NoSQL and Data Scalability 2.0

UPDATED BY **ADAM FOWLER**
ORIGINAL BY **EUGENE CIURANA**

CONTENTS

INTRODUCTION

This NoSQL Refcard provides an easy-to-understand and useful set of information about the range of NoSQL databases available today.

SCALABLE DATA ARCHITECTURES

Scalable data architectures have evolved to improve overall system efficiency and reduce operational costs. Specific NoSQL databases may have different topological requirements, but the general architecture is the same.

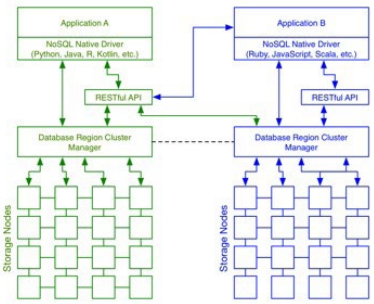

Figure 1: *NoSQL Architecture*

In general, NoSQL architectures offer:

- A range of consistency options, as opposed to the ACID-only consistency of relational databases
- High availability, some with partition tolerance (Cassandra) and some with ACID consistency (ArangoDB)
- Horizontal scalability on commodity hardware, instead of a reliance on single, large servers (optimized for large volume of reads and queries)
- Many have scale out / scale back support, for use on dynamically provisioned Cloud environments
- Distributed data storage, with four different options of formats for stored data: key-value, columnar, document, and triple/graph

Cloud readiness describes the database being used as a service and the ability to deploy the database software to a cloud provider.

NOSQL

NoSQL describes a horizontally scalable, non-relational database with built-in replication support. Applications interact with the database through a simple API, and the data is stored in a schema-free repository as large files or data blocks. The repository is often a custom file system designed to support NoSQL operations with high replication.

NoSQL is short for Not Only SQL, and refers to the fact that non-relational data can benefit from multiple different query mechanisms. The Structured Query Language (SQL) of relational systems is also supported by many NoSQL databases. This is useful for access from legacy software platforms, including Business Intelligence (BI) tools that do not support NoSQL databases natively.

NOSQL DATABASES CLASSIFICATION

There are four key types of NoSQL databases. The simplest are also the fastest, so there is a trade off on functionality when using a key-value store. The four types are below:

DATABASE CLASS	BRIEF DESCRIPTION	PRODUCT EXAMPLES
Key-value	Stores data as key-values. Over 1.5 million transactions per second possible. Useful, for example, for high speed access to web advertisements. Some support more sophisticated data structures, including lists, sets, counters, and maps.	Amazon DynamoDB, Redis, Aerospike
Columnar	A simple row key with many columns. Columns belong to named column families. Columns in the same column family are stored together, making retrieval very useful. No relationships between columns in different tables.	HBase, Accumulo, Microsoft CosmosDB, Hypertable, Cassandra

AWS NoSQL Database Services

Amazon DynamoDB is a fast and flexible NoSQL Database service for all applications that need consistent, single-digit millisecond latency at any scale.

Try it today for free: aws.amazon.com/dynamodb

Amazon ElastiCache

µs is the new ms

Amazon ElastiCache is a fully-managed service for Redis and Memcached that makes it easy to deploy, operate, and scale an in-memory data store or cache in the cloud with microsecond response times.

Try it today for free: aws.amazon.com/elasticache

amazon web services

Learn More ››

DATABASE CLASS	BRIEF DESCRIPTION	PRODUCT EXAMPLES
Document	Stores hierarchical JSON data. Some support XML and other formats. Maps very well to programming languages' object graphs. Most popular NoSQL database option with developers. Typically paired with a search engine to handle complex unstructured text.	MongoDB, MarkLogic, CouchDB, Couchbase, ArangoDB, OrientDB, Microsoft CosmosDB, IBM Cloudant, Amazon DynamoDB
Triple or Graph	Very simple structures in a directed graph. Each piece of data is a triple – Subject, Predicate, and Object. This technology underpins the Semantic Web. Triple stores are used to store webs of information with semantic inferencing, while graph stores are used for minimum distance (e.g. route planning applications) and other graph traversal problems.	Neo4j, GraphDB, Allegrograph, MarkLogic, OrientDB, ArangoDB
Hybrid, or multi-model	Supports two or more of the above types of data. Most common pairing is document and triple/graph stores.	**Document/Triple**: MarkLogic **Document/Graph**: OrientDB, ArangoDB **Document/Columnar**: Microsoft CosmosDB **Key-value/Document**: Amazon DynamoDB

While all database types are in common use, document stores are most often associated with NoSQL systems due to their pervasiveness in web and mobile content handling applications.

IS NOSQL FOR YOU?

Does your app design...

- Need to handle varying data structures (schema), or have schema that you do not control?
- Require high-speed throughput?
- Need to handle high volumes of data?
- Work well with weak data consistency, or need different consistency models at different times?
- Benefit from direct object-database entity mapping?
- Is operational, and not batch (unlike Hadoop applications)?

If you checked off four or more items from the list, then NoSQL is a good fit for you.

NOSQL TRADE OFFS

The total cost of ownership (TCO) is often lower for NoSQL databases than relational databases. This is thanks mainly to two things. First, many NoSQL databases have an open source core.

Second, they scale out on commodity hardware – i.e. a very large dataset does not require a very powerful, and very expensive, single computer. Instead you can use multiple, small computer servers – or even better, scale out in a virtualized cloud infrastructure like Amazon Web Services (AWS).

I've put together a few data points that illustrate the trade-offs. I've included relational databases for comparison. Note these show relative scores compared to each other, not absolute scores in real terms.

	RELATIONAL	KEY-VALUE	COLUMNAR	DOCUMENT	TRIPLE/GRAPH
Data model complexity	Medium	Low	Medium	High	High
Breadth of data model applicability	Low	Medium	Medium	High	High
Ease of schema change	Low	Very high	Medium	High	Very high
Performance	Medium	Very high	High	Medium	Highly variable – query dependent
Scale out cost	High	Low	Low	Low	Varies on architecture Sharded: Low, Unsharded: High
TCO for very large operational volumes	High	Low	Medium	Medium	Varies on architecture Sharded: Medium Unsharded: High

Figure 2: *Complexity and TCO*

Document and key-value stores are most popular because of their ease of use, flexibility, and applicability across many problem domains—at a reasonable TCO.

Tip: Graph databases are excellent replacements for complex relational models because relationships between entities (or graph edges) are more efficient and better suited for high-performance applications than using explicit joins and foreign-keys. This is especially the case for computational complex graph traversal algorithms such as minimum distance or sub graph comparison.

Tip: Many NoSQL vendors make over 50% of their revenue from consultancy. Be sure to interrogate suppliers for full project consultancy costs for your final analysis of TCO. Consultancy rates up to USD $2000 per day are possible for some NoSQL databases. NoSQL vendor-trained System Integration (SI) partners are a good source of experienced yet reasonably priced consultancy.

WHICH DATA MODEL TO USE?

The flowchart in Figure 3 describes how to choose the most appropriate database or store for the application.

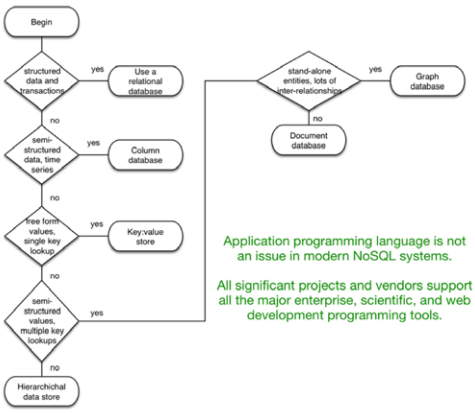

Figure 3: *Choosing the Right Data Store*

HYBRID OR MULTI-MODEL DATABASES

Many NoSQL databases are moving toward supporting multiple models. This means they may be key-value stores, that also support the storage and querying of JSON documents – such as Amazon DynamoDB.

Other NoSQL databases are supporting both the document and graph or triple store models. Examples of these include MarkLogic Server, ArangoDB, and OrientDB.

Which you choose depends primarily on how you query the data, as you can see in Figure 3. Start with what questions you are going to ask of your data, and then look at the most convenient storage model – such as cells (with column families, perhaps) or more hierarchical JSON documents.

If in doubt, start with a simple database structure that also has support for secondary indexes. Amazon DynamoDB is a good candidate here, as it stores simple JSON values natively in its key-value store, but also provides secondary indexes to pull back records and data summaries, much like more complex document stores do.

CLOUD DATABASES

Demand-based scaling is an attractive proposition for running NoSQL systems on the cloud; it maximizes the advantages of running the application on cloud-based providers like AWS, Microsoft Azure, or Google Cloud.

- Database-as-a-Service (DBaaS) offers turnkey managed functionality, which delegates all operational responsibilities to the provider.
- Hosted VM databases are provisioned on virtual images, much like they would be on premises, and all operational responsibility belongs to the user. All NoSQL databases can be used in this way.

Some NoSQL databases are available as cloud-friendly turnkey DBaaS. Some of these are listed below:

- Amazon DynamoDB on AWS
- Microsoft CosmosDB on Microsoft Azure
- MongoDB Atlas on AWS, Google Cloud (GCP), or Microsoft Azure
- IBM Cloudant on IBM Bluemix

Tip: Billing overruns are very easy when using a Database-as-a-service. Engage a usage/cost monitoring system to help manage expenses and to avoid nasty surprises. Most cloud platforms have this built in, such as AWS Charges Alerts and Notifications.

NOSQL IN PRACTICE

This section will use Amazon DynamoDB to illustrate key key-value store traits, including real-life use cases and architectures. Document database use cases are also covered briefly using DynamoDB, thanks to its storage of JSON values and secondary indexes, allowing record queries.

AMAZON DYNAMODB

DynamoDB is a key-value NoSQL database that supports eventual and strong consistency. It is a very simple to use service, and can be run standalone on a laptop or in the cloud on Amazon Web Services (AWS).

There are many use cases for DynamoDB specifically, and key-value stores in general:

- Service advertisements for web pages with sub-second response time
- Storing user preferences for a web site
- Storing temporary 'session' information, such as a shopping cart

An example architecture for using DynamoDB as a ad serving database can be found at https://media.amazonwebservices.com/architecturecenter/AWS_ac_ra_adserving_06.pdf

DynamoDB in particular is useful for web application developers, as it has a friendly API with wrappers for Node.js, Java, and other languages. It also stores and retrieves data in web app-friendly JSON format.

This data can be retrieved by a row or partition key like other key-value stores. You can also add secondary indexes to support querying by different attributes. These indexes allow for more sophisticated query mechanisms.

QUICK START GUIDE FOR DYNAMODB

This quick start guide is a modified version of Amazon DynamoDB on Node.js tutorial. The version presented below is a realistic web application to search and retrieve movie information from DynamoDB and present it on a web page.

This is the fundamental functionality of any web application, and should allow you to get up and running for your own apps very quiclly.

RUNNING DYNAMODB LOCALLY

Our first step is to download a copy of DynamoDB and run it locally. There is a very simple tutorial to do this on Amazon's website: http://docs.aws.amazon.com/amazondynamodb/latest/gettingstartedguide/GettingStarted.Download.htm

You download the .tar.gz or.zip for your platform, unzip the files, and then execute the service. This assumes you have Java installed locally.

I've created a folder called nodejs-dynamodb-sample. You can download a complete copy of this from my GitHub Page: github.com/adamfowleruk/nodejs-dynamodb-sample

Click on 'Download Zip' to get the full repository contents.

Within this file I've created a folder called 'ext' that I've unpacked the DynamoDB files in to. You should do this yourself, now.

I've then created a shell script (Linux, Mac) and batch file (Windows) that executed the below code:

```
java -Djava.library.path=./DynamoDBLocal_lib -jar ./ext/
DynamoDBLocal.jar -sharedDb –inMemory
```

For convenience you can open up a command prompt and just execute the `run-dynamodb-local.sh` or .bat file.

Note: You can find all the code here on my GitHub site. You will have to download DynamoDB yourself and unpack it in to the ext folder before running those files.

CREATING YOUR WEB APPLICATION WITH NODE.JS EXPRESS

First you'll need to download the DynamoDB SDK for Node.js. This tutorial assumes you have a working Node.js environment. If you don't, visit nodejs.org and download the latest version.

First, ensure the Express module is installed on your system, globally. This is not part of the GitHub download, so you must execute it yourself.

```
npm install –g express-generator
```

Download the sample app from GitHub, and unpack it. Now open a command prompt and move to this folder:

```
cd nodejs-dynamodb-sample
```

Now type:

```
npm install
```

After a few minutes, all of your dependency files for this application will be installed.

RUN THE SAMPLE APP

You'll now test to ensure that your web application is installed with all dependencies, and working.

From within the sample app folder, type this:

```
DEBUG=express:* npm start
```

After a short while you will see 'running on port 3000.'

Now open a browser to http://localhost:3000/

You will see a welcome page, and two search forms. These forms won't work yet as we need to configure your AWS access for DynamoDB.

CONFIGURING AWS SECURITY

In order to use DynamoDB, you'll need to register for a free AWS account, and generate an Access Key.

Register for an AWS account here: aws.amazon.com

Once registered and logged in, search for the IAM service and click on it.

IAM is AWS' Identity and Access Management service. You will need to create a user in order to store data in S3 and, later on, access the DynamoDB service on AWS (we're using a local service for now on your own computer).

Click on 'Create Individual IAM Users' and then 'Manage Users'. The page will look like this.

Now click on Add User. Use a logical username. For an example, refer to this image.

Now click on 'Next: Permissions' and click on 'Create Group' like so.

This will open up a new window. Configure a new named group with 'AmazonS3FullAccess' and 'AmazonDynamoDBFullAccess' policies. Click 'Create Group'. You should see a summary like this.

Return to the Create User window in your browser, and click 'Next: Review' then 'Next: Complete.'

Here you will see your access key, along with a secret key. Click on 'Show' then jot both the access key and secret key down somewhere safe. Click on 'Done' when you have finished.

Create a key file for AWS access.

Create this file:

- Linux users: `~/.aws/credentials`
- Windows users: `C:\Users\USER_NAME\.aws\credentials`

Now take the Access Key and Secret Key and in this file, add them as below:

```
[default]
aws_access_key_id = <YOUR_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_SECRET_ACCESS_KEY>
```

Note: You can have multiple configurations. For simplicity, we're using the default configuration.

CREATING THE TABLE

In the command prompt, execute:

```
node MoviesCreateTable.js
```

You should see an output like this:

```
adamfowookwork2:nodejs-dynamodb-sample adamfowler$ node
MoviesCreateTable.js
Created table. Table description JSON: {
  “TableDescription”: {
    “AttributeDefinitions”: [
      {
        “AttributeName”: “year”,
        “AttributeType”: “N”
      },
      {
        “AttributeName”: “title”,
        “AttributeType”: “S”
      }
    ],
    “TableName”: “Movies”,
    “KeySchema”: [
      {
        “AttributeName”: “year”,
        “KeyType”: “HASH”
      },
      {
        “AttributeName”: “title”,
        “KeyType”: “RANGE”
      }
    ],
    “TableStatus”: “ACTIVE”,
    “CreationDateTime”: “2017-08-13T09:52:48.719Z”,
    “ProvisionedThroughput”: {
      “LastIncreaseDateTime”: “1970-01-01T00:00:00.000Z”,
      “LastDecreaseDateTime”: “1970-01-01T00:00:00.000Z”,
      “NumberOfDecreasesToday”: 0,
      “ReadCapacityUnits”: 10,
      “WriteCapacityUnits”: 10
    },
    “TableSizeBytes”: 0,
    “ItemCount”: 0,
    “TableArn”:
“arn:aws:dynamodb:ddblocal:000000000000:table/Movies”
  }
}
adamfowookwork2:nodejs-dynamodb-sample adamfowler$
```

If you don’t, you probably have either copied the access key and secret key incorrectly, or did not add both the S3 Full Access and DynamoDB full access policies to the IAM user’s group.

LOAD THE DATA

Now execute the load data script, like below:

```
node MoviesLoadData.js
```

This should take 5-10 seconds to load, and loads 5000 movies in to your new database, stored in memory.

Now we’re going to vary from the Amazon tutorial again. We’re going to configure a /movies URL in your Express Node.js web application. This page will respond to the following URLs:

`GET /movies?year=1985&title=A+View+to+a+Kill` – Fetch a specific movie by year and title

`POST /movies` – List movies from a particular year using a key’ed field in DynamoDB

Now your security has been configured, you should simply be able to re-run your Node.js application: `DEBUG=express:* npm start`

Now visit loalhost:3000/ and type in ‘1985’ and ‘A View to a Kill.’ (Correct capitalization is very important) in to the Get form. Click Get. Note that only one movie is shown.

Now go back to the index page, and type in a year in the Search form. Click Search.

Express uses Jade for web page templating. To see what is happening, read the following files:

1. The execution code for /movies in ./routes/movies.js
2. The display for the results in ./views/movies.jade

Note that there are two routes configured in movies.js – a GET route and a POST route. Each route does something slightly different. The first fetches a specific single movie, the second lists movies using an indexed field.

From this basic example you can move on to create your own application. You could use DynamoDB to:

- Store User information and site preferences for your website
- Store game data, high scores
- Store shopping cart or other temporary data
- Much, much more

For further details, read all the links in the Amazon DynamoDB for Node.js documentation: docs.aws.amazon.com/amazondynamodb/latest/gettingstartedguide/GettingStarted.NodeJs.html

MOVING YOUR APP TO THE AWS HOSTED DYNAMODB

Now we’ll reconfigure the application to use the online DynamoDB service rather than the in-memory service. This means that your data will be saved between executions of your application, just like a real world web application.

Warning: Be sure to delete your tables once you’ve finished using them, else you could incur AWS hosting and service costs if you exceed Free Tier usage. These instructions can be found at the end of this tutorial.

To move this application to AWS, first shut down your local DynamoDB instance by issuing Ctrl+C in the command prompt.

Now reconfigure your web application by editing ./shared/aws-config.js.

Change the URL from localhost:3000 to https://dynamodb.eu-west-1.amazonaws.com.

Note: You may want to use us-west-2 or another region title instead of eu-west-1

Now because we’re using a different DynamoDB instance, we’ll need to recreate the table and load items. Execute these two scripts again:

```
node MoviesCreateTable.js
node MoviesLoadData.js
```

You may receive error message when loading data as we're loading 5000 movies which exceeds the default throughput quota. Don't worry about this. Once complete, or after you hit Ctrl+C to stop the load, you will have enough Movies to run your web application again.

Start the express application again:

```
DEBUG=express:* npm start
```

Now revisit localhost:3000 and type in another year, such as 1984.

Note how the application responds much faster using a hosted Cloud version of DynamoDB rather than the local version.

MONITORING USAGE AND COSTS

You can see how much storage you've used by visiting the DynamoDB console on AWS. In the AWS management console, search for the DynamoDB service.

Click on Tables and you should see a list like this.

By clicking on 'Movies,' you can view the items within the table on the Items table, access metrics from your application, and see estimated monthly costs in the Capacity tab.

To ensure we don't get charged anything, under Actions next to Create Table, click on Delete table. Confirm this action.

The code in this exercise can be found on my GitHub page: github.com/adamfowleruk/nodejs-dynamodb-sample

SUMMARY

In this tutorial you have learned:

- The 4 different categories of NoSQL database
- Where each can be used, and their advantages and disadvantages
- How to create a Node.js application and use Amazon DynamoDB in the Cloud
- How to track and manage cloud NoSQL costs

ABOUT THE AUTHOR

ADAM FOWLER Fowler is based in the UK and is the author of the books "NoSQL for Dummies" and "State of NoSQL 2016", and maintains an active blog on Data Management and NoSQL at adamfowler.org. Adam also curates important NoSQL news on his twitter feed @adamfowleruk.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com