# NoSQL Node.js tutorial: DocumentDB Node.js console application

[AZURE.SELECTOR] - [.NET](/documentation/articles/documentdb-get-started) - [Node.js](/documentation/articles/documentdb-nodejs-get-started)

Welcome to the Node.js tutorial for the DocumentDB Node.js SDK! After following this tutorial, you’ll have a console application that creates and queries DocumentDB resources, including a Node database.

We’ll cover:

* Creating and connecting to a DocumentDB account
* Setting up your application
* Creating a node database
* Creating a collection
* Creating JSON documents
* Querying the collection
* Replacing a document
* Deleting a document
* Deleting the node database

Don’t have time? Don’t worry! The complete solution is available on [GitHub](https://github.com/Azure-Samples/documentdb-node-getting-started). See [Get the complete solution](#GetSolution) for quick instructions.

After you’ve completed the Node.js tutorial, please use the voting buttons at the top and bottom of this page to give us feedback. If you’d like us to contact you directly, feel free to include your email address in your comments.

Now let’s get started!

## Prerequisites for the Node.js tutorial

Please make sure you have the following:

* An active Azure account. If you don’t have one, you can sign up for a [Free Azure Trial](https://azure.microsoft.com/pricing/free-trial/).
* [Node.js](https://nodejs.org/) version v0.10.29 or higher.

## Step 1: Create a DocumentDB account

Let’s create a DocumentDB account. If you already have an account you want to use, you can skip ahead to [Setup your Node.js application](#SetupNode).

[AZURE.INCLUDE [documentdb-create-dbaccount](../includes/documentdb-create-dbaccount.md)]

## Step 2: Setup your Node.js application

1. Open your favorite terminal.
2. Locate the folder or directory where you’d like to save your Node.js application.
3. Create two empty JavaScript files with the following commands:

* Windows:
  + fsutil file createnew app.js 0
    - fsutil file createnew config.js 0
* Linux/OS X:
  + touch app.js
    - touch config.js

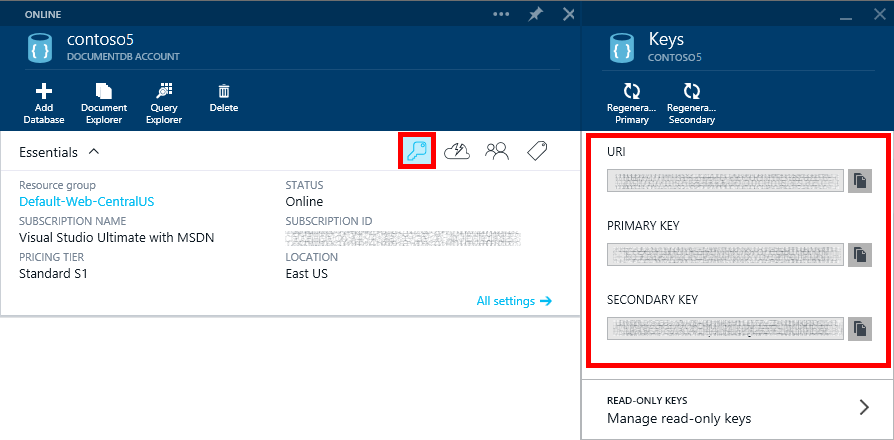
1. Install the documentdb module via npm. Use the following command:
   * npm install documentdb --save

Great! Now that you’ve finished setting up, let’s start writing some code.

## Step 3: Set your app’s configurations

Open config.js in your favorite text editor.

Then, copy and paste the code snippet below and set properties config.endpoint and config.primaryKey to your DocumentDB endpoint uri and primary key. Both these configurations can be found in the [Azure Portal](https://portal.azure.com).



Node.js tutorial - Screen shot of the Azure Portal, showing a DocumentDB account, with the ACTIVE hub highlighted, the KEYS button highlighted on the DocumentDB account blade, and the URI, PRIMARY KEY and SECONDARY KEY values highlighted on the Keys blade - Node database

// ADD THIS PART TO YOUR CODE  
var config = {}  
  
config.endpoint = "~your DocumentDB endpoint uri here~";  
config.primaryKey = "~your primary key here~";

Copy and paste the database id, collection id, and JSON documents to your config object below where you set your config.endpoint and config.authKey properties. If you already have data you’d like to store in your database, you can use DocumentDB’s [Data Migration tool](documentdb-import-data.md) rather than adding the document definitions.

config.endpoint = "~your DocumentDB endpoint uri here~";  
config.primaryKey = "~your primary key here~";  
  
// ADD THIS PART TO YOUR CODE  
config.database = {  
 "id": "FamilyDB"  
};  
  
config.collection = {  
 "id": "FamilyColl"  
};  
  
config.documents = {  
 "Andersen": {  
 "id": "Anderson.1",  
 "lastName": "Andersen",  
 "parents": [{  
 "firstName": "Thomas"  
 }, {  
 "firstName": "Mary Kay"  
 }],  
 "children": [{  
 "firstName": "Henriette Thaulow",  
 "gender": "female",  
 "grade": 5,  
 "pets": [{  
 "givenName": "Fluffy"  
 }]  
 }],  
 "address": {  
 "state": "WA",  
 "county": "King",  
 "city": "Seattle"  
 }  
 },  
 "Wakefield": {  
 "id": "Wakefield.7",  
 "parents": [{  
 "familyName": "Wakefield",  
 "firstName": "Robin"  
 }, {  
 "familyName": "Miller",  
 "firstName": "Ben"  
 }],  
 "children": [{  
 "familyName": "Merriam",  
 "firstName": "Jesse",  
 "gender": "female",  
 "grade": 8,  
 "pets": [{  
 "givenName": "Goofy"  
 }, {  
 "givenName": "Shadow"  
 }]  
 }, {  
 "familyName": "Miller",  
 "firstName": "Lisa",  
 "gender": "female",  
 "grade": 1  
 }],  
 "address": {  
 "state": "NY",  
 "county": "Manhattan",  
 "city": "NY"  
 },  
 "isRegistered": false  
 }  
};

The database, collection, and document definitions will act as your DocumentDB database id, collection id, and documents’ data.

Finally, export your config object, so that you can reference it within the app.js file.

},  
 "isRegistered": false  
 }  
};  
  
// ADD THIS PART TO YOUR CODE  
module.exports = config;

## Step 4: Connect to a DocumentDB account

Open your empty app.js file in the text editor. Copy and paste the code below to import the documentdb module and your newly created config module.

// ADD THIS PART TO YOUR CODE  
"use strict";  
  
var documentClient = require("documentdb").DocumentClient;  
var config = require("./config");  
var url = require('url');

Copy and paste the code to use the previously saved config.endpoint and config.primaryKey to create a new DocumentClient.

var config = require("./config");  
var url = require('url');  
  
// ADD THIS PART TO YOUR CODE  
var client = new documentClient(config.endpoint, { "masterKey": config.primaryKey });

Now that you have the code to initialize the documentdb client, let’s take a look at working with DocumentDB resources.

## Step 5: Create a Node database

Copy and paste the code below to set the HTTP status for Not Found, the database url, and the collection url. These urls are how the DocumentDB client will find the right database and collection.

var client = new documentClient(config.endpoint, { "masterKey": config.primaryKey });  
  
// ADD THIS PART TO YOUR CODE  
var HttpStatusCodes = { NOTFOUND: 404 };  
var databaseUrl = `dbs/${config.database.id}`;  
var collectionUrl = `${databaseUrl}/colls/${config.collection.id}`;

A [database](documentdb-resources.md#databases) can be created by using the [createDatabase](https://azure.github.io/azure-documentdb-node/DocumentClient.html) function of the **DocumentClient** class. A database is the logical container of document storage partitioned across collections.

Copy and paste the **getDatabase** function for creating your new database in the app.js file with the id specified in the config object. The function will check if the database with the same FamilyRegistry id does not already exist. If it does exist, we’ll return that database instead of creating a new one.

var collectionUrl = `${databaseUrl}/colls/${config.collection.id}`;  
  
// ADD THIS PART TO YOUR CODE  
function getDatabase() {  
 console.log(`Getting database:\n${config.database.id}\n`);  
  
 return new Promise((resolve, reject) => {  
 client.readDatabase(databaseUrl, (err, result) => {  
 if (err) {  
 if (err.code == HttpStatusCodes.NOTFOUND) {  
 client.createDatabase(config.database, (err, created) => {  
 if (err) reject(err)  
 else resolve(created);  
 });  
 } else {  
 reject(err);  
 }  
 } else {  
 resolve(result);  
 }  
 });  
 });  
}

Copy and paste the code below where you set the **getDatabase** function to add the helper function **exit** that will print the exit message and the call to **getDatabase** function.

} else {  
 resolve(result);  
 }  
 });  
 });  
}  
  
// ADD THIS PART TO YOUR CODE  
function exit(message) {  
 console.log(message);  
 console.log('Press any key to exit');  
 process.stdin.setRawMode(true);  
 process.stdin.resume();  
 process.stdin.on('data', process.exit.bind(process, 0));  
}  
  
getDatabase()  
.then(() => { exit(`Completed successfully`); })  
.catch((error) => { exit(`Completed with error ${JSON.stringify(error)}`) });

In your terminal, locate your app.js file and run the command: node app.js

Congratulations! You have successfully created a DocumentDB database.

## Step 6: Create a collection

[AZURE.WARNING] **CreateDocumentCollectionAsync** will create a new collection, which has pricing implications. For more details, please visit our [pricing page](https://azure.microsoft.com/pricing/details/documentdb/).

A [collection](documentdb-resources.md#collections) can be created by using the [createCollection](https://azure.github.io/azure-documentdb-node/DocumentClient.html) function of the **DocumentClient** class. A collection is a container of JSON documents and associated JavaScript application logic.

Copy and paste the **getCollection** function underneath the **getDatabase** function for creating your new collection with the id specified in the config object. Again, we’ll check to make sure a collection with the same FamilyCollection id does not already exist. If it does exist, we’ll return that collection instead of creating a new one.

} else {  
 resolve(result);  
 }  
 });  
 });  
}  
  
// ADD THIS PART TO YOUR CODE  
function getCollection() {  
 console.log(`Getting collection:\n${config.collection.id}\n`);  
  
 return new Promise((resolve, reject) => {  
 client.readCollection(collectionUrl, (err, result) => {  
 if (err) {  
 if (err.code == HttpStatusCodes.NOTFOUND) {  
 client.createCollection(databaseUrl, config.collection, { offerThroughput: 400 }, (err, created) => {  
 if (err) reject(err)  
 else resolve(created);  
 });  
 } else {  
 reject(err);  
 }  
 } else {  
 resolve(result);  
 }  
 });  
 });  
}

Copy and paste the code below the call to **getDatabase** to execute the **getCollection** function.

getDatabase()  
  
// ADD THIS PART TO YOUR CODE  
.then(() => getCollection())  
// ENDS HERE  
  
.then(() => { exit(`Completed successfully`); })  
.catch((error) => { exit(`Completed with error ${JSON.stringify(error)}`) });

In your terminal, locate your app.js file and run the command: node app.js

Congratulations! You have successfully created a DocumentDB collection.

## Step 7: Create a document

A [document](documentdb-resources.md#documents) can be created by using the [createDocument](https://azure.github.io/azure-documentdb-node/DocumentClient.html) function of the **DocumentClient** class. Documents are user defined (arbitrary) JSON content. You can now insert a document into DocumentDB.

Copy and paste the **getFamilyDocument** function underneath the **getCollection** function for creating the documents containing the JSON data saved in the config object. Again, we’ll check to make sure a document with the same id does not already exist.

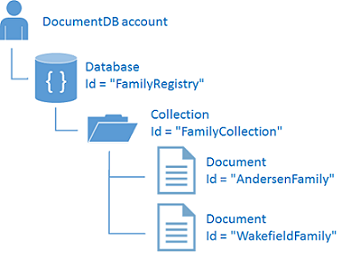
} else {  
 resolve(result);  
 }  
 });  
 });  
}  
  
// ADD THIS PART TO YOUR CODE  
function getFamilyDocument(document) {  
 let documentUrl = `${collectionUrl}/docs/${document.id}`;  
 console.log(`Getting document:\n${document.id}\n`);  
  
 return new Promise((resolve, reject) => {  
 client.readDocument(documentUrl, { partitionKey: document.district }, (err, result) => {  
 if (err) {  
 if (err.code == HttpStatusCodes.NOTFOUND) {  
 client.createDocument(collectionUrl, document, (err, created) => {  
 if (err) reject(err)  
 else resolve(created);  
 });  
 } else {  
 reject(err);  
 }  
 } else {  
 resolve(result);  
 }  
 });  
 });  
};

Copy and paste the code below the call to **getCollection** to execute the **getFamilyDocument** function.

getDatabase()  
.then(() => getCollection())  
  
// ADD THIS PART TO YOUR CODE  
.then(() => getFamilyDocument(config.documents.Andersen))  
.then(() => getFamilyDocument(config.documents.Wakefield))  
// ENDS HERE  
  
.then(() => { exit(`Completed successfully`); })  
.catch((error) => { exit(`Completed with error ${JSON.stringify(error)}`) });

In your terminal, locate your app.js file and run the command: node app.js

Congratulations! You have successfully created a DocumentDB documents.



Node.js tutorial - Diagram illustrating the hierarchical relationship between the account, the database, the collection, and the documents - Node database

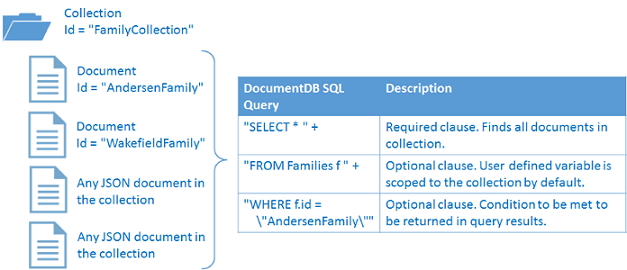
## Step 8: Query DocumentDB resources

DocumentDB supports [rich queries](documentdb-sql-query.md) against JSON documents stored in each collection. The following sample code shows a query that you can run against the documents in your collection.

Copy and paste the **queryCollection** function underneath the **getFamilyDocument** function. DocumentDB supports SQL-like queries as shown below. For more information on building complex queries, check out the [Query Playground](https://www.documentdb.com/sql/demo) and the [query documentation](documentdb-sql-query.md).

} else {  
 resolve(result);  
 }  
 });  
 });  
}  
  
// ADD THIS PART TO YOUR CODE  
function queryCollection() {  
 console.log(`Querying collection through index:\n${config.collection.id}`);  
  
 return new Promise((resolve, reject) => {  
 client.queryDocuments(  
 collectionUrl,  
 'SELECT VALUE r.children FROM root r WHERE r.lastName = "Andersen"'  
 ).toArray((err, results) => {  
 if (err) reject(err)  
 else {  
 for (var queryResult of results) {  
 let resultString = JSON.stringify(queryResult);  
 console.log(`\tQuery returned ${resultString}`);  
 }  
 console.log();  
 resolve(results);  
 }  
 });  
 });  
};

The following diagram illustrates how the DocumentDB SQL query syntax is called against the collection you created.



Node.js tutorial - Diagram illustrating the scope and meaning of the query - Node database

The [FROM](documentdb-sql-query.md#from-clause) keyword is optional in the query because DocumentDB queries are already scoped to a single collection. Therefore, “FROM Families f” can be swapped with “FROM root r”, or any other variable name you choose. DocumentDB will infer that Families, root, or the variable name you chose, reference the current collection by default.

Copy and paste the code below the call to **getFamilyDocument** to execute the **queryCollection** function.

.then(() => getFamilyDocument(config.documents.Andersen))  
.then(() => getFamilyDocument(config.documents.Wakefield))  
  
// ADD THIS PART TO YOUR CODE  
.then(() => queryCollection())  
// ENDS HERE  
  
.then(() => { exit(`Completed successfully`); })  
.catch((error) => { exit(`Completed with error ${JSON.stringify(error)}`) });

In your terminal, locate your app.js file and run the command: node app.js

Congratulations! You have successfully queried DocumentDB documents.

## Step 9: Replace a document

DocumentDB supports replacing JSON documents.

Copy and paste the **replaceDocument** function underneath the **queryCollection** function.

}  
 console.log();  
 resolve(result);  
 }  
 });  
 });  
}  
  
// ADD THIS PART TO YOUR CODE  
function replaceFamilyDocument(document) {  
 let documentUrl = `${collectionUrl}/docs/${document.id}`;  
 console.log(`Replacing document:\n${document.id}\n`);  
 document.children[0].grade = 6;  
  
 return new Promise((resolve, reject) => {  
 client.replaceDocument(documentUrl, document, (err, result) => {  
 if (err) reject(err);  
 else {  
 resolve(result);  
 }  
 });  
 });  
};

Copy and paste the code below the call to **queryCollection** to execute the **replaceDocument** function. Also, add the code to call **queryCollection** again to verify that the document had successfully changed.

.then(() => getFamilyDocument(config.documents.Andersen))  
.then(() => getFamilyDocument(config.documents.Wakefield))  
.then(() => queryCollection())  
  
// ADD THIS PART TO YOUR CODE  
.then(() => replaceFamilyDocument(config.documents.Andersen))  
.then(() => queryCollection())  
// ENDS HERE  
  
.then(() => { exit(`Completed successfully`); })  
.catch((error) => { exit(`Completed with error ${JSON.stringify(error)}`) });

In your terminal, locate your app.js file and run the command: node app.js

Congratulations! You have successfully replaced a DocumentDB document.

## Step 10: Delete a document

DocumentDB supports deleting JSON documents.

Copy and paste the **deleteDocument** function underneath the **replaceDocument** function.

else {  
 resolve(result);  
 }  
 });  
 });  
};  
  
// ADD THIS PART TO YOUR CODE  
function deleteFamilyDocument(document) {  
 let documentUrl = `${collectionUrl}/docs/${document.id}`;  
 console.log(`Deleting document:\n${document.id}\n`);  
  
 return new Promise((resolve, reject) => {  
 client.deleteDocument(documentUrl, (err, result) => {  
 if (err) reject(err);  
 else {  
 resolve(result);  
 }  
 });  
 });  
};

Copy and paste the code below the call to the second **queryCollection** to execute the **deleteDocument** function.

.then(() => queryCollection())  
.then(() => replaceFamilyDocument(config.documents.Andersen))  
.then(() => queryCollection())  
  
// ADD THIS PART TO YOUR CODE  
.then(() => deleteFamilyDocument(config.documents.Andersen))  
// ENDS HERE  
  
.then(() => { exit(`Completed successfully`); })  
.catch((error) => { exit(`Completed with error ${JSON.stringify(error)}`) });

In your terminal, locate your app.js file and run the command: node app.js

Congratulations! You have successfully deleted a DocumentDB document.

## Step 11: Delete the Node database

Deleting the created database will remove the database and all children resources (collections, documents, etc.).

Copy and paste the following code snippet (function **cleanup**) to remove the database and all the children resources.

else {  
 resolve(result);  
 }  
 });  
 });  
};  
  
// ADD THIS PART TO YOUR CODE  
function cleanup() {  
 console.log(`Cleaning up by deleting database ${config.database.id}`);  
  
 return new Promise((resolve, reject) => {  
 client.deleteDatabase(databaseUrl, (err) => {  
 if (err) reject(err)  
 else resolve(null);  
 });  
 });  
}

Copy and paste the code below the call to **deleteDocument** to execute the **cleanup** function.

.then(() => deleteFamilyDocument(config.documents.Andersen))  
  
// ADD THIS PART TO YOUR CODE  
.then(() => cleanup())  
// ENDS HERE  
  
.then(() => { exit(`Completed successfully`); })  
.catch((error) => { exit(`Completed with error ${JSON.stringify(error)}`) });

## Step 12: Run your Node.js application all together!

Altogether, the sequence for calling your functions should look like this:

getDatabase()  
.then(() => getCollection())  
.then(() => getFamilyDocument(config.documents.Andersen))  
.then(() => getFamilyDocument(config.documents.Wakefield))  
.then(() => queryCollection())  
.then(() => replaceFamilyDocument(config.documents.Andersen))  
.then(() => queryCollection())  
.then(() => deleteFamilyDocument(config.documents.Andersen))  
.then(() => cleanup())  
.then(() => { exit(`Completed successfully`); })  
.catch((error) => { exit(`Completed with error ${JSON.stringify(error)}`) });

In your terminal, locate your app.js file and run the command: node app.js

You should see the output of your get started app. The output should match the example text below.

Getting database:  
FamilyDB  
  
Getting collection:  
FamilyColl  
  
Getting document:  
Anderson.1  
  
Getting document:  
Wakefield.7  
  
Querying collection through index:  
FamilyColl  
 Query returned [{"firstName":"Henriette Thaulow","gender":"female","grade":5,"pets":[{"givenName":"Fluffy"}]}]  
  
Replacing document:  
Anderson.1  
  
Querying collection through index:  
FamilyColl  
 Query returned [{"firstName":"Henriette Thaulow","gender":"female","grade":6,"pets":[{"givenName":"Fluffy"}]}]  
  
Deleting document:  
Anderson.1  
  
Cleaning up by deleting database FamilyDB  
  
Completed successfully  
Press any key to exit

Congratulations! You’ve created you’ve completed the Node.js tutorial and have your first DocumentDB console application!

## Get the complete Node.js tutorial solution

To build the GetStarted solution that contains all the samples in this article, you will need the following:

* [DocumentDB account](/documentation/articles/documentdb-create-account).
* The [GetStarted](https://github.com/Azure-Samples/documentdb-node-getting-started) solution available on GitHub.

Install the **documentdb** module via npm. Use the following command: \* npm install documentdb --save

Next, in the config.js file, update the config.endpoint and config.authKey values as described in [Step 3: Set your app’s configurations](#Config).

## Next steps

* Want a more complex Node.js sample? See [Build a Node.js web application using DocumentDB](/documentation/articles/documentdb-nodejs-application).
* Learn how to [monitor a DocumentDB account](/documentation/articles/documentdb-monitor-accounts).
* Run queries against our sample dataset in the [Query Playground](https://www.documentdb.com/sql/demo).
* Learn more about the programming model in the Develop section of the [DocumentDB documentation page](/documentation/services/documentdb/).