**CS595 – Draft Paper**

* **Paper Topic:** Research Security for NoSQL Database and cloud NoSQL databases (Security between Cassandra, HBASE, DynamoDB and Cosmo DB)
* **Paper subject area:** Compare and evaluate security between Cassandra, HBASE, DynamoDB and Cosmo DB

**Abstract**

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.

NoSQL systems store and manage data in ways that allow for high operational speed and great flexibility on the part of the developers. It can be scaled horizontally across hundreds or thousands of servers. NoSQL encloses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications.

Currently, NoSQL databases are in the evolutionary stage of their lifecycle and the attack vectors for NoSQL databases aren’t well mapped out. It is likely that new attack vectors may emerge in future that will target NoSQL data stores in new ways. NoSQL has not been designed with security as a priority, so developers or security teams need to add a security layer to their organization’s [NoSQL applications](https://www.computerweekly.com/feature/Big-data-applications-bring-new-database-choices-challenges).

Few of the application that is being widely used as NoSQL database are Cassandra and HBase.

**Cassandra**: It offers capabilities that NoSQL databases simply cannot match such as continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones. Cassandra provides security in the form of: 1. authentication and authorization, and 2. inter-node + client-to-node encryption.

**HBase:** It is an open source NoSQL database that handle huge data sets with billions of rows and millions of columns, and it easily combines data sources that use a wide variety of different structures and schemas. HBase provides the much-needed means for secure communication with other technologies it relies upon.

Like all NoSQL databases, HBase and Cassandra have their security issues (the main one being that securing data spoils performance making the system heavy and inflexible). But it is safe to say that both databases have some features to ensure data security: authentication and authorization in both and inter-node + client-to-node encryption in Cassandra. HBase, in its turn, provides the much-needed means for secure communication with other technologies it relies upon.

Both Cassandra and HBase provide not just database-wide access control but allow a certain level of granularity. Cassandra enables row-level access and HBase goes as deep as cell-level. Cassandra defines user roles and sets conditions for these roles which later determine whether a user can see particular data or not. While HBase has an inverse ‘move.’ Its administrators assign a visibility label to data sets and then ‘tell’ users and user groups what labels they can see.

Few of the application that is being widely used as cloud NoSQL database are Amazon DynamoDB and Azure Cosmos DB.

**Amazon DynamoDB**: It is a nonrelational database that delivers reliable performance at any scale. It's a fully managed, multi-region, multi-master database that provides consistent single-digit millisecond latency, and offers built-in security, backup and restore, and in-memory caching.

DynamoDB is a key-value store. It works really well if you are retrieving individual records based on key lookups. Complex queries or scans require careful indexing and are tricky or straight-up inadvisable to write — even if you don’t have a terribly large amount of data, and even if you have some familiarity with NoSQL design principles.

Access to Amazon DynamoDB requires credentials. Those credentials must have permissions to access AWS resources, such as an Amazon DynamoDB table. For using AWS Identity and Access Management (IAM) and DynamoDB to help secure access to your resources, DynamoDB provides security in the form of:

1. Authentication, and

2. Access Control

1.Authentication: You can access AWS as any of the following types of identities:

AWS account root user – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. It is recommended to not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM user – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create a table in DynamoDB). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don’t use AWS tools, you must sign the request yourself. DynamoDB supports Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the AWS General Reference.

IAM role – An IAM role is an IAM identity that you can create in your account that has specific permissions. It is similar to an IAM user, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

Federated user access – Instead of creating an IAM user, you can use existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.

AWS service access – You can use an IAM role in your account to grant an AWS service permission to access your account’s resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster.

Applications running on Amazon EC2 – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials.

2.Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon DynamoDB resources. For example, you must have permissions to create an Amazon DynamoDB table.

**Azure Cosmos DB**: It offers multiple NoSQL choices including key-value, graph, column-family, and document data in one service. Azure Cosmos DB is Microsoft's globally distributed, multi-model database. With the click of a button, Azure Cosmos DB enables you to elastically and independently scale throughput and storage across any number of Azure's geographic regions. It offers throughput, latency, availability, and consistency guarantees with comprehensive service level agreements (SLAs), something no other database service can offer.

It 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 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.

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.

You can also provide access to the database account using Access control (IAM) in the Azure portal. IAM provides role-based access control and integrates with Active Directory.

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.

* **Reference resources:**
  + <https://data-flair.training/blogs/hbase-vs-cassandra/>
  + <https://db-engines.com/en/system/Cassandra%3BHBase>
  + <https://www.ibm.com/analytics/hadoop/hbase>
  + https://docs.microsoft.com/en-us/azure/cosmos-db/introduction