range of business, consumer, and scientific applications. For example, database and spreadsheet programs can import and export CSV files. Similarly, most batch and stream data processing engines, such as Spark and Hadoop, natively support serializing and deserializing CSV-formatted files and offer ways to apply a schema on read. This makes it easier to work with the data, by offering options to query against it and store the information in a more efficient data format for faster processing.

About JSON format

JSON (JavaScript Object Notation) data is represented as key-value pairs in a semi-structured format. JSON is often compared to XML, as both are capable of storing data in hierarchical format, with child data represented inline with its parent. Both are self-describing and human readable, but JSON documents tend to be much smaller, leading to their popular use in online data exchange, especially with the advent of REST-based web services.

JSON-formatted files have several benefits over CSV:

- JSON maintains hierarchical structures, making it easier to hold related data in a single document and represent complex relationships.
- Most programming languages provide native support for deserializing JSON into objects, or provide lightweight JSON serialization libraries.
- JSON supports lists of objects, helping to avoid messy translations of lists into a relational data model.
- JSON is a commonly used file format for NoSQL databases, such as MongoDB, Couchbase, and Azure Cosmos DB.

Since a lot of data coming across the wire is already in JSON format, most web-based programming languages support working with JSON natively, or through the use of external libraries to serialize and deserialize JSON data. This universal support for JSON has led to its use in logical formats through data structure representation, exchange formats for hot data, and data storage for cold data.

Many batch and stream data processing engines natively support JSON serialization and deserialization. Though the data contained within JSON documents may ultimately be stored in a more performance-optimized formats, such as Parquet or Avro, it serves as the raw data for source of truth, which is critical for reprocessing the data as needed.

When to use CSV or JSON formats

CSVs are more commonly used for exporting and importing data, or processing it for analytics and machine learning. JSON-formatted files have the same benefits, but are more common in hot data exchange solutions. JSON documents are often sent by web and mobile devices performing online transactions, by IoT (internet of things) devices for one-way or bidirectional communication, or by client applications communicating with SaaS and PaaS services or serverless architectures.

CSV and JSON file formats both make it easy to exchange data between dissimilar systems or devices. Their semi-structured formats allow flexibility in transferring almost any type of data, and universal support for these formats make them simple to work with. Both can be used as the raw source of truth in cases where the processed data is stored in binary formats for more efficient querying.

Working with CSV and JSON data in Azure

Azure provides several solutions for working with CSV and JSON files, depending on your needs. The primary landing place for these files is either Azure Storage or Azure Data Lake Store. Most Azure services that work with these and other text-based files integrate with either object storage service. In some situations, however, you may opt to directly import the data into Azure SQL or some other data store. SQL Server has native support for storing and working with JSON documents, which makes it easy to import and process those types of files. You can use a utility like SQL Bulk Import to easily import CSV files.

You can also query JSON files directly from Azure Blob Storage without importing them into Azure SQL. For a complete example of this approach, see Work with JSON files with Azure SQL. Currently this option isn't available for CSV files.

Depending on the scenario, you may perform batch processing or real-time processing of the data.

Challenges

There are some challenges to consider when working with these formats:

- Without any restraints on the data model, CSV and JSON files are prone to data corruption ("garbage in, garbage out"). For instance, there's no notion of a date/time object in either file, so the file format does not prevent you from inserting "ABC123" in a date field, for example.
- Using CSV and JSON files as your cold storage solution does not scale well when working with big data. In most cases, they cannot be split into partitions for parallel processing, and cannot be compressed as well as binary formats. This often leads to processing and storing this data into read-optimized formats such as Parquet and ORC (optimized row columnar), which also provide indexes and inline statistics about the data contained.
- You may need to apply a schema on the semi-structured data to make it easier to query and analyze. Typically, this requires storing the data in another form that complies with your environment's data storage needs, such as within a database.