Couchbase

JSON vs. BSON

Breaking down the difference between JSON and BSON formats

Learn More About JSON Compare databases

JSON vs. BSON performance overview

This page will cover the following to help you better understand the key differences between JSON and BSON:

JSON (which stands for JavaScript Object Notation) is a lightweight text-based data interchange format that's easy for humans to read and write, and easy for machines to parse and generate. JSON is a popular choice for exchanging data between different systems, especially in web development.

BSON (which stands for Binary JSON) is a binary-encoded serialization format that supports additional data types and is natively supported only by MongoDB™.

Hypothetically, BSON can be more efficient than JSON in terms of network transmission, but Couchbase, which uses JSON as its primary data format, actually **outperforms MongoDB™ for many practical workloads**. Also, BSON's complexity and limited compatibility with other systems can limit its usefulness in some contexts.

What is JSON?

JSON format was originally derived from a subset of the JavaScript programming language syntax, so it shares many of the same syntax rules and data types as JavaScript. As a result, JSON can be easily parsed and generated using JavaScript and is often used in web development for exchanging data between client-side JavaScript and server-side programs written in various programming languages.

Because JSON is lightweight, it is efficient to transmit over a network, which is important for web-based applications that need to transfer data quickly. JSON's user-friendly format allows it to be easily understood and edited by developers and non-developers alike. And because it's easy for machines to parse and generate, JSON can be easily integrated into a wide range of programming languages and platforms, making it a versatile and widely adopted format for exchanging data.

JSON data is represented as key-value pairs, similar to a dictionary or hash table in other programming languages, which makes it easy for developers to understand and use in their programs. By representing data as key-value pairs, **JSON** provides a flexible and intuitive way to organize and access data. The key-value pair structure also makes it easy to map data to objects in various programming languages, which is useful for integrating data between different systems.

Why use JSON?

Because JSON is lightweight, it is efficient to transmit over a network, which is important for web-based applications that need to transfer data quickly. JSON's user-friendly format allows it to be easily understood and edited by developers and non-developers alike. And because it's easy for machines to parse and generate, JSON can be easily integrated into a wide range of programming languages and platforms.

JSON also supports a wide range of data types, including strings, numbers, arrays, and objects, which makes it a flexible and versatile format for representing data. This flexibility makes JSON an excellent tool for exchanging data between different systems and programming languages. This allows developers to use it in a wide range of applications, from simple data storage and retrieval to complex data processing and analysis.

What is BSON?

BSON is a binary-encoded serialization format that is more compact than raw JSON, and more efficient for storing data or transmitting over a network.

BSON supports additional data types that are outside of standard JSON, such as binary data and date types. By supporting data types beyond the strings, numbers, and arrays supported by JSON, BSON can more accurately represent complex data structures and types. This increases the complexity of the format, which can make it more difficult to work with in certain contexts. There is also a greater risk of compatibility issues when exchanging data between systems that do not fully support BSON's additional types.

MongoDB is currently the only database system that natively uses BSON as its storage format. Because BSON was developed specifically for MongoDB, it's optimized for their unique architecture and data model, and MongoDB is able to provide support for complex data types. But because BSON is not widely supported outside of MongoDB, its usefulness is limited in some contexts, mainly when interoperability with other systems is a top priority.

Main differences between JSON and BSON

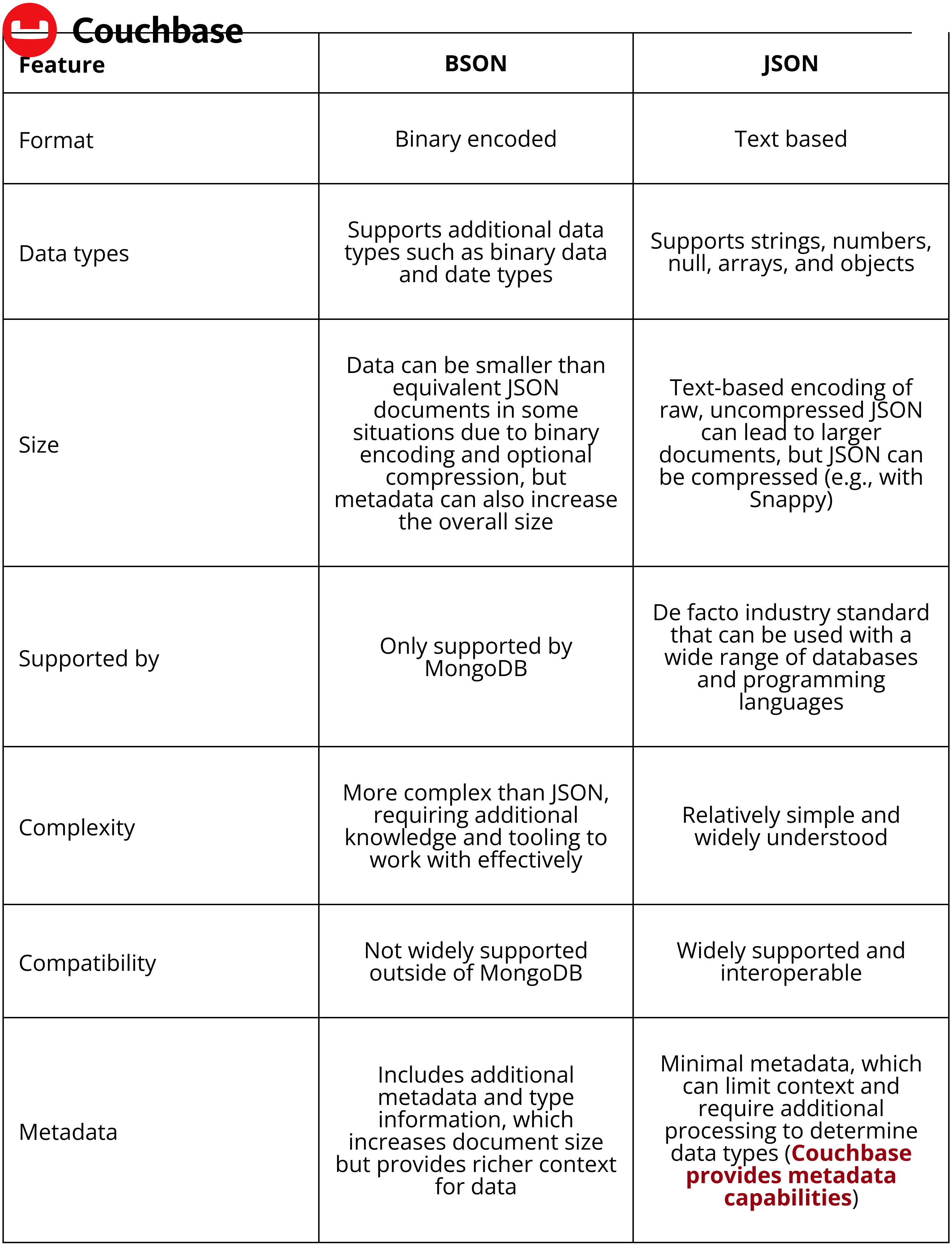
Binary vs. text: BSON is a binary-encoded format, whereas JSON is a text-based format. This means that BSON is compact for transmitting over a network, while JSON is human-readable and easier to work with in various contexts.

Extended data support: JSON is limited to JavaScript data types, including string, number, boolean, null, object, and array. Those data types can be used in combination to represent complex data types. BSON supports additional data types (such as binary data and date types) that are not supported by JSON.

Supported by: BSON is natively supported only by MongoDB. JSON, on the other hand, is widely supported and can be used with **distributed database systems**, programming languages, and platforms.

Footprint: In some situations, BSON documents can be larger than equivalent JSON documents because they include additional metadata and type information that is not present in JSON. This can impact transmission times and storage requirements, especially for large datasets. Both BSON and JSON can benefit from compression.

Complexity and compatibility: BSON is more complex than JSON, making it difficult to work with in certain contexts. Developers may need to learn new data types and encoding/decoding methods to work with BSON effectively. Compatibility issues may also arise when exchanging data between systems that do not fully support BSON's additional types.



Feature	BSON	JSON
Format	Binary encoded	Text based
Data types	Supports additional data types such as binary data and date types	Supports strings, numbers, null, arrays, and objects
Size	Data can be smaller than equivalent JSON documents in some situations due to binary encoding and optional compression, but metadata can also increase the overall size	Text-based encoding of raw, uncompressed JSON can lead to larger documents, but JSON can be compressed (e.g., with Snappy)
Supported by	Only supported by MongoDB	De facto industry standard that can be used with a wide range of databases and programming languages
Complexity	More complex than JSON, requiring additional knowledge and tooling to work with effectively	Relatively simple and widely understood
Compatibility	Not widely supported outside of MongoDB	Widely supported and interoperable
Metadata	Includes additional metadata and type information, which increases document size but provides richer context for data	Minimal metadata, which can limit context and require additional processing to determine data types (**Couchbase provides metadata capabilities**)

Feature	BSON	JSON
Use cases	Suited for working with MongoDB	Suitable for a wide range of data interchange scenarios, from web APIs to data storage and transmission

Advantages of JSON

Simplicity: JSON is a simple, lightweight, and easy-to-read data format that is easy for both humans and machines to understand, making it a popular choice for data exchange on the web.

Platform and language agnostic: JSON can be used with virtually any programming language, making it a versatile choice for developers working across different platforms and systems.

Data serialization: JSON is an efficient method of serializing and transmitting complex data structures over the network, making it a popular choice for web APIs and other distributed systems.

Supports complex data structures: JSON supports complex data structures such as arrays, objects, and nested structures, making it a powerful tool for data modeling and representation.

Advantages of BSON

Compactness: BSON is a binary format that can be more compact than JSON in some situations.

Support for additional data types: BSON supports additional data types, such as binary data and timestamp, that are not part of standard JSON.

Couchbase

Does Couchbase use JSON or BSON format?

Couchbase uses JSON as its primary data format and does not natively support BSON.

Why use JSON? JSON is a human-readable and lightweight data format widely used in web development, making it easy to work with across different platforms and systems. Because JSON offers flexibility in data modeling and supports complex data structures, it's an ideal choice for applications that require efficient data serialization and transmission over the network. While BSON offers advantages such as compactness and additional data types, Couchbase has opted to stick with the simplicity and versatility of JSON for its data storage and retrieval needs.

Couchbase combines a cache with a JSON document database and is the original multi-model database. Using the foundations of standard JSON, Couchbase supports the following models and access methods:

- **Key-value –** the use of key-value pairs enables **fast lookup** of JSON documents and can be serialized/deserialized efficiently by every developer language/platform.
- **SQL++ (SQL for JSON) –** SQL is declarative, concise, and readable, which is why it's the world's most popular data query language and is natively supported by the most popular databases. **SQL++** is simply an extension of SQL that supports JSON.
- **Full-text search –** JSON is not only readable, it's also searchable. Using the open source Bleve engine, Couchbase supports **full-text search** indexes of JSON data, including fuzziness, regular expressions, wildcards, faceting, and everything else you'd expect from a text search engine.
- **Geospatial –** JSON data can include **GeoJSON or Geopoint** data. Using the full-text search engine, JSON can be searched by location using radius, bounding box, or polygon.
- **Mobile sync –** JSON is conducive to **mobile development** because it enables efficient and ubiquitous JSON serialization/deserialization.

Because all of these methods work on the same pool of JSON data, Couchbase can do the job of two, three, or more point solutions without adding more data