TERM PAPER – CSP 554

ADITYA PRAKASH YAJI – A20426486

Comparison and Evaluation of security from HBase, Redis, Cosmos DB and AWS DynamoDB.

ABSTRACT

The increasing popularity of Big Data and Cloud computing is the moving power for the organizations to migrate from Relational Databases towards Non-Relational databases referring as NoSQL popularly called as "NOT ONLY SQL". NoSQL gained popularity since April 2009 and from that time till date we have been using NoSQL databases for managing huge data generated every day. The goal of this paper is to survey, compare and evaluate the security features of few popular NoSQL databases such as HBase, Redis, Azure Cosmos DB and AWS Dynamo DB. [1]

INTRODUCTION

Relational databases have been leading model for data management, storage and retrieval for several years. But due to increased demands for scalability and performance, new systems have emerged, such as NoSQL. NoSQL databases have been designed for utilizing distributed, collaborating hosts for storing and retrieving the data. There are four different classes of NoSQL databases, these are Key value stores, Document databases, Column-oriented databases and Graph databases. Each type is precisely used for some specific type of dataset. The only thing common between all these datasets are, they are used to manage non-relational datasets. One can store huge amount of data into the NoSQL Databases without any failure. In recent years, a growing number of companies have adopted various types of NoSQL databases, and as the applications they serve, emerge and they gain extensive market interest. These new database systems are not relational by definition and therefore they do not support full SQL functionality. Moreover, as opposed to relational databases they trade consistency and security for performance and scalability. As large amounts of user related sensitive information are stored in NoSQL databases, security issues become growing concerns. [2] [3]

Why do we need NoSQL databases?

Relational databases have following challenges:

Relational databases are not suitable for large volume of data with different data types (such as image, videos, text) and we also cannot scale up for large volumes of data. Relational databases are limited by memory and CPU capabilities. Also, they can't scale out, as they are limited by cache dependent read and write operations. Shrading (break database into pieces and store in different nodes) causes operational problems i.e. managing a shared failure. Relational databases have complex RDBMS model and consistency limits the scalability.

To overcome above challenges, NoSQL databases came into picture. They provide continuous availability; low latency rate and they can handle changes. NoSQL databases support multiple data structure. It's a scale-out, shared-nothing architecture, capable of running on many nodes. They have a non-locking concurrency control mechanism so that real-time reads will not conflict writes. When compared to RDBMS, this has the architecture that gives high performance per node. Also, this is a schema-less data mode.

NEED FOR SECURITY OF NOSQL

Data breaches are a serious concern for any enterprise, especially as the frequency and the severity of the security breaches are increasing. So, securing the NoSQL database should be a top priority. There is always a belief that the attacks would increase substantially. NoSQL has grown over the years overcoming the limitations of the Relational database, but these databases have their own unique security considerations as many enterprises opt to run NoSQL deployments in the cloud. Hence there is a strong need to secure the NoSQL database.

Rather than providing the security issues of NoSQL databases to all the databases I have summarized based on few assessment criteria's such as:

1. **Authentication:** The process of verifying a user's identity who wants to access the resources, data or applications of an organization is known as authentication. A few of the well-known authentication techniques include Password based authentication, Multi-Factor authentication, Certificate based authentication, authentication protocols like SSL, SSH and Kerberos etc. [4]
2. **Access Controls:** Access control is the mechanism through which we can ensure that only an authorized person is allowed to access system resources. Access control can be applied at system, database, object and content level depending on the configurations of the database administrator. Three of the most conventional access control models include Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role Based Access control (RBAC) model. [4]
3. **Secure Configurations:** A security breach may easily occur due to misconfigurations at the OS, database or application layer therefore, NOSQL database must include a list of configurations that can be applied by the system administrators to secure databases uniformly across clusters at both physical and logical level. Some of the important categories of secure configurations include securing of patches and updates, services, protocols, roles and accounts, files and directories, backups, ports and registries etc. [4]
4. **Data Encryption:** Data Encryption is used to provide confidentiality of data and applications in a database system. It includes the encryption of data-at-rest as well as the encryption of data-in-transit over the network. A few of the well-known standard encryption techniques such as SSL, IPSec, TLS and SSH etc. Moreover, management of encryption keys is also an important factor of data encryption and should be handled with proper care. [4]
5. **Auditing:** Database Auditing refers to the monitoring and recording of individual and collective actions performed by database users. Auditing helps in the identification of foot prints or possible password cracking attempts before the occurrence of an attack. Some of the important auditing activities include auditing of database connections, privileged activities and transaction logs. [4]

The NoSQL attack surface is diverse. The Scheme, Query and JavaScript injection attacks affect different solutions differently. There is a need to understand how these attacks affect the application and NoSQL Environment. However, the enterprise which makes use of a NoSQL database, should not rush to implement NoSQL without evaluating the security implications of switching from RDBMS to NoSQL database systems. From the available resources and the products designed to implement NoSQL, we see that the NoSQL has not been designed with security as a priority. Developers or the security teams must add a security layer to their organizations NoSQL applications. The below section will provide a brief description and overview of the security of 4 different types of NoSQL databases.

DISCUSSON ON SECURITY OF NOSQL DATABASES

HBase

Apache HBase is “the Apache Hadoop database”, a horizontally scalable nonrelational datastore built on top of components offered by the Apache Hadoop ecosystem, notably Apache ZooKeeper and Apache Hadoop HDFS. Although HBase therefore offers first class Hadoop integration, and is often chosen for that reason, it has come into its own as a good choice for high scale data storage of record. HBase is often the kernel of big data infrastructure.

There are many advantages offered by HBase: it is free open source software, it offers linear and modular scalability, it has strictly consistent reads and writes, automatic and configurable sharding and automatic failover are core concerns, and it has a deliberate tolerance for operation on “commodity” hardware, which is a very significant benefit at high scale. As more organizations are faced with Big Data challenges, HBase is increasingly found in roles formerly occupied by traditional data management solutions, gaining new users with new perspectives and the requirements and challenges of new use cases. Some of those users have a strong interest in security. They may be a healthcare provider operating within a strict regulatory regime regarding the access and sharing of patient information. They might be a consumer web property in a jurisdiction with strong data privacy laws. They could be a military or government agency that must manage a strict separation between multiple levels of information classification.

Assessing the security features of the HBase database based on the assessment criteria:

1. **Authentication:**

HBase not only supports token-based authentication for map-reduce tasks but user authentication is also supported by HBase. The user authentication is done by using SASL (Simple Authentication and Security Layer) with Kerberos on per connection basis.

2. **Access Controls:**

HBase Access Controls is based upon the Remote Procedure call protocol (RPC). Once the user authentication is done then the next target is to allow the user to access only certain sets of data tables. For this purpose, HBase has brought in Access Control List (ACL) which describes set of rules for user to authorize a particular set of data. In ACL each entry describes the user and the data to which he has the access. Thus, whenever a user requests for an access it uses the authentication mechanism which is based on Kerberos protocol to identify the user identity and then it checks in ACL for its appropriate entry in order to give the access to the data.

3. **Secure Configurations:**

There are two main approaches for securing communication configuration with and within cloud-based HBase clusters: HBase native security and an HBase-independent architecture based on virtual private clouds (VPC) and VPN.

4. **Data Encryption:**

HBase is working over HDFS and HDFS support encryption so when the data is stored, they are encrypted using the HDFS encryption method. HBase does not have column-level encryption feature out of box. We should use Dataguise or go with the option to develop a custom UDF for encryption and decryption separately using some algorithm.

5. **Auditing:**

HBase is working over HDFS and the auditing feature is derived from HDFS.HDFS is at the core of Hadoop, providing the distributed file system that makes Hadoop so successful. HDFS has two different audit logs, hdfs-audit.log for user activity and SecurityAuthhdfs.audit for service activity. Both of these logs are implemented with Apache Log4j, a common and well-known mechanism for logging in Java.

Let's talk more about new "**cell**" based security for HBase which was copied form another NoSQL database called "**Accumulo"**.

High scale data storage options in the Apache family, notably Apache Accumulo, take a different approach. Accumulo has a data model almost identical to that of HBase but implements a security mechanism called cell-level security. Every cell in an Accumulo store can have a label, stored effectively as part of the key, which is used to determine whether a value is visible to a given subject or not. The label is not an ACL, it is a different way of expressing security policy. An ACL says explicitly what subjects are authorized to do what. A label instead turns this on its head and describes the sensitivity of the information to a decision engine that then figures out if the subject is authorized to view data of that sensitivity based on (potentially, many) factors. This enables data of various security levels to be stored within the same row, and users of varying degrees of access to query the same table, while enforcing strict separation between multiple levels of information classification. HBase users might approximate this model using ACLs, but it would be labour intensive and error prone. [17]

New HBase Cell Security Features

Intel has been busy extending HBase with cell level security features. First, contributed as **HBASE-8496**, HBase can now store arbitrary metadata for a cell, called tags, along with the cell. Then, as of **HBASE-7662**, HBase can store into and apply ACLs from cell tags, extending the current HBase ACL model down to the cell. Then, as of **HBASE-7663**, HBase can store visibility expressions into tags, providing cell-level security capabilities similar to Apache Accumulo, with API and shell support that will be familiar to Accumulo users. Finally, we have also contributed transparent server side encryption, as **HBASE-7544**, for additional assurance against accidental leakage of data at rest. We are working with the HBase community to make these features available in the next major release of HBase, 0.98. [17]

HBase Visibility Labels

We have introduced a new coprocessor, the VisibilityController, which can be used on its own or in conjunction with HBase's AccessController (responsible for ACL handling). The VisibilityController determines, based on label metadata stored in the cell tag and associated with a given subject, if the user is authorized to view the cell. The maximal set of labels granted to a user is managed by new shell commands getauths, setauths, and clearauths, and stored in a new HBase system table. Accumulo users will find the new HBase shell commands familiar.

When storing or mutating a cell, the HBase user can now add visibility expressions, using a backwards compatible extension to the HBase API. (By backwards compatible, we mean older servers will simply ignore the new cell metadata, as opposed to throw an exception or fail.)

Mutation#setCellVisibility(new CellVisibility(String labelExpression));

The visibility expression can contain labels joined with logical expressions '&', '|' and '!'. Also using '(', ')' one can specify the precedence order. For example, consider the label set { confidential, secret, topsecret, probationary }, where the first three are sensitivity classifications and the last describes if an employee is probationary or not. If a cell is stored with this visibility expression:

(secret | topsecret) & !probationary

Then any user associated with the secret or topsecret label will be able to view the cell, as long as the user is not also associated with the probationary label. Furthermore, any user only associated with the

confidential label, whether probationary or not, will not see the cell or even know of its existence. Accumulo users will also find HBase visibility expressions familiar, but also providing a superset of boolean operators.

We build the user's label set in the RPC context when a request is first received by the HBase RegionServer. How users are associated with labels is pluggable. The default plugin passes through labels specified in Authorizations added to the Get or Scan. This will also be familiar to Accumulo users.

Get#setAuthorizations(new Authorizations(String,…));

Scan#setAuthorizations(new Authorizations(String,…));

Authorizations not in the maximal set of labels granted to the user are dropped. From this point, visibility expression processing is very fast, using set operations.

In the future we envision additional plugins which may interrogate an external source when building the effective label set for a user, for example LDAP or Active Directory. Perhaps the sensitivity classifications are attached when cells are stored into HBase, but the probationary label, determined by the user's employment status, is provided by an external authorization service. [17]

Redis

Redis is a document-oriented, highly available and distributed NoSQL data store. Unlike other NoSQL databases, Redis supports three kinds of partitioning namely "client-side partitioning", "proxy-assisted partitioning" and "Query routing". The client-side partitioning gives rights to the Redis client to directly select the appropriate node for the storage of data keys. Proxy assisted partitioning, on the other hand, needs a query to be delivered to a proxy server instead of sending it directly to the appropriate Redis instance. The query routing partitioning suggests that a client query can be sent to any instance of the Redis cluster, and it is the responsibility of the instance to forward client query to the appropriate instance. However, partitioning can only be survived in the scenarios where the majority of the master nodes are reachable in a cluster having master-slave configurations. Redis also supports master-slave replication within its cluster for the efficiency of reads and data redundancy.

Redis provides password-based authentication to its clients. These passwords are stored in plain text format and set by the system administrators. However, it does not provide authentication by default and listens on all IP addresses on port 6739. In addition, Redis does not ensure any kind of access control mechanisms and rely on third-party SSL implementations for the security of data transmissions over untrusted networks. Redis also does not provide support for configuration security, data encryption and auditing mechanisms.[8]

Redis general security model

Redis is designed to be accessed by trusted clients inside trusted environments. This means that usually it is not a good idea to expose the Redis instance directly to the internet or, in general, to an environment where untrusted clients can directly access the Redis TCP port or UNIX socket.

For instance, in the common context of a web application implemented using Redis as a database, cache, or messaging system, the clients inside the front-end (web side) of the application will query Redis to generate pages or to perform operations requested or triggered by the web application user.

In this case, the web application mediates access between Redis and untrusted clients (the user browsers accessing the web application).

This is a specific example, but, in general, untrusted access to Redis should always be mediated by a layer implementing ACLs, validating user input, and deciding what operations to perform against the Redis instance.

In general, Redis is not optimized for maximum security but for maximum performance and simplicity.[6]

Network security

Access to the Redis port should be denied to everybody but trusted clients in the network, so the servers running Redis should be directly accessible only by the computers implementing the application using Redis.

In the common case of a single computer directly exposed to the internet, such as a virtualized Linux instance (Linode, EC2, ...), the Redis port should be firewalled to prevent access from the outside. Clients will still be able to access Redis using the loopback interface.

Note that it is possible to bind Redis to a single interface by adding a line like the following to the redis.conf file:

```
bind 127.0.0.1
```

Failing to protect the Redis port from the outside can have a big security impact because of the nature of Redis. For instance, a single FLUSHALL command can be used by an external attacker to delete the whole data set.[6]

Protected mode

Unfortunately, many users fail to protect Redis instances from being accessed from external networks. Many instances are simply left exposed on the internet with public IPs. For these reasons since version 3.2.0, when Redis is executed with the default configuration (binding all the interfaces) and without any password in order to access it, it enters a special mode called protected mode. In this mode Redis only replies to queries from the loopback interfaces and reply to other clients connecting from other addresses with an error, explaining what is happening and how to configure Redis properly. We expect protected mode to seriously decrease the security issues caused by unprotected Redis instances executed without proper administration, however the system administrator can still ignore the error given by Redis and just disable protected mode or manually bind all the interfaces.[6]

Authentication feature

While Redis does not try to implement Access Control, it provides a tiny layer of authentication that is optionally turned on editing the redis.conf file.
When the authorization layer is enabled, Redis will refuse any query by unauthenticated clients. A client can authenticate itself by sending the AUTH command followed by the password.
The password is set by the system administrator in clear text inside the redis.conf file. It should be long enough to prevent brute force attacks for two reasons:

1. Redis is very fast at serving queries. Many passwords per second can be tested by an external client.
2. The Redis password is stored inside the redis.conf file and inside the client configuration, so it does not need to be remembered by the system administrator, and thus it can be very long.

The goal of the authentication layer is to optionally provide a layer of redundancy. If firewalling or any other system implemented to protect Redis from external attackers fail, an external client will still not be able to access the Redis instance without knowledge of the authentication password.

The AUTH command, like every other Redis command, is sent unencrypted, so it does not protect against an attacker that has enough access to the network to perform eavesdropping.[6]

Data encryption support

Redis does not support encryption. In order to implement setups where trusted parties can access a Redis instance over the internet or other untrusted networks, an additional layer of protection should be implemented, such as an SSL proxy. We recommend spiped.[9][6]

Disabling of specific commands

It is possible to disable commands in Redis or to rename them into an unguessable name, so that normal clients are limited to a specified set of commands.

For instance, a virtualized server provider may offer a managed Redis instance service. In this context, normal users should probably not be able to call the Redis **CONFIG** command to alter the configuration of the instance, but the systems that provide and remove instances should be able to do so.

In this case, it is possible to either rename or completely shadow commands from the command table. This feature is available as a statement that can be used inside the redis.conf configuration file. For example:

```
rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52
```

In the above example, the CONFIG command was renamed into an unguessable name. It is also possible to completely disable it (or any other command) by renaming it to the empty string, like in the following example:[8]

```
rename-command CONFIG ""
```

Attacks triggered by carefully selected inputs from external clients

There is a class of attacks that an attacker can trigger from the outside even without external access to the instance. An example of such attacks is the ability to insert data into Redis that triggers pathological (worst case) algorithm complexity on data structures implemented inside Redis internals.

For instance an attacker could supply, via a web form, a set of strings that is known to hash to the same bucket into a hash table in order to turn the O(1) expected time (the average time) to the O(N) worst case, consuming more CPU than expected, and ultimately causing a Denial of Service.

To prevent this specific attack, Redis uses a per-execution pseudo-random seed to the hash function.

Redis implements the SORT command using the qsort algorithm. Currently, the algorithm is not randomized, so it is possible to trigger a quadratic worst-case behavior by carefully selecting the right set of inputs.[8]

CATEGORY	FEATURE	DETAILS
Access control	AWS security group – allows only application with the same security group configuration to access the database	Applicable in Redis Enterprise Cloud deployment
	A Redis Enterprise cluster node can only talk with other nodes residing on the same cluster	
	Only reserved ports are available to the outside world	Per this list
Authentication	UI/API SSL authentication	
	UI/API user/password authentication	
	Database SIP authentication	
	Database SSL authentication	
	Database password authentication	
Authorization	Role based authorization for UI/API operations	
	LDAP support	Applicable in Redis Enterprise software deployment
Forensics	Admin action logging, monitoring & alerting for forensics	
Encryption	Data in transit	• Client<>Redis – SSL/TLS • Inter cluster (between cluster's nodes) – IPSec • Across-cluster – SSL/TLS
	Data at rest (storage encryption)	**Based on cloud provider capabilities. Available as a package for Redis Enterprise software**

[7]

String escaping and NoSQL injection

The Redis protocol has no concept of string escaping, so injection is impossible under normal circumstances using a normal client library. The protocol uses prefixed-length strings and is completely binary safe.
Lua scripts executed by the EVAL and EVALSHA commands follow the same rules, and thus those commands are also safe. While it would be a very strange use case, the application should avoid composing the body of the Lua script using strings obtained from untrusted sources.[6]

Code security

In a classical Redis setup, clients are allowed full access to the command set, but accessing the instance should never result in the ability to control the system where Redis is running.

Internally, Redis uses all the well-known practices for writing secure code, to prevent buffer overflows, format bugs and other memory corruption issues. However, the ability to control the server configuration using the CONFIG command makes the client able to change the working dir. of the program and the name of the dump file. This allows clients to write RDB Redis files at random paths, that is a security issue that may easily lead to the ability to compromise the system and/or run untrusted code as the same user as Redis is running. Redis does not requires root privileges to run. It is recommended to run it as an unprivileged Redis user that is only used for this purpose. The Redis authors are currently investigating the possibility of adding a new configuration parameter to prevent CONFIG SET/GET dir. and other similar run-time configuration directives. This would prevent clients from forcing the server to write Redis dump files at arbitrary locations [6].

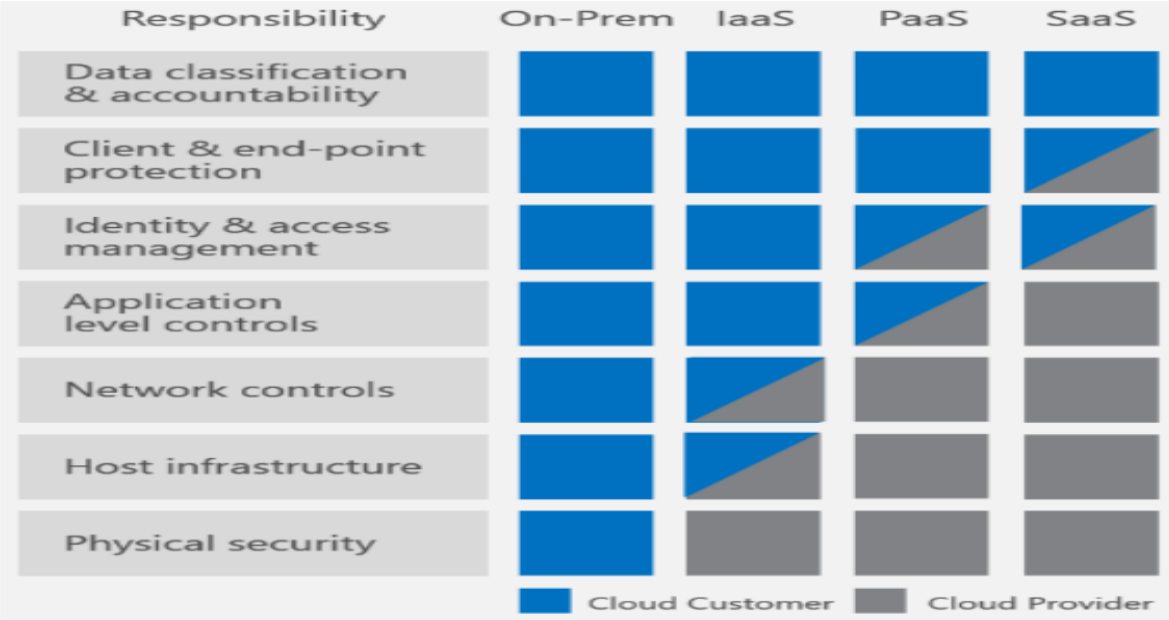
Azure Cosmos DB

Today's applications are required to be highly responsive and always online. To achieve low latency and high availability, instances of these applications need to be deployed in datacenters that are close to their users. Applications need to respond in real time to large changes in usage at peak hours, store ever increasing volumes of data, and make this data available to users in milliseconds.

Azure Cosmos DB is Microsoft's globally distributed, multi-model database service. With a click of a button, Cosmos DB enables you to elastically and independently scale throughput and storage across any number of Azure regions worldwide. You can elastically scale throughput and storage, and take advantage of fast, single-digit-millisecond data access using your favorite API including SQL, MongoDB, Cassandra, Tables, or Gremlin. Cosmos DB provides comprehensive service level agreements (SLAs) for throughput, latency, availability, and consistency guarantees, something no other database service offers.[10]

How do I secure my Cosmos DB?

Data security is a shared responsibility between you, the customer, and your database provider. Depending on the database provider you choose, the amount of responsibility you carry can vary. If you choose an on-premises solution, you need to provide everything from end-point protection to physical security of your hardware - which is no easy task. If you choose a PaaS cloud database provider such as Azure Cosmos DB, your area of concern shrinks considerably. The following image, borrowed from Microsoft's Shared Responsibilities for Cloud Computing white paper, shows how your responsibility decreases with a PaaS provider like Azure Cosmos DB.



[11]

How does Azure Cosmos DB secure my database?

Let's dig into each one in detail.

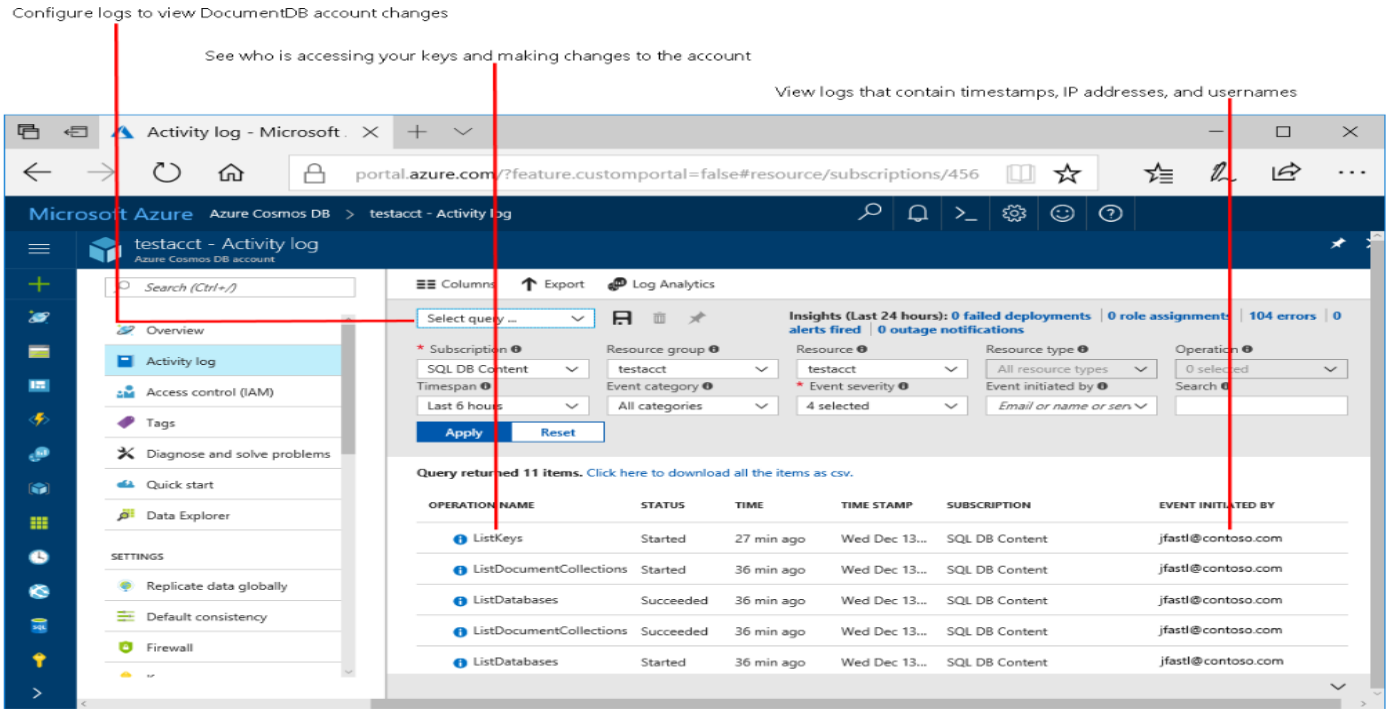
Security requirement	**Azure Cosmos DB's security approach**
Network security	Using an IP firewall is the first layer of protection to secure your database. Azure Cosmos DB supports policy driven IP-based access controls for inbound firewall support. The IP-based access controls are similar to the firewall rules used by traditional database systems, but they are expanded so that an Azure Cosmos DB database account is only accessible from an approved set of machines or cloud services. Azure Cosmos DB enables you to enable a specific IP address (168.61.48.0), an IP range (168.61.48.0/8), and combinations of IPs and ranges. All requests originating from machines outside this allowed list are blocked by Azure Cosmos DB. Requests from approved machines and cloud services then must complete the authentication process to be given access control to the resources.
Authorization	Azure Cosmos DB uses hash-based message authentication code (HMAC) for authorization. Each request is hashed using the secret account key, and the subsequent base-64 encoded hash is sent with each call to Azure Cosmos DB. To validate the request, the Azure Cosmos DB service uses the correct secret key and properties to generate a hash, then it compares the value with the one in the request. If the two values match, the operation is authorized successfully and the request is processed, otherwise

	there is an authorization failure and the request is rejected. You can use either a master key, or a resource token allowing fine-grained access to a resource such as a document.
Users and permissions	Using the master key for the account, you can create user resources and permission resources per database. A resource token is associated with a permission in a database and determines whether the user has access (read-write, read-only, or no access) to an application resource in the database. Application resources include container, documents, attachments, stored procedures, triggers, and UDFs. The resource token is then used during authentication to provide or deny access to the resource.
Active directory integration (RBAC)	You can also provide or restrict access to the Cosmos account, database, container, and offers (throughput) using Access control (IAM) in the Azure portal. IAM provides role-based access control and integrates with Active Directory. You can use built in roles or custom roles for individuals and groups.
Global replication	Azure Cosmos DB offers turnkey global distribution, which enables you to replicate your data to any one of Azure's world-wide datacenters with the click of a button. Global replication lets you scale globally and provide low-latency access to your data around the world. In the context of security, global replication ensures data protection against regional failures.
Regional failovers	If you have replicated your data in more than one data center, Azure Cosmos DB automatically rolls over your operations should a regional data center go offline. You can create a prioritized list of failover regions using the regions in which your data is replicated.
Local replication	Even within a single data center, Azure Cosmos DB automatically replicates data for high availability giving you the choice of consistency levels. This replication guarantees a 99.99% availability SLA for all single region accounts and all multi-region accounts with relaxed consistency, and 99.999% read availability on all multi-region database accounts.
Automated online backups	Azure Cosmos DB databases are backed up regularly and stored in a geo redundant store.
Restore deleted data	The automated online backups can be used to recover data you may have accidentally deleted up to ~30 days after the event.
Protect and isolate sensitive data	All data in the regions listed in What's new? is now encrypted at rest. Personal data and other confidential data can be isolated to specific container and read-write, or read-only access can be limited to specific users.

Monitor for attacks	By using audit logging and activity logs, you can monitor your account for normal and abnormal activity. You can view what operations were performed on your resources, who initiated the operation, when the operation occurred, the status of the operation, and much more as shown in the screenshot following this table.
Respond to attacks	Once you have contacted Azure support to report a potential attack, a 5-step incident response process is kicked off. The goal of the 5-step process is to restore normal service security and operations as quickly as possible after an issue is detected and an investigation is started.
HTTPS/SSL/TLS encryption	All client-to-service Azure Cosmos DB interactions are SSL/TLS 1.2 capable. Also, all intra datacentre and cross datacentre replications are SSL/TLS 1.2 enforced.

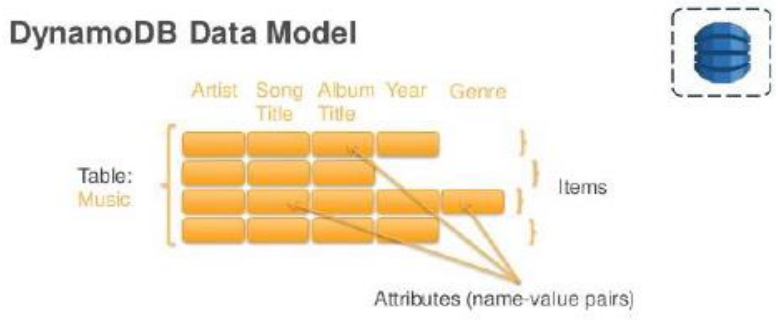
[11]

The following screenshot shows how you can use audit logging and activity logs to monitor your account: [11]



Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database, so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. Also, DynamoDB offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data. [12]



DynamoDB uses three basic data model units, Tables, Items, and Attributes. Tables are collections of Items, and Items are collections of Attributes. Attributes are basic units of information, like key-value pairs. Tables are like tables in relational databases, except that in DynamoDB, tables do not have fixed schemas associated with them. Items are like rows in an RDBMS table, except that DynamoDB requires a Primary Key. The Primary Key in DynamoDB must be unique so that it can find the exact item in the table. DynamoDB supports two kinds of Primary Keys:

- Hash Type Primary Key: If an attribute uniquely identifies an item, it can be considered as Primary. DynamoDB builds a hash index on the attribute to facilitate the uniqueness. A Hash Key is mandatory in a DynamoDB table.
- Hash and Range Type Primary Key: This type of Primary Key is built upon the hashed key and the range key in the table: a hashed index on the hash primary key attribute, and a range sort index on the range primary key attribute. This type of primary key allows for AWS's rich query capabilities. [12]

Assessing the security features of the Amazon DynamoDB database based on the assessment criteria:

1. Authentication:

Access to Amazon DynamoDB requires credentials. Those credentials must have permissions to access AWS resources, such as an Amazon DynamoDB table or an Amazon Elastic Compute Cloud (Amazon EC2) instance. We can access and authenticate Amazon DynamoDB being as IAM user. An IAM user is an identity within your AWS account that has specific custom permissions i.e. permissions to create a table in DynamoDB. [13]

2. Access Controls:

We can have valid credentials to authenticate requests, but unless we have permissions we cannot create or access Amazon DynamoDB resources. There are 2 ways permissions are provided to access the amazon dynamo DB.

- Attach a permissions policy to a user or a group in your account – To grant a user permission to create an Amazon DynamoDB resource, such as a table, you can attach a permissions policy to a user or group that the user belongs to.
- Attach a permissions policy to a role (grant cross-account permissions) – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in account A can create a role to grant cross-account permissions to another AWS account (for example, account B). [13]

3. Secure Configurations:

DynamoDB allows easier management of complex configuration object, and easier searching for configurations. There is no configuration server that needs to be connected to, we just need use the AWS SDK's to connect to DynamoDB and load your configuration table(s). It provides easier movement between AWS environments i.e.: when you run your application in a new environment it will connect to that environments DynamoDB tables and load that environments specific configurations.

4. Data Encryption:

The DynamoDB has an Encryption Client to protect the table data before we send it to DynamoDB. Encrypting our sensitive data in transit and at rest helps assure that plaintext data isn't available to any third party, including AWS.

5. Auditing:

DynamoDB is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in DynamoDB. The AWS Cloud train helps in logging include calls from the DynamoDB console,
code calls to the DynamoDB API operations and all the inserts, deletes and updates to the database are recorded. [14]

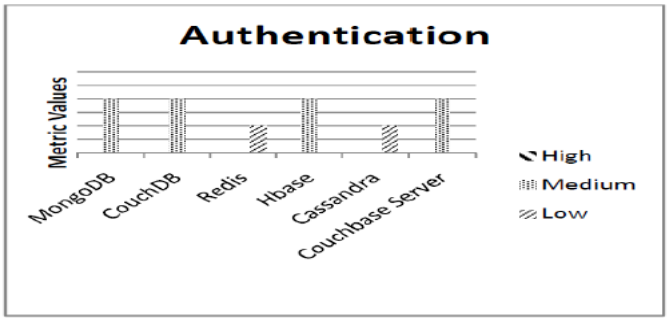
Comparison and Evaluation based on security parameters:

Now as we got to know about the overview of each of the databases and their security features let's compare the NOSQL databases together:

Authentication:

- DynamoDB's security is mostly provided by the normal AWS security measures. Access to DynamoDB is granted with IAM through access/secret key pair or roles from the machine the code is running on. Both systems are secure and allow easy access to your data.
- HBase not only supports token-based authentication for MapReduce tasks but user authentication is supported by HBase. The user authentication is done by using SASL (Simple Authentication and Security Layer) with Kerberos on per connection basis.
- Azure Cosmos DB uses hash-based message authentication code (HMAC) for authorization.
- In Redis, the goal of the authentication layer is to optionally provide a layer of redundancy. If firewalling or any other system implemented to protect Redis from external attackers fail, an external client will still not be able to access the Redis instance without knowledge of the authentication password. The AUTH command, like every other Redis command, is sent unencrypted, so it does not protect against an attacker that has enough access to the network to perform eavesdropping.

The below graph displays the **Authentication** with respect to the 2 NoSQL databases: HBase and Redis. [3]

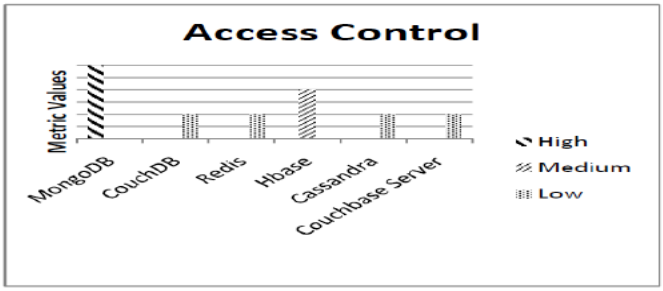


Access Control:

- HBase has few predefined roles such as user and DBA or no other support for access control at all. whenever a user requests for an access it uses the authentication mechanism which is based on Kerberos protocol to identify the user identity and then it checks in ACL for its appropriate entry in order to give the access to the data.
- Amazon DynamoDB: Dynamo DB has a user group IAM role to grant cross-account permissions but access to that group would allow access to all the users which is always a threat. [15]
- In Redis, AWS security group – allows only application with the same security group configuration to access the database.

- Using the master key for the account, you can create user resources and permission resources per database. A resource token is associated with a permission in a database and determines whether the user has access (read-write, read-only, or no access) to an application resource in the database. Application resources include container, documents, attachments, stored procedures, triggers, and UDFs. The resource token is then used during authentication to provide or deny access to the resource.

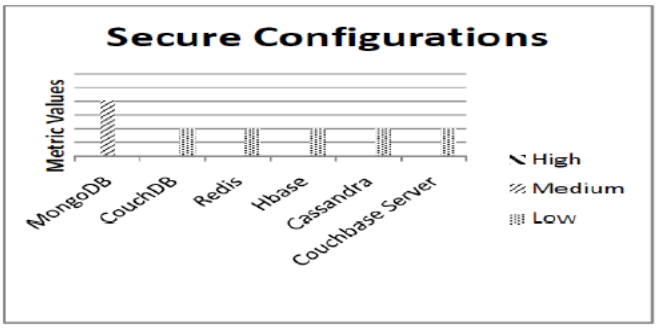
The below graph displays the **Access Control** with respect to the 2 NoSQL databases: HBase and Redis. [3]



Secure Configuration:

- HBase only supports a limited number of protocols for communication and strict access control to registries and accounts.
- Amazon DynamoDB allows easier management of complex configuration object, and easier searching for configuration provided if it has to integrate with amazon products if it has to integrate with many legacy infrastructure mongo would be easily integrated.
- A Redis Enterprise cluster node can only talk with other nodes residing on the same cluster. Only reserved ports are available to the outside world.
- Using an IP firewall is the first layer of protection to secure your database. Azure Cosmos DB supports policy driven IP-based access controls for inbound firewall support. The IP-based access controls are similar to the firewall rules used by traditional database systems, but they are expanded so that an Azure Cosmos DB database account is only accessible from an approved set of machines or cloud services. Azure Cosmos DB enables you to enable a specific IP address (168.61.48.0), an IP range (168.61.48.0/8), and combinations of IPs and ranges. All requests originating from machines outside this allowed list are blocked by Azure Cosmos DB. Requests from approved machines and cloud services then must complete the authentication process to be given access control to the resources.

The below graph displays the **Secure Configurations** with respect to the 2 NoSQL databases: HBase and Redis. [3]



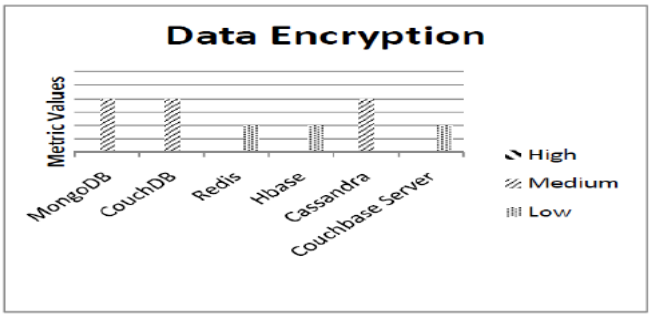
Data Encryption:

- Amazon DynamoDB has an Encryption Client which provides Encryption of data-at-transit as well as data-at-rest at application, network and database level and strong encryption with secure key management mechanisms for data security.
- The data files in HBase are stored without encryption and the database does not have an automatic data encryption.
- In Redis, when data in transit:

- Client<>Redis – SSL/TLS
- Inter cluster (between cluster's nodes) – IPSec
- Across-cluster – SSL/TLS

➢ All client-to-service Azure Cosmos DB interactions are SSL/TLS 1.2 capable. Also, all intra datacenter and cross datacenter replications are SSL/TLS 1.2 enforced. All data stored into Azure Cosmos DB is encrypted at rest.

The below graph displays the **Data Encryption** with respect to 2 NoSQL databases: HBase and Redis: [3]



Auditing:

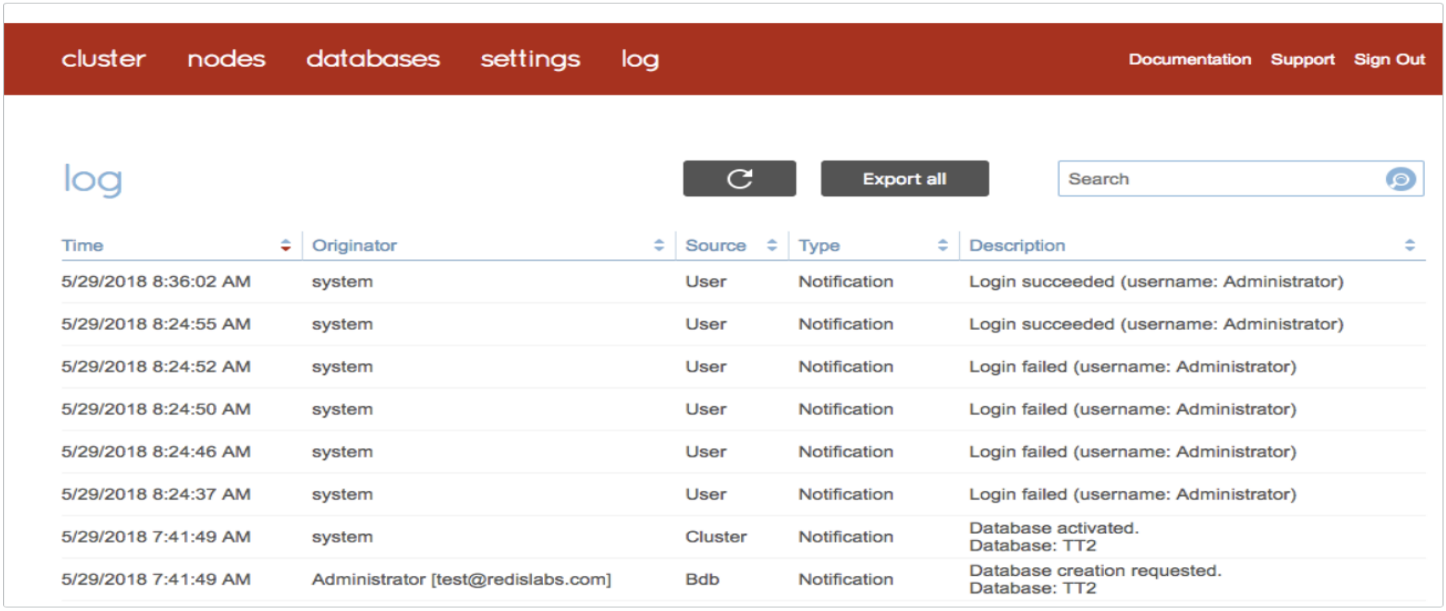
➢ Amazon DynamoDB has an auditing and logging client called the AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in DynamoDB.

➢ HBase is working over HDFS and the auditing feature is derived from HDFS.HDFS has two different audit logs*, hdfs-audit.log* for **user** activity and *SecurityAuthhdfs.audit* for service activity.

➢ Azure diagnostics logs for Azure Cosmos DB will enable users to see logs for all requests made to their respective database account at the individual request level. The diagnostics logs help track how and when your databases are accessed. This feature will also provide a convenient method for configuring the destination of the logs for the customer. Users will be able to choose the destination to either Storage Account, Event Hub or Operation Management Suite Log Analytics. [16]

➢ Management actions performed with Redis Enterprise are audited in order to fulfill two major objectives:
- To make sure that system management tasks are appropriately performed and/or monitored by the Administrator(s)
- To facilitate compliance with regulatory standards. [17]

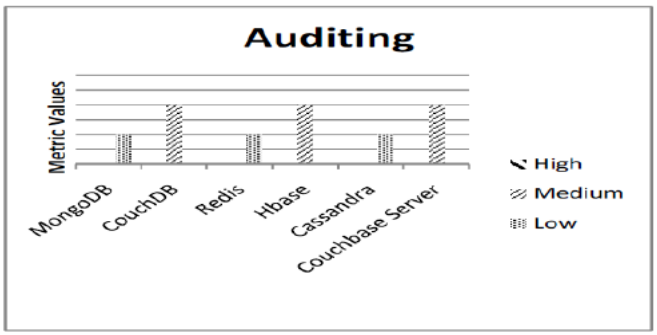
In order to fulfil both objectives, the audit records contain the following information as shown in figure:



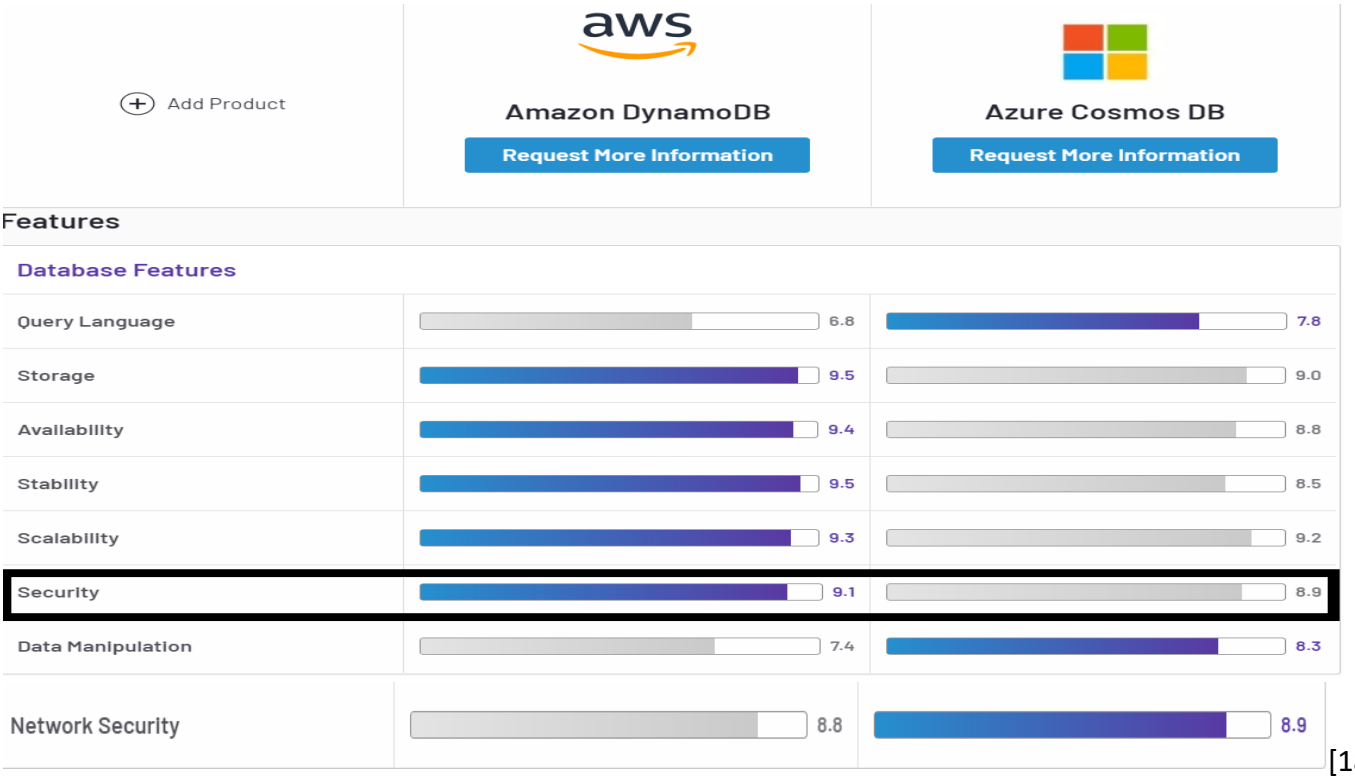
Time	Originator	Source	Type	Description
5/29/2018 8:36:02 AM	system	User	Notification	Login succeeded (username: Administrator)
5/29/2018 8:24:55 AM	system	User	Notification	Login succeeded (username: Administrator)
5/29/2018 8:24:52 AM	system	User	Notification	Login failed (username: Administrator)
5/29/2018 8:24:50 AM	system	User	Notification	Login failed (username: Administrator)
5/29/2018 8:24:46 AM	system	User	Notification	Login failed (username: Administrator)
5/29/2018 8:24:37 AM	system	User	Notification	Login failed (username: Administrator)
5/29/2018 7:41:49 AM	system	Cluster	Notification	Database activated. Database: TT2
5/29/2018 7:41:49 AM	Administrator [test@redislabs.com]	Bdb	Notification	Database creation requested. Database: TT2

[17]

The below graph displays the **Auditing** with respect to the 2 NoSQL databases: HBase and Redis. [3]

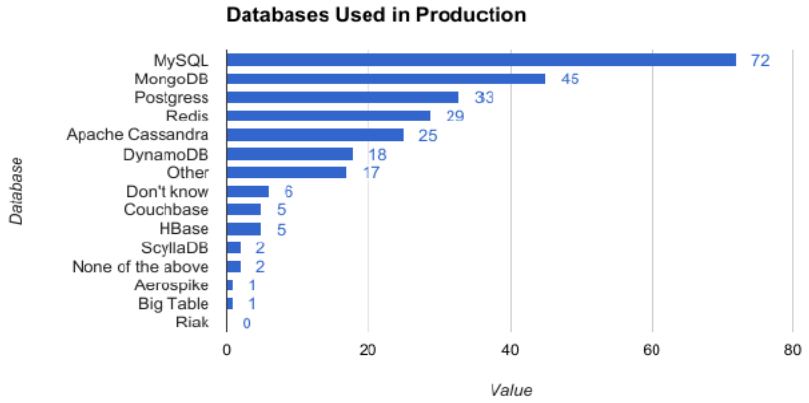


Screenshot of security rating between DynamoDB and Cosmos DB:



[18]

Though we find that Amazon's DynamoDB and Microsoft's Cosmos DB (Cloud DBS) are evolving rigorously in today's world by providing many advanced features, Redis and HBase still continues to be used predominantly in the present world. Some of the advantages of Local in-house dB's over Cloud DBS are to provide better indexing which helps in faster retrieval of data. Local DBS' help in better integration with legacy applications when compared to Cloud DBS as latter has many functions which are more compatible with those company products. Data validation is done a great extent in Local DBS when compared to the Cloud DBS. Considering all the above mention points, the below figures illustrates by concluding that local DBS are the most widely used NoSQL databases in the current industry.



Conclusion:

This paper presents assessment criteria for the evaluation of various NoSQL databases. These databases are analyzed according to a proposed assessment criterion and their criticality is identified in the existing systems. This helps in further analysis of those areas in databases which lacks in security. On researching we find that improving security is a continuous process and should not be stopped at any cost. It has also found that there is not one complete solution to all database security problems and any organization that needs to implement these databases must consider security at every level including security of database cluster itself, transmission security, security of data-at-rest and backups/replica security.

References:

[1]: http://www.ijircce.com/upload/2016/april/194_39_A%20survey.pdf
[2]:https://www.researchgate.net/publication/301633978_A_Comparison_the_Level_of_Security_on_Top_5_Open_Source_NoSQL_Databases
[3]: https://www.researchgate.net/publication/254018091_Security_Issues_in_NoSQL_Databases
[4]: https://www.semanticscholar.org/paper/Security-of-sharded-NoSQL-databases%3A-A-comparative-Zahid-Masood/5bce6add14c82dd49e7636e10b0422908d6ce594
[5]: https://data-flair.training/blogs/features-of-HBase/
[6]: https://redis.io/topics/security
[7]: https://redislabs.com/redis-enterprise/technology/redis-security-reliability/
[8]:https://www.researchgate.net/publication/269293971_Security_of_sharded_NoSQL_databases_A_comparative_analysis
[9]: http://www.tarsnap.com/spiped.html
[10]: https://docs.microsoft.com/en-us/azure/cosmos-db/introduction
[11]: https://docs.microsoft.com/en-us/azure/cosmos-db/database-security
[12]: https://www.allthingsdistributed.com/2012/01/amazon-dynamodb.html
[13]: https://read.acloud.guru/why-amazon-dynamodb-isnt-for-everyone-and-how-to-decide-when-it-s-for-you- aefc52ea9476
[14]: https://aws.amazon.com/blogs/database/how-to-automate-the-audit-of-operational-best-practices-for-your- aws-account/
[15]: https://www.kdnuggets.com/2018/08/dynamodb-vs-cassandra.html
[16]: https://azure.microsoft.com/en-us/updates/azure-cosmos-db-database-account-auditing/
[17]: https://blogs.apache.org/hbase/entry/hbase_cell_security
[18]: https://www.g2.com/compare/aws-amazon-dynamodb-vs-azure-cosmos-db