**Understanding**

**Azure**

**DocumentDB**

**Table of Contents:**

[**1.** **Introduction to Azure’s DocumentDB** 3](#_Toc404122319)

[**1.1.** **Why DocumentDB** 3](#_Toc404122320)

[**1.2.** **Microsoft Azure Data Services** 4](#_Toc404122321)

[**1.3.** **Key capabilities and benefits:** 4](#_Toc404122322)

[**1.4.** **DocumentDB Limits for the Preview Release** 5](#_Toc404122323)

[**2.** **Introduction to Azure DocumentDb Resources** 6](#_Toc404122324)

[**2.1.** **Definition of Resources** 7](#_Toc404122325)

[**2.2.** **Capacity Units** 8](#_Toc404122326)

[**2.3.** **System vs. User defined Resources** 8](#_Toc404122327)

[**2.4.** **Indexing and it Configuration** 8](#_Toc404122328)

[**2.5.** **Optimistic Concurrency Control** 9](#_Toc404122329)

[**2.6.** **Connectivity Options** 9](#_Toc404122330)

[**2.7.** **Request Units** 10](#_Toc404122331)

[**2.8.** **Tuning Consistency** 10](#_Toc404122332)

[**2.9.** **Query Consistency** 11](#_Toc404122333)

[**3.** **Developing Against Azure DocumentDB** 11](#_Toc404122334)

[**3.1.** **Available SDKs** 12](#_Toc404122335)

[**3.2.** **Addressing a resource** 12](#_Toc404122336)

[**3.3.** **Basic Interaction Model** 12](#_Toc404122337)

[**3.4.** **Querying a Collection** 13](#_Toc404122338)

[**3.5.** **Multi-document transactions** 13](#_Toc404122339)

[**4.** **Setting up DocumentDB account in preview portal** 14](#_Toc404122340)

[**5.** **Demo with NodeJS** 15](#_Toc404122341)

[**6.** **Demo with ASP.Net MVC** 15](#_Toc404122342)

[**7.** **Glossary** 15](#_Toc404122343)

[**8.** **References** 15](#_Toc404122344)

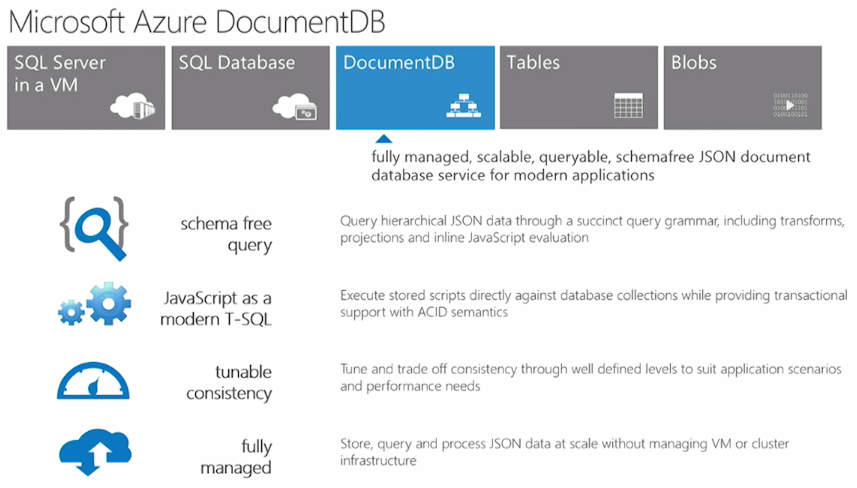
[**9.** **Appendix** 16](#_Toc404122345)

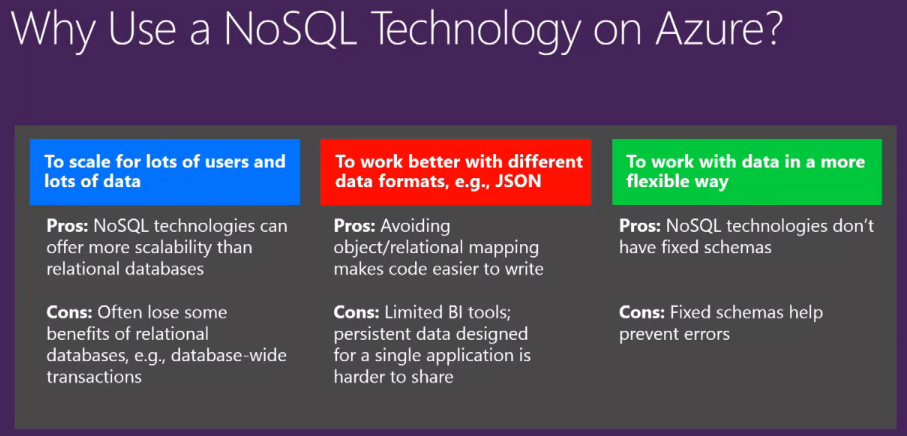
# **Introduction to Azure’s DocumentDB**

Many schema-free (NoSQL) databases do not allow non-trivial queries and transactional processing, making advanced data management hard. DocumentDB delivers consistently fast reads and writes, schema flexibility and the ability to easily scale a database up and down on demand. DocumentDB enables complex ad hoc queries using the SQL dialect, supports well defined consistency levels, and offers JavaScript language integrated, multi-document transaction processing using the familiar programming model of stored procedures, triggers and UDFs. Deep integration of JavaScript also allows developers to execute application logic efficiently directly within the database engine within a database transaction. Microsoft Azure DocumentDB is a NoSQL document database service designed for modern mobile and web applications.

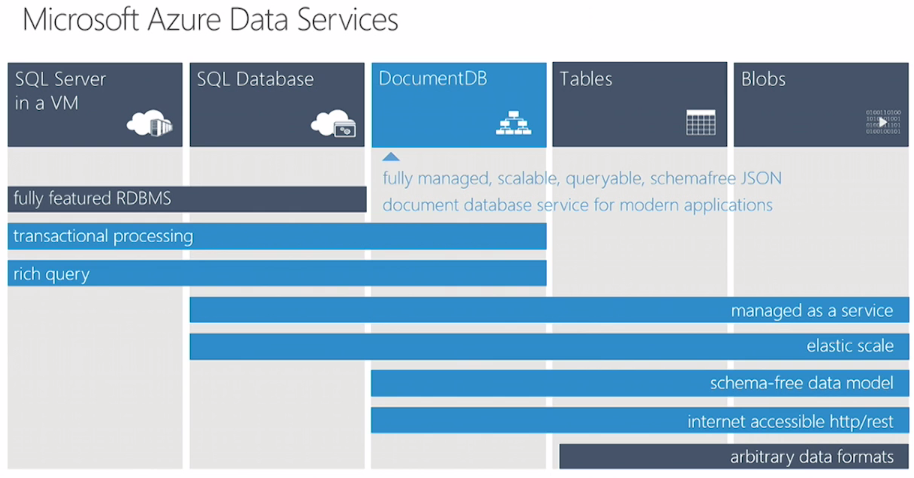
## **Why DocumentDB**

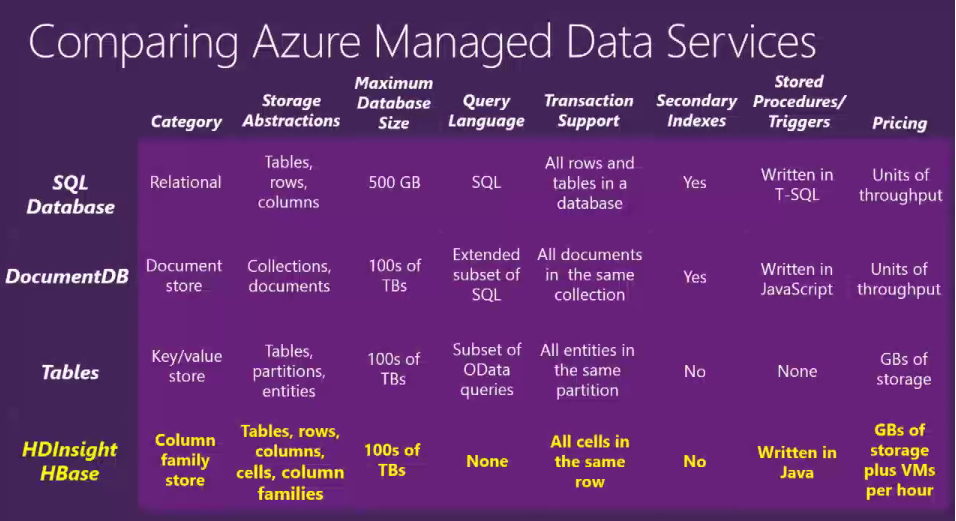
* Produce and consume data at staggering rate while requiring near instantaneous response time.
* Developed iteratively with many versions supported concurrently and continuously evolving data models.
* Increasingly complex, compensating for the lack of query and transactional processing in today’s NoSQL database systems.
* Supports unpredictable and explosive growth of data. Tuneable, high performance.
* It is a fully managed, highly scalable, queryable, truly schema-free JSON document database.
* It is delivered as service for modern applications. Designed as cloud first.
* Query over schema-free JSON. Multi-document transactions.





## **Microsoft Azure Data Services**





## **Key capabilities and benefits:**

**Ad hoc queries with familiar SQL syntax:** Store heterogeneous JSON documents within DocumentDB and query these documents through a familiar SQL syntax. DocumentDB utilizes a highly concurrent, lock free, log structured indexing technology to automatically index all document content. This enables rich real-time queries without the need to specify schema hints, secondary indexes or views.

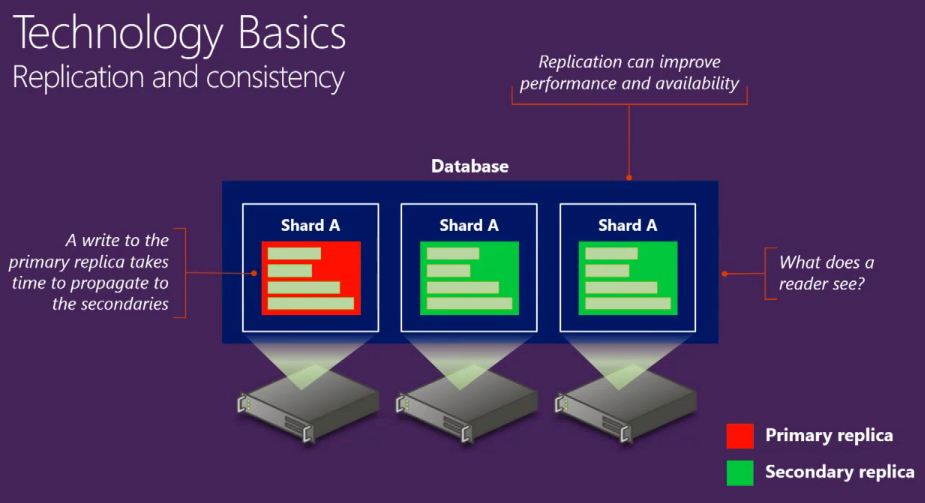
**JavaScript execution within the database:** Express application logic as stored procedures, triggers and user defined functions (UDFs) using standard JavaScript. This allows your application logic to operate over JSON data without worrying about the impedance mismatch between the application and the database schema. DocumentDB provides full transactional execution of JavaScript application logic directly inside the database engine. The deep integration of JavaScript enables the execution of INSERT, REPLACE, DELETE and SELECT operations from within a JavaScript program as an isolated transaction.

**Tunable consistency levels:** Select from four well defined consistency levels to achieve optimal trade-off between consistency and performance. For queries and read operations, DocumentDB offers four distinct consistency levels - Strong, Bounded-Staleness, Session, and Eventual.

**Fully managed:** Eliminate the need to manage database and machine resources. As a fully-managed Microsoft Azure service, you do not need to manage virtual machines, deploy and configure software, or deal with complex data-tier upgrades. Every database is automatically backed up and protected against regional failures. You can easily add a DocumentDB account and provision capacity as you need it, allowing you to focus on your application vs. operating and managing your database.

**Elastically scalable throughput and storage:** Easily scale up or down DocumentDB to meet your application needs. Scaling is done through fine grained units of reserved SSD backed storage and throughput. You can elastically scale DocumentDB with predictable performance by purchasing more capacity units, as your application grows.

**Open by design:** Get started quickly using existing skills and tools. Programming against DocumentDB is simple, approachable and does not require you to adopt new tools or adhere to custom extensions to JSON or JavaScript. You can access all of the database functionality including CRUD, query and JavaScript processing over a simple RESTful HTTP interface. DocumentDB embraces existing formats, languages and standards while offering high value database capabilities on top of them.



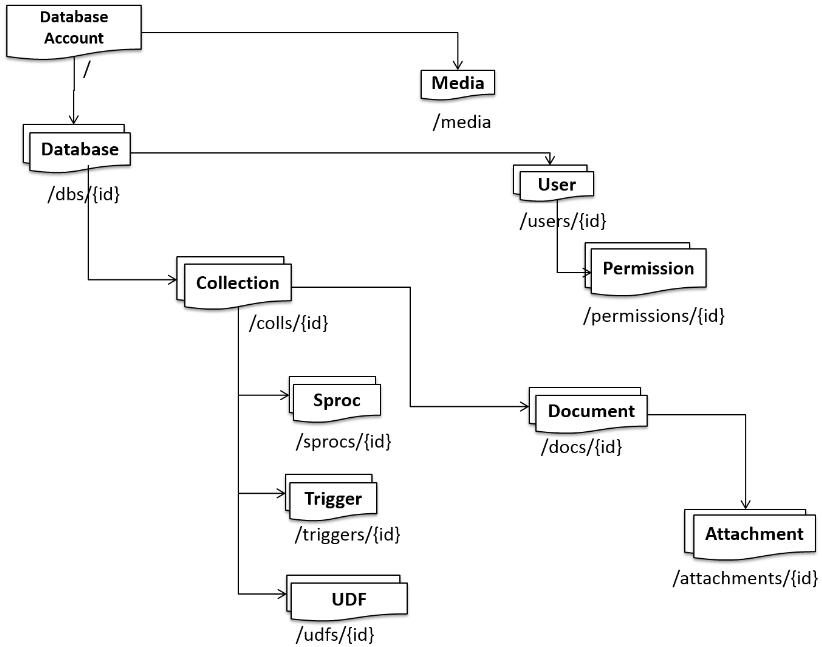
## **DocumentDB Limits for the Preview Release**

Below list are few of the items and its quota.

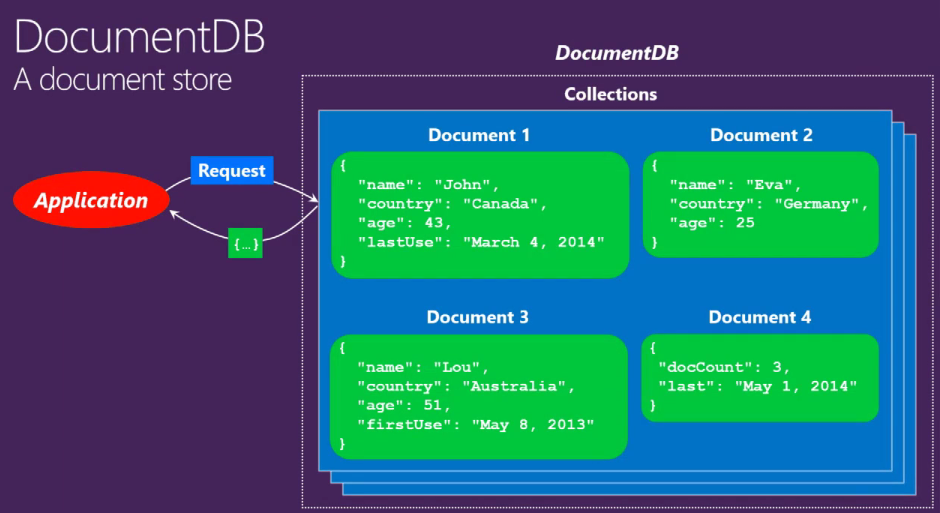
|  |  |
| --- | --- |
| **Entity** | **Quota** |
| Database Accounts | 5 |
| Number of databases per database account | 100 |
| Number of users per database account – across all databases | 500,000 |
| Number of permissions per database account – across all databases | 2,000,000 |
| Attachment storage per database account | 2 GB |
| Maximum number of capacity units per database account | 5 |
| Number of collections per capacity unit | 3 |
| Minimum Allocated Storage per collection with minimum 1 document | 3.3 GB |
| Minimum Allocated Throughput per collection with minimum 1 document | 667 RUs |
| Elasticity of a collection | 0-10 GB |
| Maximum Request Units / sec per collection | 2000 |
| Number of stored procedures, triggers and UDFs per collection | 25 each |
| Maximum execution time for stored procedure and trigger | 5 seconds |
| Provisioned document storage / capacity unit | 10 GB |
| Provisioned Request Units / sec / capacity unit | 2000 |
| Maximum document storage per database (5 capacity units) | 50 GB |
| Maximum Length of the Id property | 255 characters |
| Default number of items per page | 100 |
| Maximum items per page | 1000 |
| Maximum request size of document and attachment | 256KB |
| Maximum request size of stored procedure, trigger and UDF | 256KB |
| Maximum response size | 1MB |
| Maximum number of unique paths per collection | 100 |

# **Introduction to Azure DocumentDb Resources**

A database account is associated with one or more capacity units representing provisioned document storage and throughput, a set of databases and blob storage. You can create one or more database accounts using your Azure subscription. A database account consists of a set of databases, each containing multiple collections, each of which can contain stored procedures, triggers, UDFs, documents and related attachments. A database also has associated users each with a set of permissions to access various other collections, stored procedures, triggers, UDFs, documents or attachments.



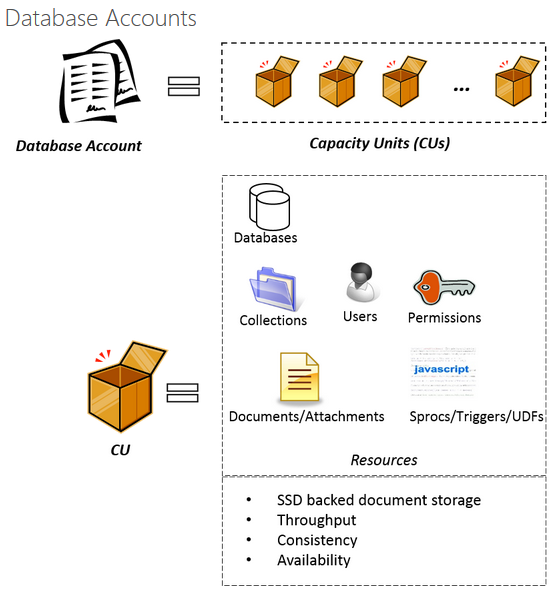
* Partitioned for scale out. Accounts scale out through addition of capacity units. Replicated for High Availability
* Entities represented as JSON. Entities addressable by logical Uri.
* RESTful interaction over HTTP with Standard HTTP verbs and semantics. HTTP and TCP connectivity.
* SDKs available to .Net, Node JS, Python, and Java Script. SDK libraries simplify many aspects of working with Azure DocumentDB by handling details such as address caching, exception management, automatic retries.
* SQL for query expression, .Net LINQ. Java Script for server-side application logic.
* The query and script execution interfaces are available through all platform libraries as well as the REST APIs.



## **Definition of Resources**

|  |  |
| --- | --- |
| **Resource** | **Description** |
| **Database** **Account** | A database account is associated with one or more capacity units representing provisioned document storage and throughput, a set of databases and blob storage. You can create one or more database accounts using your Azure subscription. Each database account has a unique DNS name. |
| **Database** | A database is a logical container of document storage partitioned across collections. It is also a container for users. |
| **User** | The logical namespace for scoping permissions. |
| **Permission** | An authorization token associated with a user for authorized access to a specific resource. |
| **Collection** | A collection is a container of JSON documents and associated JavaScript application logic. Queries and transactions are scoped to collections. |
| **Stored Procedure** | Application logic written in JavaScript which is registered with a collection and transactionally executed within the database engine. |
| **Trigger** | Application logic written in JavaScript modeling side effects associated with an insert, replace or delete operations. |
| **UDF** | A side effect free, application logic written in JavaScript. UDFs enable you to model a custom query operator and thereby extend the core DocumentDB query language. |
| **Document** | User defined (arbitrary) JSON content. By default, no schema needs to be defined or secondary indices need to be provided for all the documents added to a collection. |
| **Attachment** | Attachment are special documents containing references and associated metadata to an external blob/media. The developer can choose to have the blob managed by DocumentDB or store it with an external blob service provider such as OneDrive, Dropbox etc. |

## **Capacity Units**



## **System vs. User defined Resources**

Resources such as database accounts, databases, collections, users, permissions, stored procedures, triggers, and UDFs - all have a fixed schema and are called system resources. In contrast, resources such as documents and attachments have no restrictions on the schema and are examples of user defined resources. In DocumentDB, both system and user defined resources are represented and managed as standard compliant JSON. All resources, system or user defined have the following common properties. Note that all system generated properties in a resource are prefixed with an underscore (\_) in their JSON representation.

|  |  |  |
| --- | --- | --- |
| **Property** | **User settable or system generated?** | **purpose** |
| \_rid | System Generated | System generated, unique and hierarchical identifier of the resource |
| \_etag | System generated | etag of the resource required for optimistic concurrency control |
| \_ts | System generated | Last updated timestamp of the resource |
| \_self | System generated | Unique addressable URI of the resource |
| Id | User settable | User defined unique name of the resource |

## **Indexing and it Configuration**

As we add documents to a collection, DocumentDB automatically indexes them and they are available for us to query. Automatic indexing of documents without requiring schema or secondary indexes is a key capability of DocumentDB and is enabled by write-optimized, lock-free and log-structured index maintenance techniques.

The indexing policy of each collection allows us to make performance and storage trade-offs associated with indexing. The following options are available to us as part of indexing configuration:

Choose whether the collection automatically indexes all of the documents or not. By default, all documents are automatically indexed. You can choose to turn off automatic indexing and selectively add only specific documents to the index. Conversely, you can selectively choose to exclude only specific documents. You can achieve this by setting the automatic property to be true or false on the indexingPolicy of a collection and using the [x-ms-indexingdirective] request header while inserting, replacing or deleting a document.

Choose whether to include or exclude specific paths or patterns in your documents from the index. You can achieve this by setting includedPaths and excludedPaths on the indexingPolicy of a collection respectively. You can also configure the storage and performance tradeoffs for range and hash queries for specific path patterns.

Choose between synchronous (consistent) and asynchronous (lazy) index updates. 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 boosts the write performance further and is ideal for bulk ingestion scenarios for primarily read-heavy collections.

**Note:**

For the Preview release of DocumentDB, the indexing policy can only be configured during the creation of a collection. Once the collection has been created the policy cannot be updated.

## **Optimistic Concurrency Control**

Most Web applications rely on entity tag based Optimistic Concurrency Control to avoid the infamous “Lost Update” and “Lost Delete” problems. The entity tag is a HTTP friendly, logical timestamp associated with a resource. DocumentDB natively support the optimistic concurrency control by ensuring that every HTTP response contains the version (durably) associated with the specific resource. The concurrency control conflicts are correctly detected for the following cases:

* If two clients simultaneously issue mutating requests (via PUT/ DELETE verbs) on a resource with the latest version of the resource (specified via the [if-match] request header), the DocumentDB database engine subjects them to the transactional concurrency control.
* If a client presents with an older version of the resource (specified via the [if-match] request header), its request is rejected.

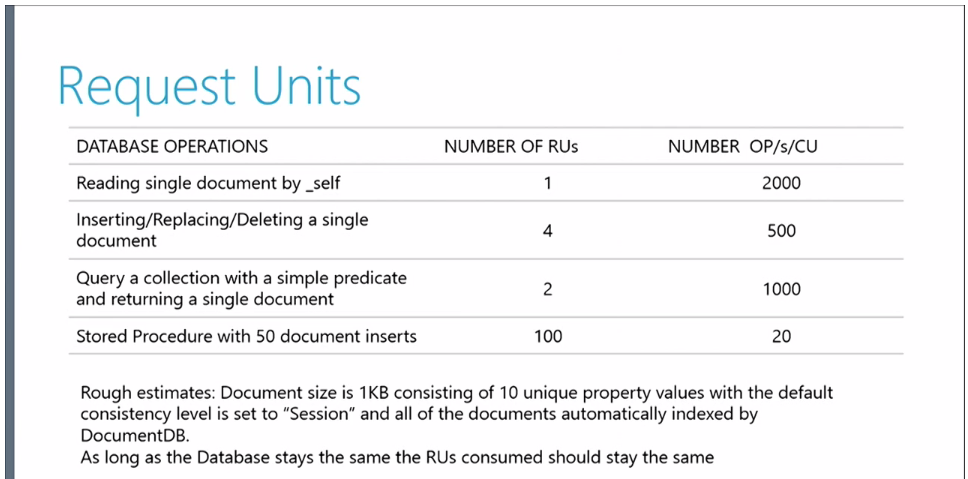
## **Connectivity Options**

DocumentDB exposes a logical addressing model wherein each resource has a logical and stable URI identified by its \_self link. As a distributed storage system spread across regions, the resources under various database accounts in DocumentDB are partitioned across numerous machines and each partition is replicated for high availability. The replicas managing the resources of a given partition register physical addresses. While the physical addresses change over the course of time due to failures, their logical addresses remain stable and constant. The logical to physical address translation is kept in a routing table which is also internally available as a resource. DocumentDB exposes two connectivity modes:

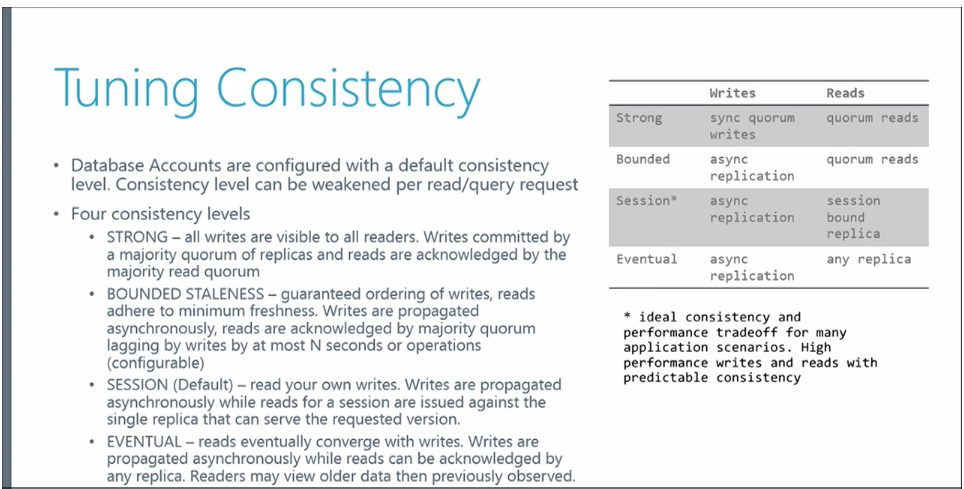
* **Gateway Mode**: The clients are shielded from the translation between logical to physical addresses or the details of the routing; they simply deal with logical URIs and RESTfully navigate the resource model. The clients issue the requests using logical URI and the edge machines translate the logical URI to the physical address of the replica which manages the resource and forwards the request. With the edge machines caching (and periodically refreshing) the routing table, routing is extremely efficient.
* **Direct Connectivity Mode**: The clients directly manage the routing table in their process space and periodically refresh it. Client can directly connect with the replicas and bypass the edge machines.

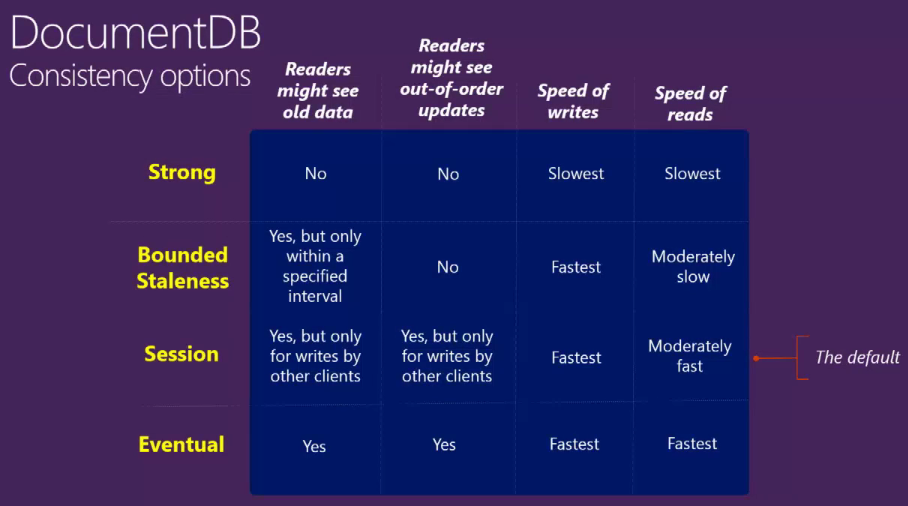
|  |  |  |  |
| --- | --- | --- | --- |
| **Mode** | **Protocol** | **Details** | **SDKs** |
| Gateway | Https | Applications directly connect with the edge nodes using logical URIs. This incurs an extra network hop. | REST APIs  .NET, JavaScript, Node.js, Python |
| Direct | Https & TCP | The applications can directly access the routing table and perform the client side routing to directly connect with replicas. | .NET |

## **Request Units**



## **Tuning Consistency**



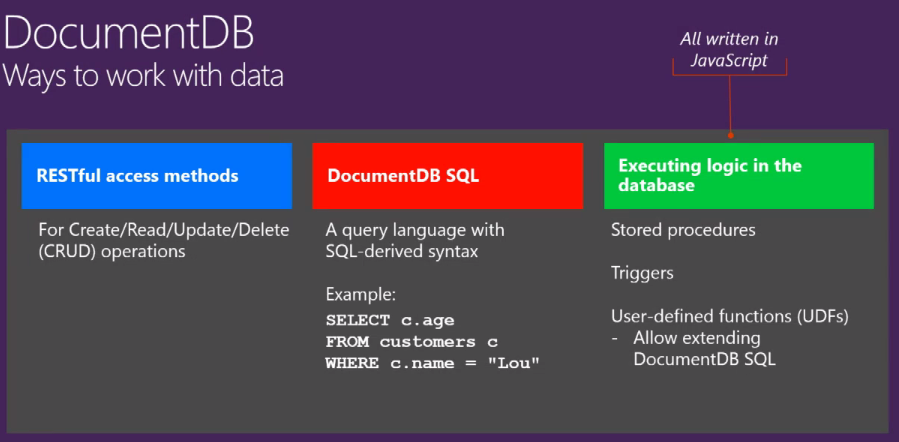


## **Query Consistency**

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 boosts the write performance further 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 |

# **Developing Against Azure DocumentDB**



## **Available SDKs**

Azure DocumentDB exposes resources via a REST API that can be called by any language capable of making HTTP/HTTPS requests. Additionally, Azure DocumentDB offers programming libraries for several popular languages. These libraries simplify many aspects of working with Azure DocumentDB by handling details such as address caching, exception management, automatic retries and so forth. Libraries are currently available for the following languages and platforms, with others on the way:

* .NET
* Node.js
* JavaScript
* Python

Beyond basic Create, Read, Update and Delete operations, Azure DocumentDB provides a rich SQL query interface for retrieving JSON documents and server side support for transactional execution of JavaScript application logic. The query and script execution interfaces are available through all platform libraries as well as the REST APIs.

## **Addressing a resource**

All resources are URI addressable. The value of the **\_self** property of a resource represents the relative URI of the resource. The format of the URI consists of the //{\_rid} path segments:

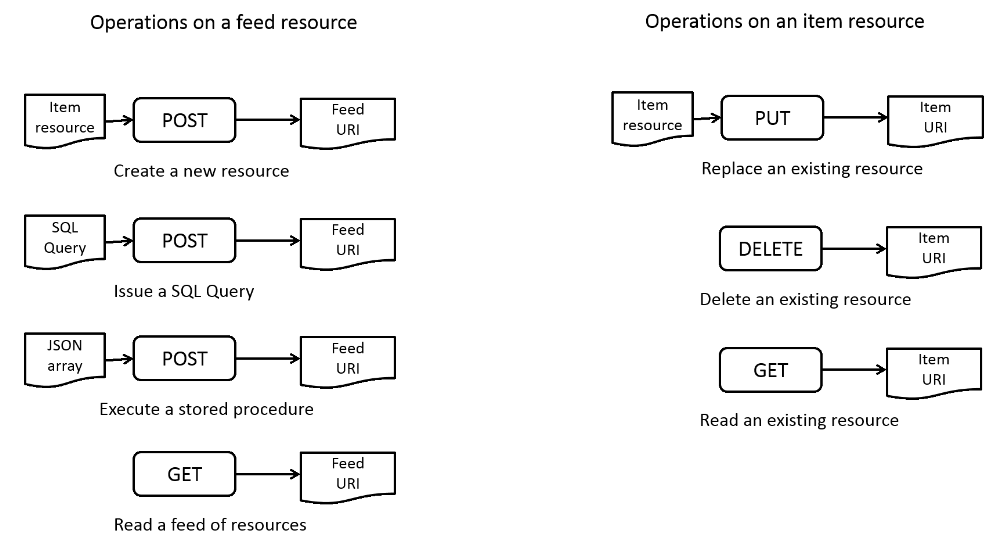
|  |  |
| --- | --- |
| **Value of the \_self** | **Description** |
| /dbs | feed of databases under a database account |
| /dbs/{\_rid-db} | Database with the unique id property with the value {\_rid-db} |
| /dbs/{\_rid-db}/colls/ | feed of collections under a database |
| /dbs/{\_rid-db}/colls/{\_rid-coll} | Collection with the unique id property with the value {\_rid-coll} |
| /dbs/{\_rid-db}/users/ | feed of users under a database |
| /dbs/{\_rid-db}/users/{\_rid-user} | User with the unique id property with the value {\_rid-user} |
| /dbs/{\_rid-db}/users/{\_rid-user}/permissions | feed of permissions under a database |
| /dbs/{\_rid-db}/users/{\_rid-user}/permissions/{\_rid-permission} | Permission with the unique id property with the value {\_rid-permission} |

## **Basic Interaction Model**

The resources support the following HTTP verbs with their standard interpretation:

* POST means create a new item resource.
* GET means read an existing item or a feed resource
* PUT means replace an existing item resource
* DELETE means delete an existing item resource
* HEAD means GET sans the response payload (i.e. just the headers)

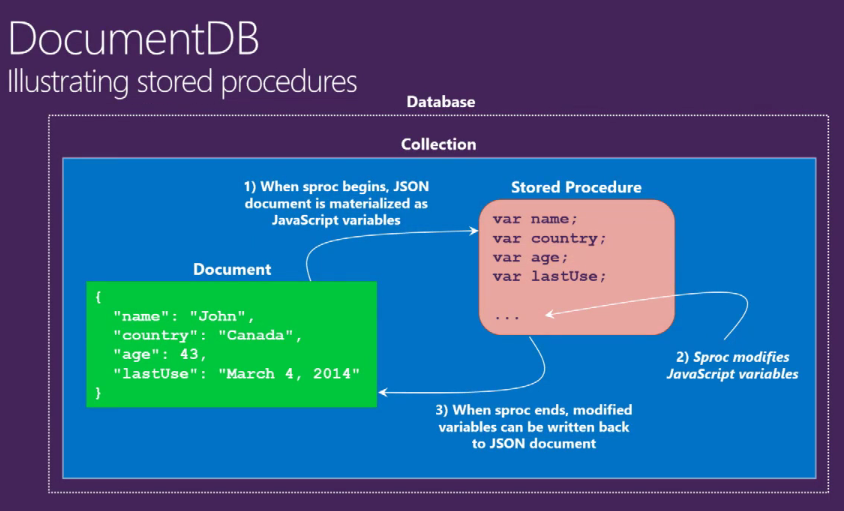
POST can only be issued against a Feed resource; PUT, DELETE can only be issued against an Item resource; GET and HEAD can be issued against either Feed or Item resources.



## **Querying a Collection**

The documents within a collection can have arbitrary schemas and you can query documents within a collection without providing any schema or secondary indices upfront. You can query the collection using the DocumentDB SQL query language which provides rich hierarchical and relational operators and extensibility via JavaScript-based UDFs. JSON grammar allows for modeling JSON documents as trees with labels as the tree nodes. This is exploited both by the DocumentDB’s automatic indexing techniques as well as, the SQL query dialect of DocumentDB. DocumentDB query language consists of three main aspects:

* A small set of query operations that map naturally to the tree structure including hierarchical queries and projections
* A subset of relational operations including composition, filter, projections, aggregates and self joins.
* Pure JavaScript based UDFs which compose with (1) and (2)

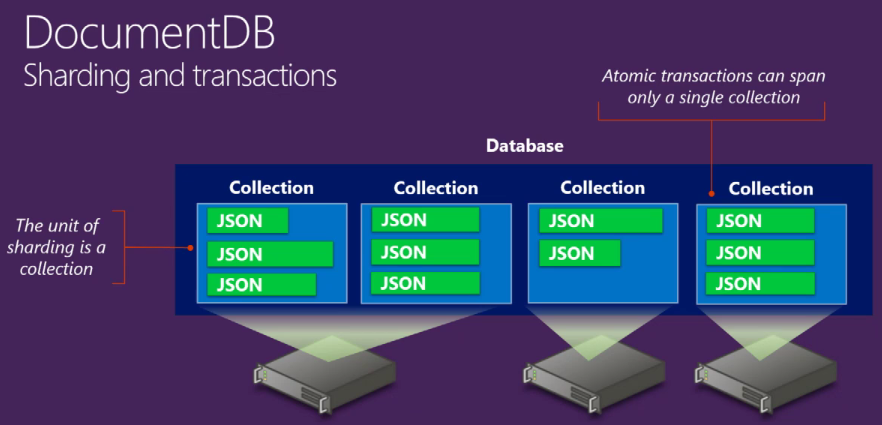


## **Multi-document transactions**

DocumentDB provides an intuitive programming model for executing JavaScript based application logic directly on the collections in terms of stored procedures and triggers. This allows for both, (a) efficient implementation of concurrency control, recovery, automatic indexing of the JSON object graphs directly in the database engine as well as, (b) naturally expressing control flow, variable scoping, assignment and integration of exception handling primitives with database transactions directly in terms of the JavaScript programming language.

The JavaScript logic registered at a collection level can then issue database operations on the documents of the given collection. DocumentDB implicitly wraps the JavaScript based stored procedures and triggers within an ambient ACID transactions with snapshot isolation across documents within a collection. During the course of its execution, if the JavaScript throws an exception, then the entire transaction is aborted. The resulting programming model is a very simple yet powerful. JavaScript developers get a “durable” programming model while still using their familiar language constructs and library primitives.

The ability to execute JavaScript directly within the database engine in the same address space as the buffer pool enables performant and transactional execution of database operations against the documents of a collection.



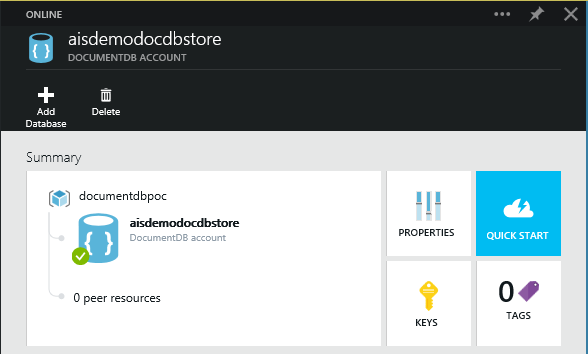
# **Setting up DocumentDB account in preview portal**

Below are the steps to create the DocumentDB account from preview azure portal.

* Sign in to the Azure management preview portal
* Click New -> DocumentDB Account. Alternatively, you can browse the Azure Gallery, select the “Data, storage, + backup” category, choose DocumentDB and then click Create.
* In the New DocumentDB (Preview) blade, specify the desired configuration for the DocumentDB account.
* Once the new DocumentDB account options are configured, click Create. It can take a few minutes for the DocumentDB account to be created.
* To check the status, you can monitor the progress on the startboard OR from the Notifications hub.
* After the DocumentDB account has been created, it is ready for use with the default settings
* You may also access your existing DocumentDB accounts from the Browse blade.

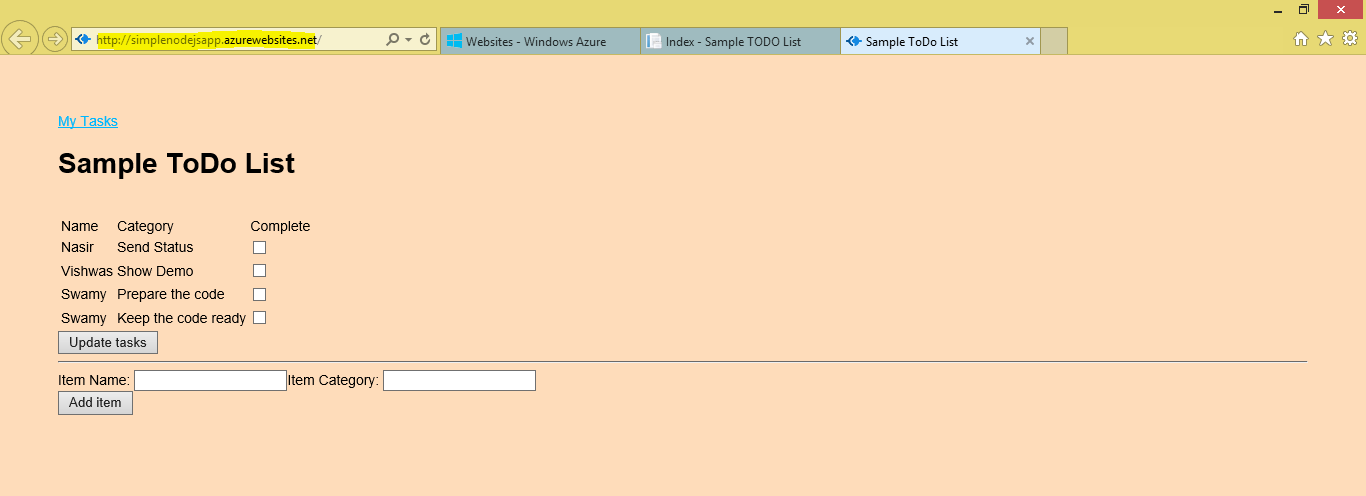
**Note:**

* You should have your own azure subscription.
* In Name, enter a name to use in the URI for the DocumentDB account. This value becomes the host name within the URI that is used to address the DocumentDB account. The name entry may contain only lowercase letters, numbers, and the '-' character and must be between 3 and 50 characters.



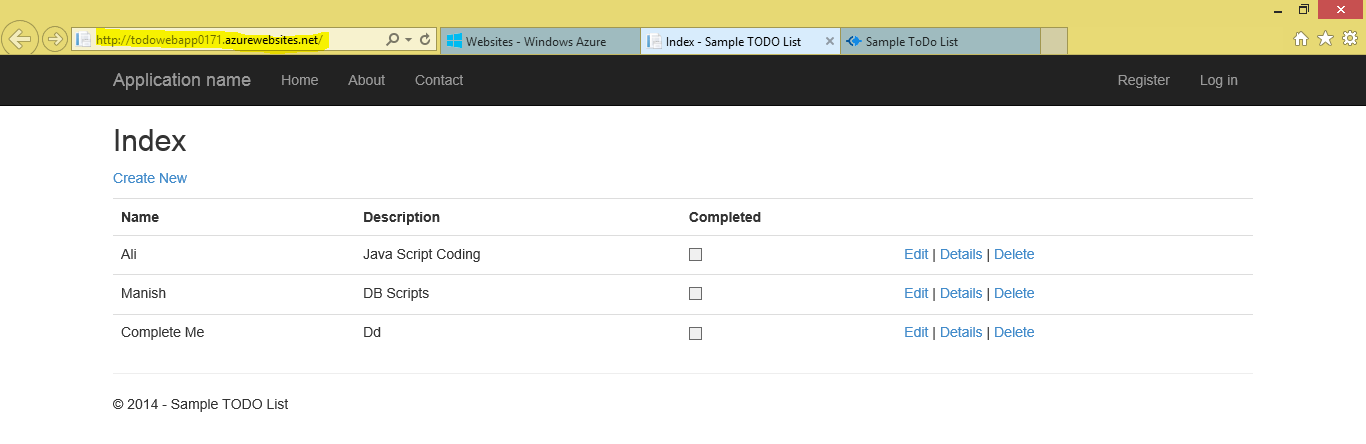
# **Demo with NodeJS**

Url: <http://simplenodejsapp.azurewebsites.net/>



# **Demo with ASP.Net MVC**

Url: <http://todowebapp0171.azurewebsites.net/>



# **Glossary**

|  |  |
| --- | --- |
| **Term** | **Description** |
| **UDF** | User Defined Function |
| **SDK** | Software Development Kit |
| **JSON** | JavaScript Object Notation |
| **SSD** | Solid-State Drive |
| **CRUD** | Create Read Update and Delete |

# **References**

<http://azure.microsoft.com/en-us/documentation/articles/documentdb-limits/>

<http://azure.microsoft.com/en-us/documentation/articles/documentdb-introduction/>

<http://channel9.msdn.com/Shows/Data-Exposed/Introduction-to-Azure-DocumentDB>

<http://channel9.msdn.com/events/TechEd/Europe/2014/DBI-B217>

# **Appendix**

