# Using consistency levels to maximize availability and performance in DocumentDB

Developers are often faced with the challenge of choosing between the two extremes of strong and eventual consistency. The reality is that there are multiple consistency click-stops between these two extremes. In most real world scenarios, applications benefit from making fine grained trade-offs between consistency, availability, and latency. DocumentDB offers four well-defined consistency levels with associated performance levels. This allows application developers to make predictable consistency, availability, and latency trade-offs.

All system resources, including database accounts, databases, collections, users, and permissions are always strongly consistent for reads and queries. The consistency levels apply only to the user defined resources. For queries and read operations on user defined resources, including documents, attachments, stored procedures, triggers, and UDFs, DocumentDB offers four distinct consistency levels:

* Strong consistency
* Bounded staleness consistency
* Session consistency
* Eventual consistency

These granular, well-defined consistency levels allow you to make sound trade-offs between consistency, availability, and performance. These consistency levels are backed by predictable performance levels ensuring consistent results for your application.

## Consistency levels for databases

You can configure a default consistency level on your database account that applies to all the collections (across all of the databases) under your database account. By default, all reads and queries issued against the user defined resources will use the default consistency level specified on the database account. However, you can lower the consistency level of a specific read/query request by specifying [x-ms-consistency-level] request header. There are four types of consistency levels supported by the DocumentDB replication protocol - these are briefly described below.

[AZURE.NOTE] In a future release, we intend to support overriding the default consistency level on a per collection basis.

**Strong**: Strong consistency guarantees that a write is only visible after it is committed durably by the majority quorum of replicas. A write is either synchronously committed durably by both the primary and the quorum of secondaries or it is aborted. A read is always acknowledged by the majority read quorum - a client can never see an uncommitted or partial write and is always guaranteed to read the latest acknowledged write.

Strong consistency provides absolute guarantees on data consistency, but offers the lowest level of read and write performance.

**Bounded staleness**: Bounded staleness consistency guarantees the total order of propagation of writes with the possibility that reads lag behind writes by at most K prefixes. The read is always acknowledged by a majority quorum of replicas. The response of a read request specifies its relative freshness (in terms of K). With bounded staleness you can set configurable threshold of staleness (as prefixes or time) for reads to tradeoff latency and consistency in steady state.

Bounded staleness provides more predictable behavior for read consistency while offering the lowest latency writes. As reads are acknowledged by a majority quorum, read latency is not the lowest offered by the system. Bounded Staleness is an option for scenarios where you want strong consistency but where strong consistency is not practical. If you configure the “staleness interval” for Bounded Staleness consistency to be arbitrary large, it will still preserve the total global order of writes. This provides a stronger guarantee than Session or Eventual.

[AZURE.NOTE] Bounded staleness guarantees monotonic reads only on explicit read requests. The echoed server response for write requests does not offer bounded staleness guarantees.

**Session**: Unlike the global consistency models offered by strong and bounded staleness consistency levels, “session” consistency is tailored for a specific client session. Session consistency is usually sufficient since it provides guaranteed monotonic reads, and writes and ability to read your own writes. A read request for session consistency is issued against a replica that can serve the client requested version (part of the session cookie).

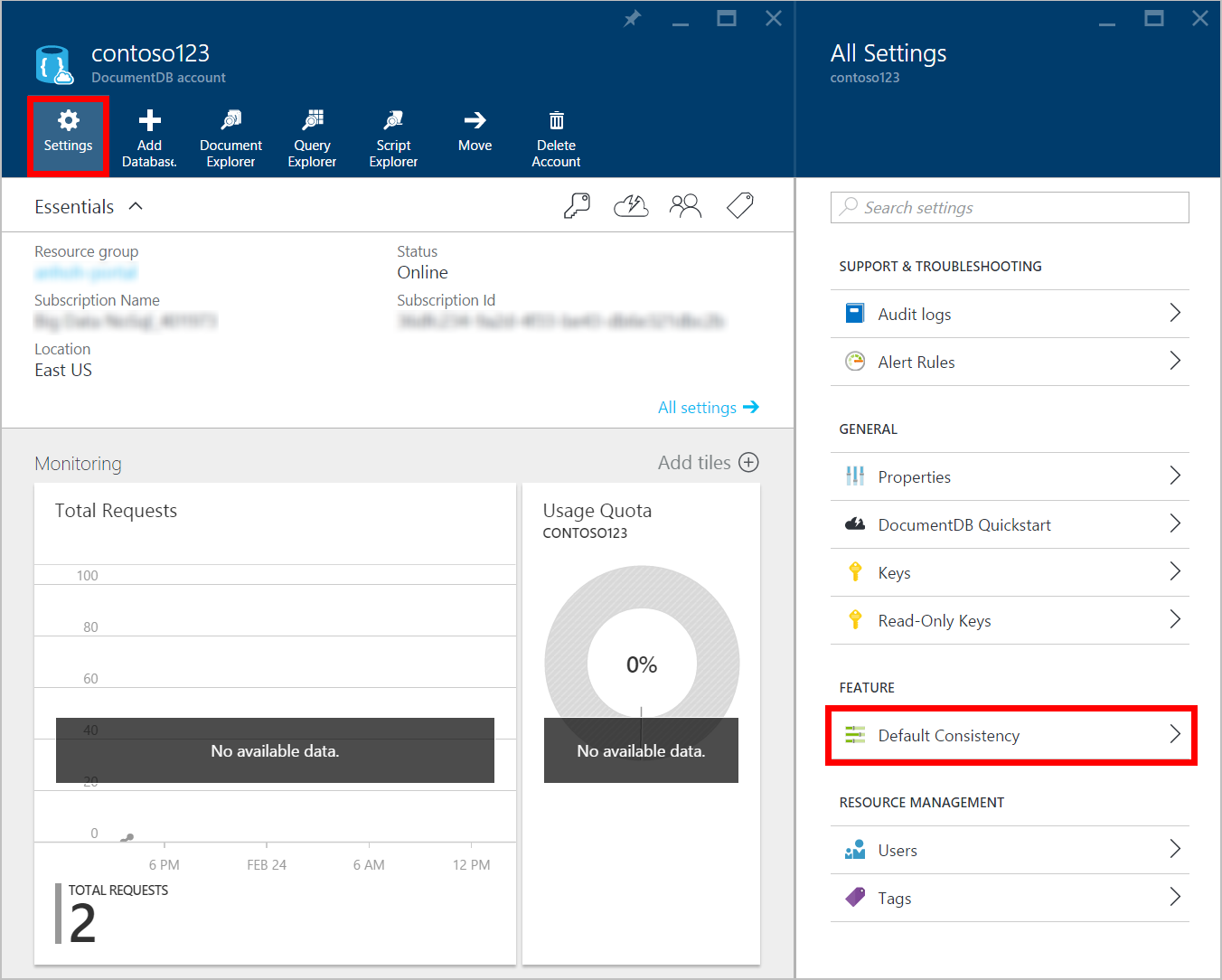
Session consistency provides predictable read data consistency for a session while offering the lowest latency writes. Reads are also low latency as except in the rare cases, the read will be served by a single replica.

**Eventual**: Eventual consistency is the weakest form of consistency wherein a client may get the values which are older than the ones it had seen before, over time. In the absence of any further writes, the replicas within the group will eventually converge. The read request is served by any secondary index.

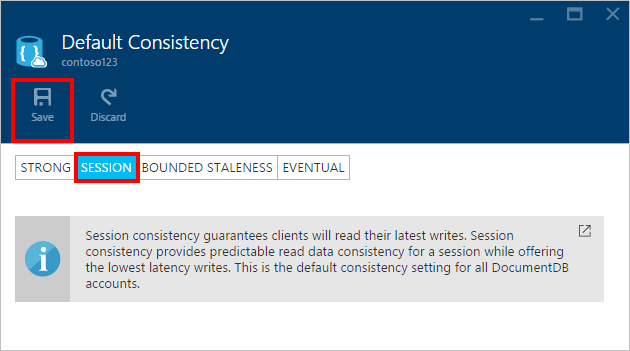
Eventual consistency provides the weakest read consistency but offers the lowest latency for both reads and writes.

### Changing the database consistency level

1. In the [Azure Portal](https://portal.azure.com/), in the Jumpbar, click **DocumentDB Accounts**.
2. In the **DocumentDB Accounts** blade, select the database account to modify.
3. In the account blade, if the **Settings** blade is not already opened, click the **Settings** icon on the top command bar.
4. In the **All Settings** blade, click on the **Default Consistency** entry under **Feature**.

* 
* Screen shot highlighting the Settings icon and Default Consistency entry

1. In the **Default Consistency** blade, select the new consistency level and click **Save**.

* 
* Screen shot highlighting the Consistency level and the Save button

## Consistency levels for queries

By default, for user defined resources, the consistency level of the queries is the same as the reads. By default, the index is updated synchronously on each insert, replace, or delete of a document to the collection. This enables the queries to honor the same consistency level as that of the document reads. While DocumentDB is write optimized and supports sustained volumes of document writes along with synchronous index maintenance and serving consistent queries, you can configure certain collections to update their index lazily. Lazy indexing further boosts the write performance and is ideal for bulk ingestion scenarios when a workload is primarily read-heavy.

|  |  |  |
| --- | --- | --- |
| Indexing Mode | Reads | Queries |
| Consistent (default) | Select from Strong, Bounded staleness, Session, or Eventual | Select from Strong, Bounded staleness, Session, or Eventual |
| Lazy | Select from Strong, Bounded staleness, Session, or Eventual | Eventual |

As with read requests, you can lower the consistency level of a specific query request by specifying the [x-ms-consistency-level] request header.

## Next steps

If you’d like to do more reading about consistency levels and tradeoffs, we recommend the following resources:

* Doug Terry. Replicated Data Consistency explained through baseball.  
  <http://research.microsoft.com/pubs/157411/ConsistencyAndBaseballReport.pdf>
* Doug Terry. Session Guarantees for Weakly Consistent Replicated Data.  
  <http://dl.acm.org/citation.cfm?id=383631>
* Daniel Abadi. Consistency Tradeoffs in Modern Distributed Database Systems Design: CAP is only part of the story”.  
  <http://computer.org/csdl/mags/co/2012/02/mco2012020037-abs.html>
* Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, Ion Stoica. Probabilistic Bounded Staleness (PBS) for Practical Partial Quorums.  
  <http://vldb.org/pvldb/vol5/p776_peterbailis_vldb2012.pdf>
* Werner Vogels. Eventual Consistent - Revisited.  
  <http://allthingsdistributed.com/2008/12/eventually_consistent.html>