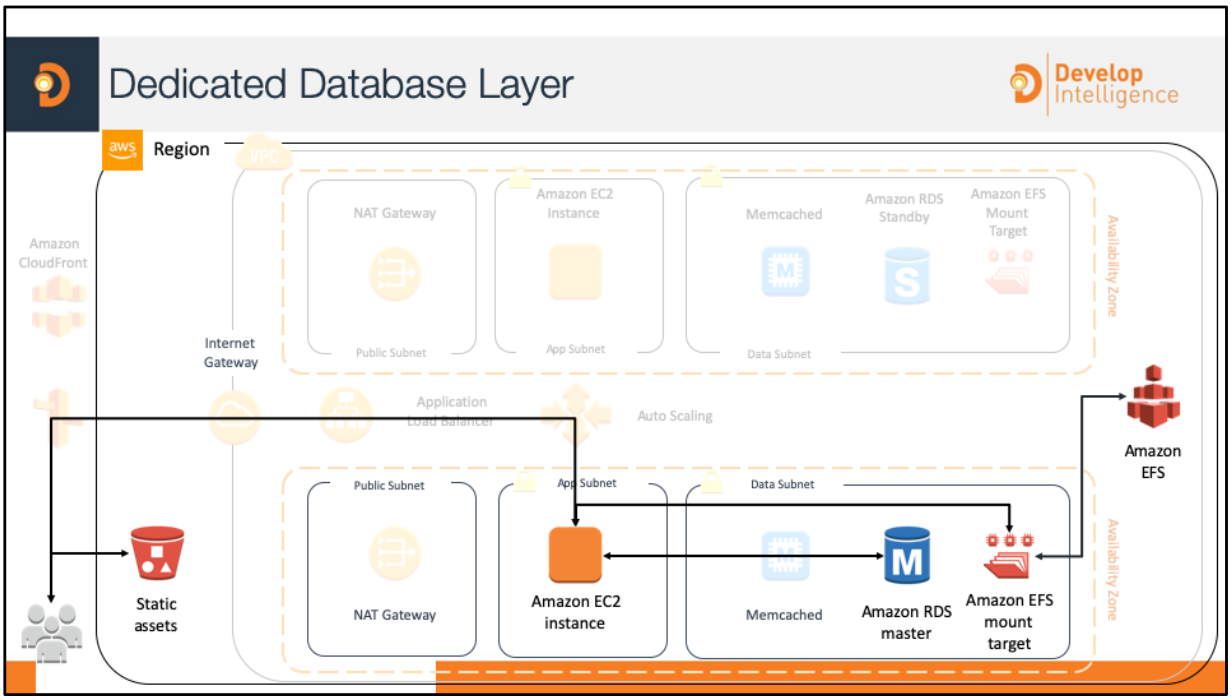
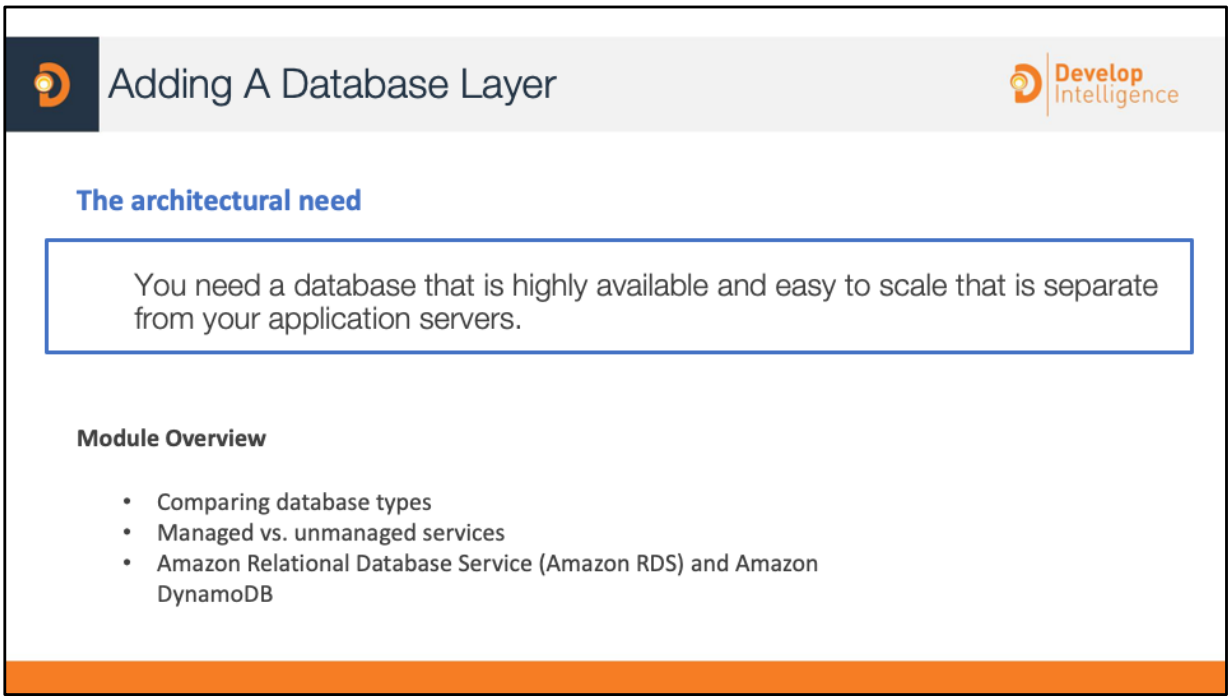
Adding A Database Layer

Develop Intelligence

- Module 4



By the end of class, you will be able to understand all of the components of this architectural diagram. You will also be able to construct your own architectural solutions that are just as large and robust.


Adding A Database Layer
Develop Intelligence
The architectural need
You need a database that is highly available and easy to scale that is separate from your application servers.
Module Overview
• Comparing database types
• Managed vs. unmanaged services
• Amazon Relational Database Service (Amazon RDS) and Amazon DynamoDB

What Should You Consider?

Develop Intelligence

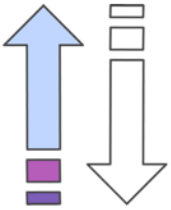
Scalability

Total storage requirements

Object size and type

Durability

How much throughput do we need?
Will the solution we choose be able to scale up later if needed?



What Should You Consider?

Develop Intelligence

- Scalability
- **Total storage requirements**
- Object size and type
- Durability

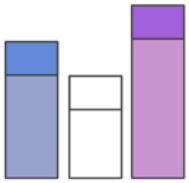
How large does our database need to be?
Will we have GB, TB, or PB of data?

What Should You Consider?

Develop Intelligence

- Scalability
- Total storage requirements
- **Object size and type**
- Durability

Do we need to store simple data structures, large data objects, or both?

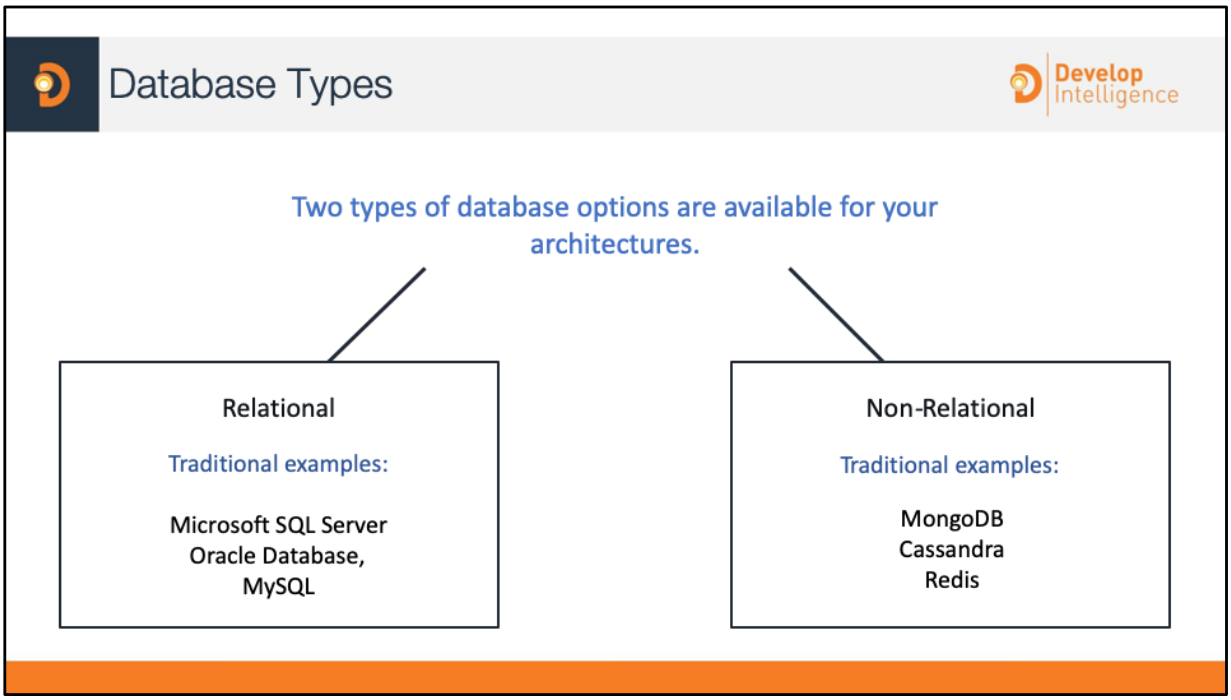


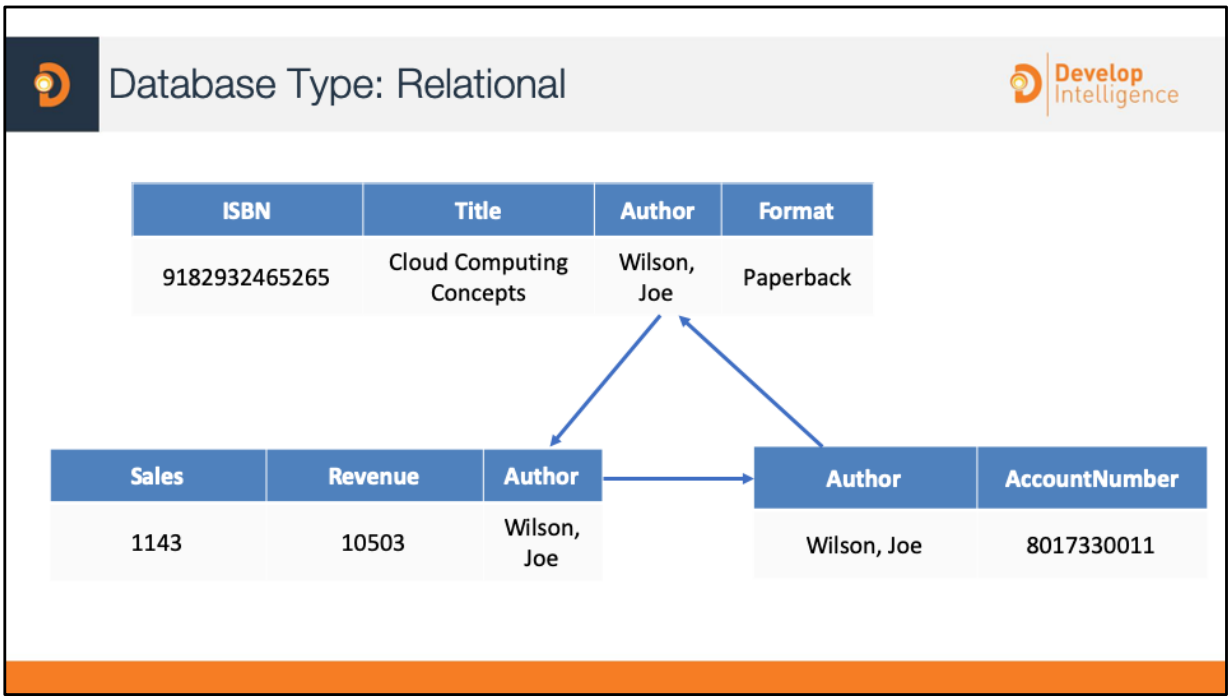
What Should You Consider?

Develop Intelligence

- Scalability
- Total storage requirements
- Object size and type
- **Durability**

What level of data durability, data availability, and recoverability do you require?
Do you have a related regulatory obligation?


Database Types
Develop Intelligence
Two types of database options are available for your architectures.
Relational
Traditional examples:
Microsoft SQL Server
Oracle Database,
MySQL
Non-Relational
Traditional examples:
MongoDB
Cassandra
Redis



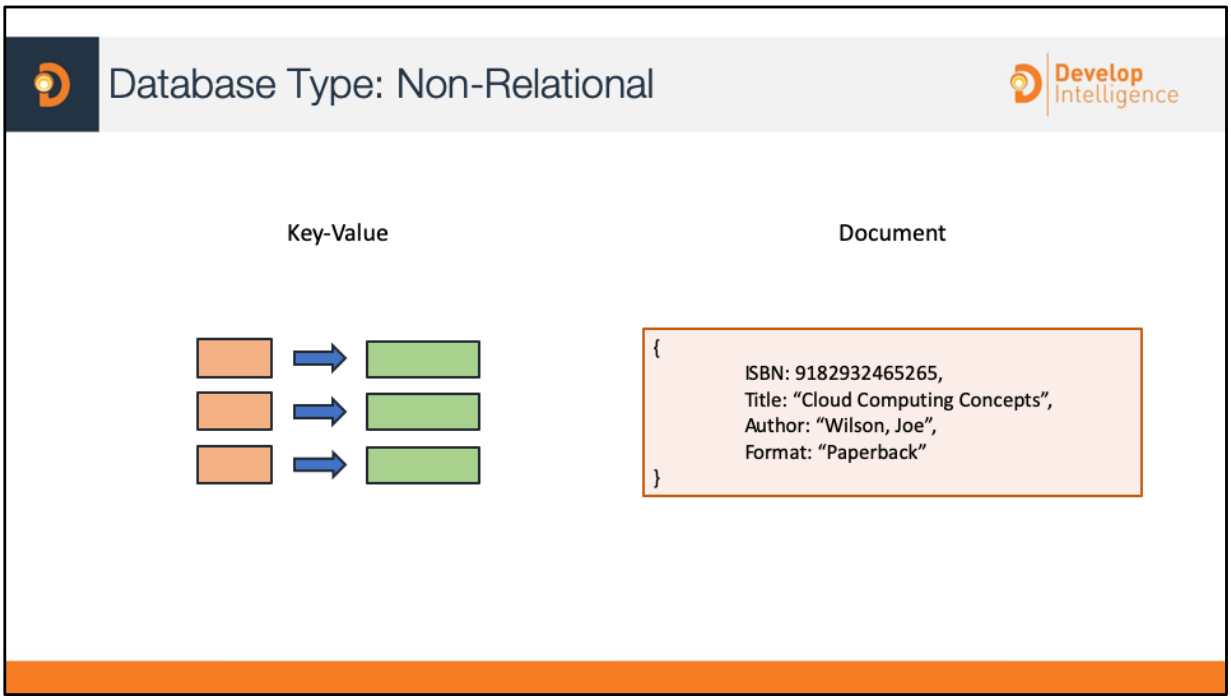
A SQL database stores data in rows and columns. *Rows* contain all the information about one entry, and *columns* are the attributes that separate the data points. A SQL database schema is fixed: columns must be locked before data entry. Schemas can be amended if the database is altered entirely and taken offline. Data in SQL databases is queried using SQL (Structure Query Language), which can allow for complex queries. SQL databases scale vertically by increasing hardware power.

Database Type: Relational

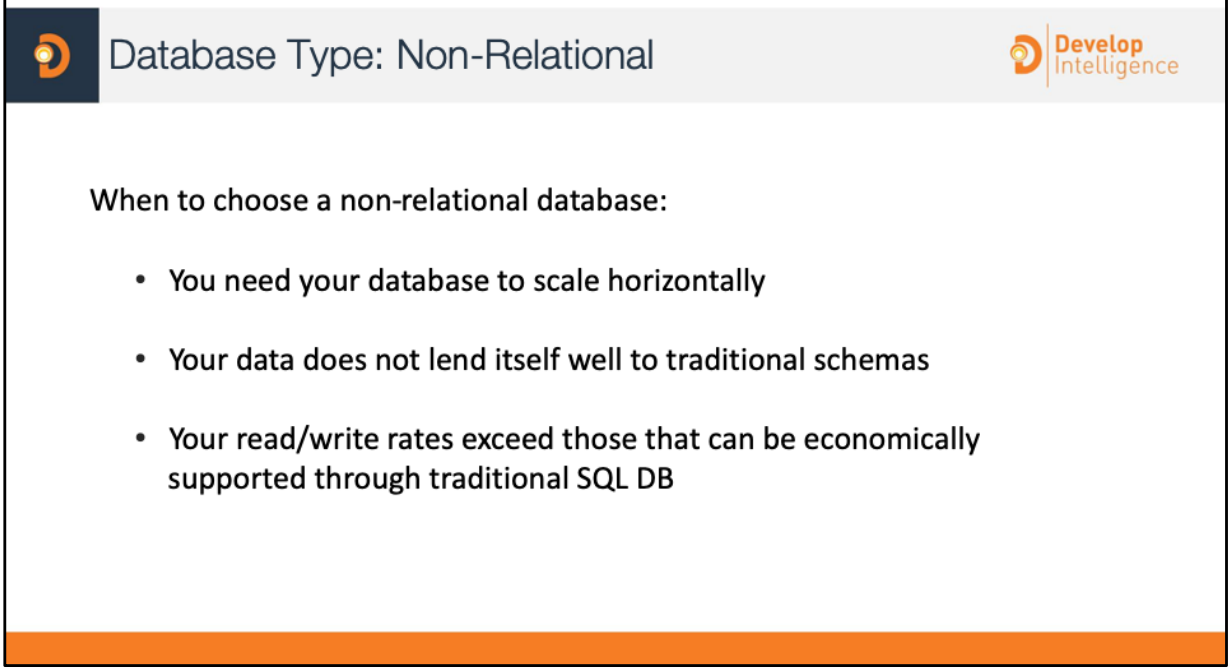
Develop Intelligence

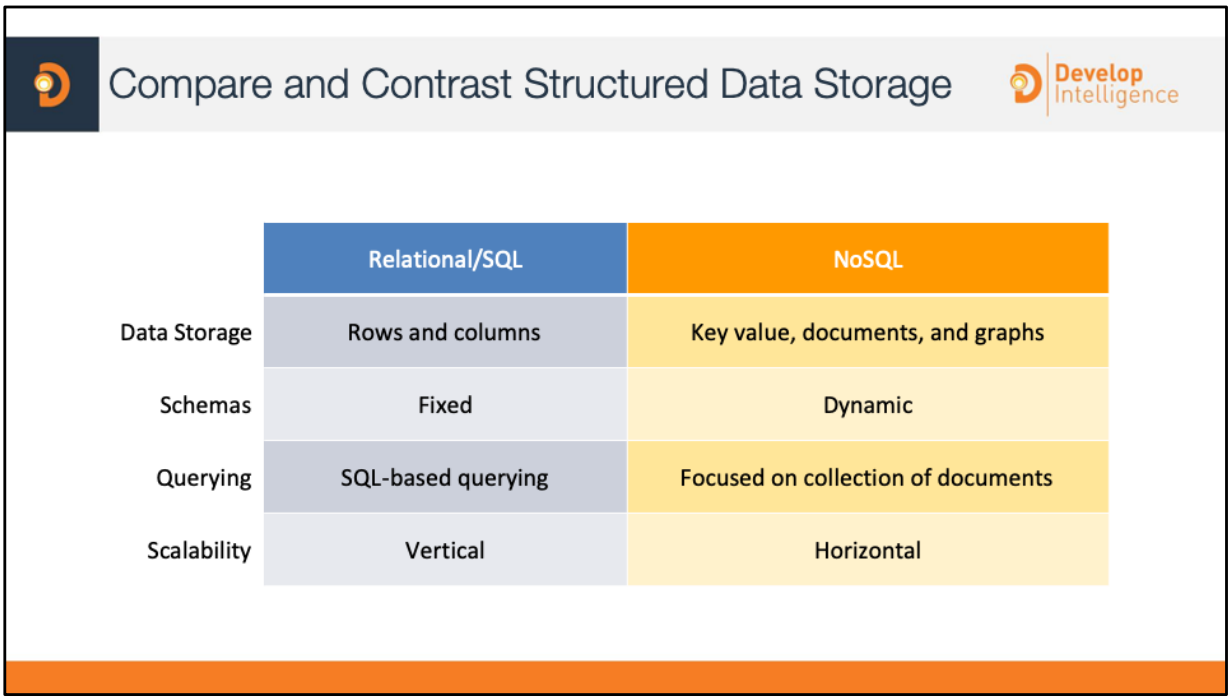
When to choose a relational database:

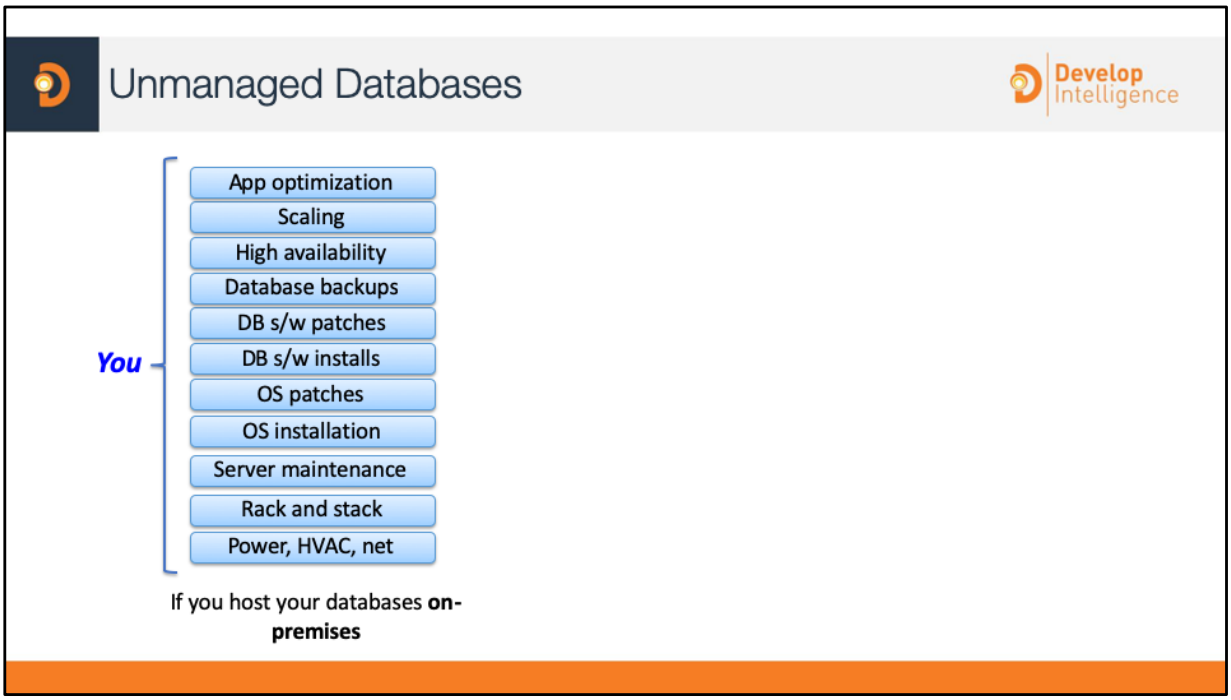
- You require strict schema rules and data quality enforcement
- Your database doesn't need extreme read/write capacity
- If you have a relational dataset that does not require extreme performance, an RDBMS can be the best, lowest effort solution.



A NoSQL database stores data using one of many storage models, including key-value pairs, documents, and graphs. NoSQL schemas are dynamic. A row doesn't have to contain data for each column. Data in NoSQL databases is queried by focusing on collections of documents. NoSQL databases scale horizontally by increasing servers.


Database Type: Non-Relational
Develop Intelligence
When to choose a non-relational database:
• You need your database to scale horizontally
• Your data does not lend itself well to traditional schemas
• Your read/write rates exceed those that can be economically supported through traditional SQL DB


Compare and Contrast Structured Data Storage
Develop Intelligence
Relational/SQL
NoSQL
Data Storage
Rows and columns
Key value, documents, and graphs
Schemas
Fixed
Dynamic
Querying
SQL-based querying
Focused on collection of documents
Scalability
Vertical
Horizontal

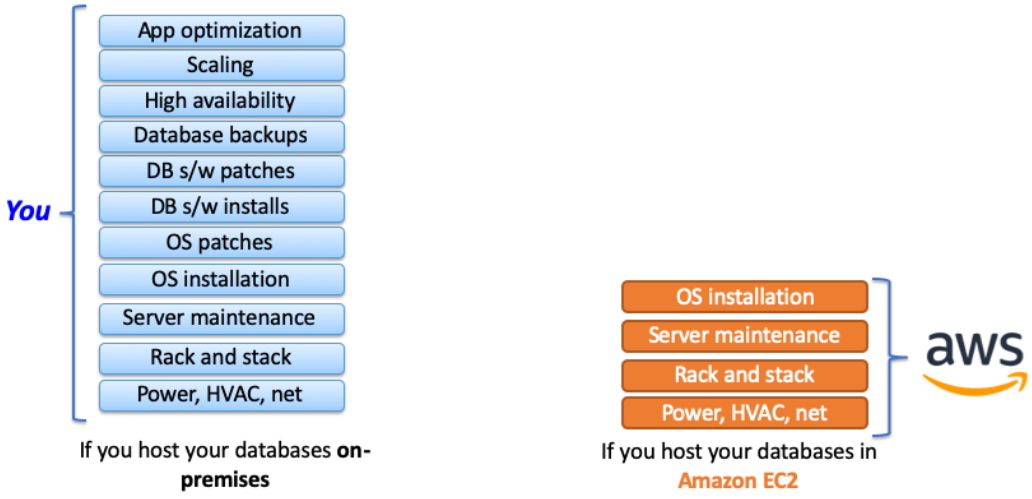


In general: you are responsible for all security backups, DB tuning and replication.

You might be required by compliance obligations to create a self-managed DB solution for your application.

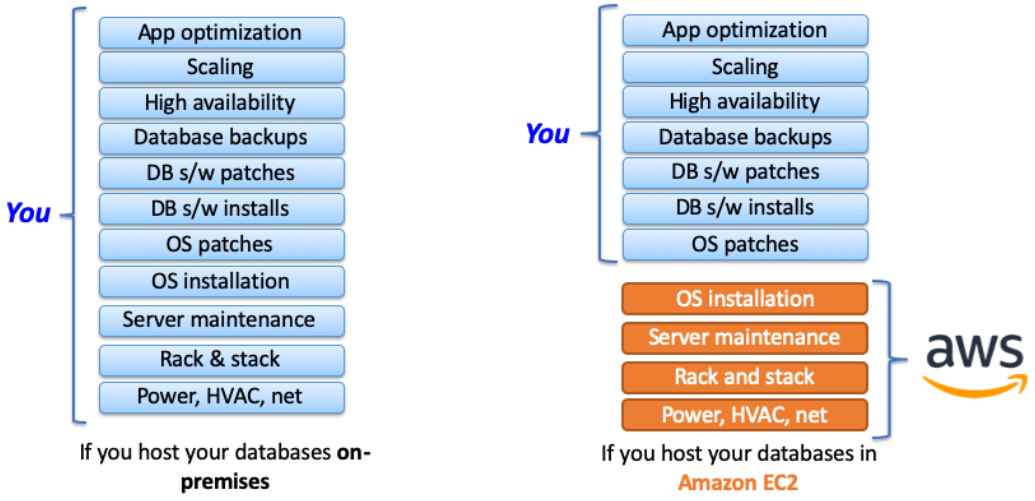
Unmanaged Databases

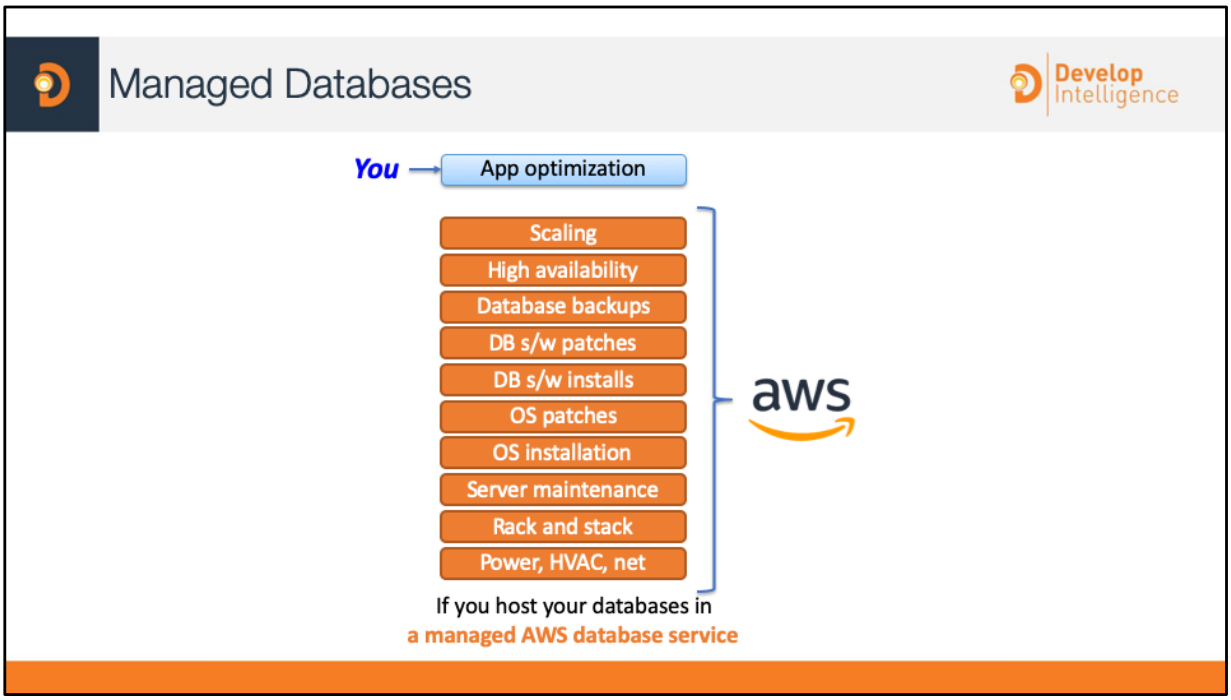
Develop Intelligence



Unmanaged Databases

Develop Intelligence



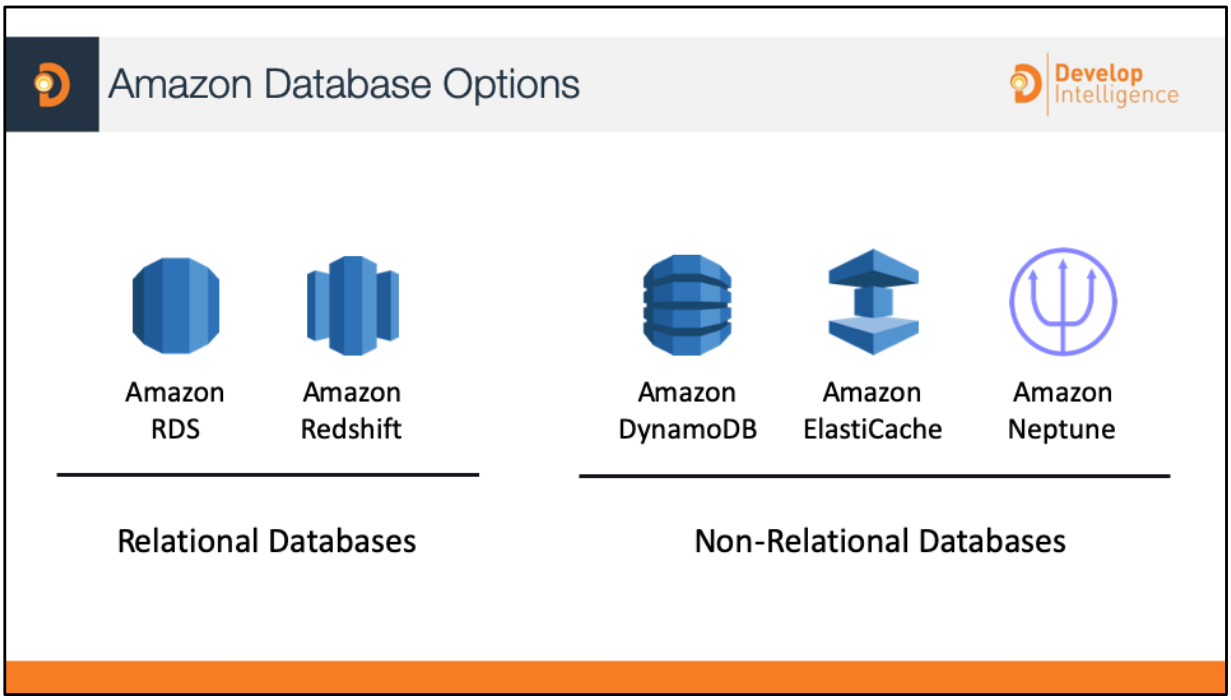


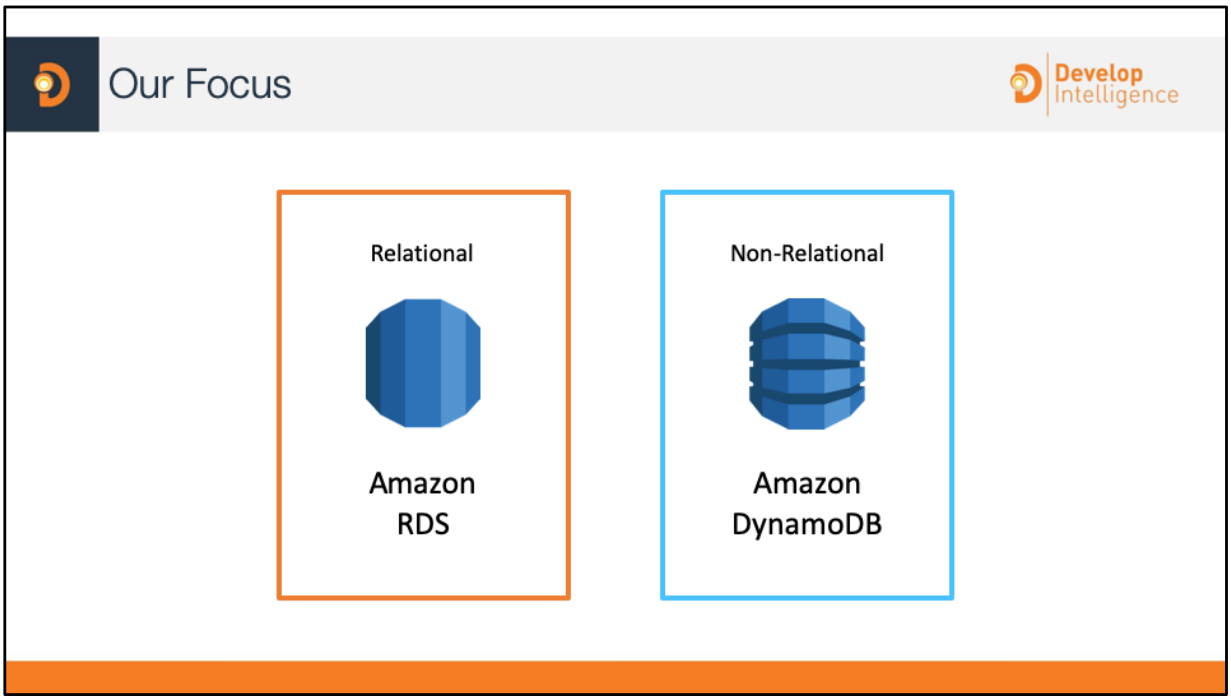
AWS-Managed Databases

These provide systems for high availability scalability and backups. Packages that you can opt into. Choose to use
scaling, high availability, database backups, database software patches, database software installs, OS patches

Handing over the repetitive heavy lifting
OS installation, server maintenance, rack and stack, Power,hvac,net

In general: You are responsible only for app optimization “making sure the database layer works as well as possible with your application”


Amazon Database Options
Develop Intelligence
Amazon RDS
Amazon Redshift
Amazon DynamoDB
Amazon ElastiCache
Amazon Neptune
Relational Databases
Non-Relational Databases


Our Focus
Develop Intelligence
Relational
Amazon RDS
Non-Relational
Amazon DynamoDB

Amazon RDS

Develop Intelligence

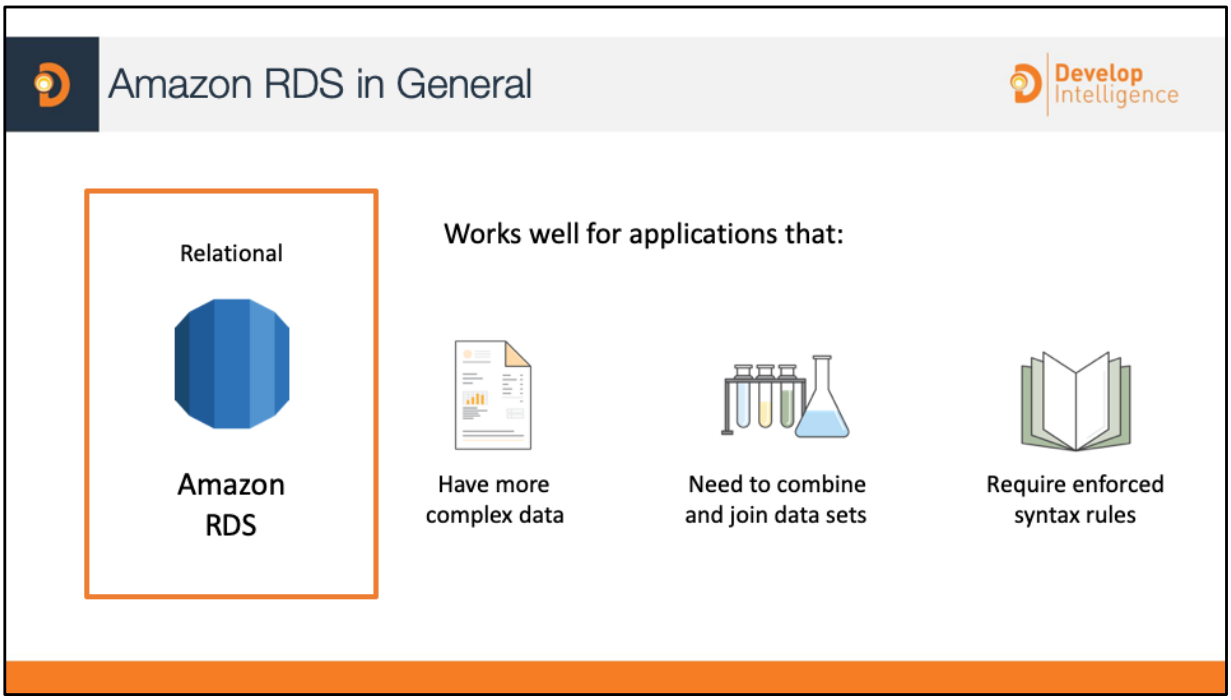
Relational

Amazon RDS

Fully managed relational database service

Provisions new instances in a few minutes

Scaling vertically with a few mouse clicks



To protect against accidental deletion of data, Amazon Relational Database Service (RDS) supports the retention of automated backups after database deletion. In situations where accidental data loss occurs, your Amazon RDS instance can be restored to any point in time defined by the retained backups. Any retained automatic backups will adhere to the lifecycle policy specified for the deleted database. The automated backups will be deleted after the specified retention period passes, removing the need for manually clearing any stale backups. This is available for MySQL, MariaDB, PostgreSQL, Oracle, and Microsoft SQL Server database engines.

Amazon RDS Storage is Fully Managed

Develop Intelligence

Relational

Amazon RDS

MySQL, MariaDB, PostgreSQL, Oracle – 16TB max

Aurora for MySQL and PostgreSQL – 64TB max

Amazon RDS and Amazon Aurora

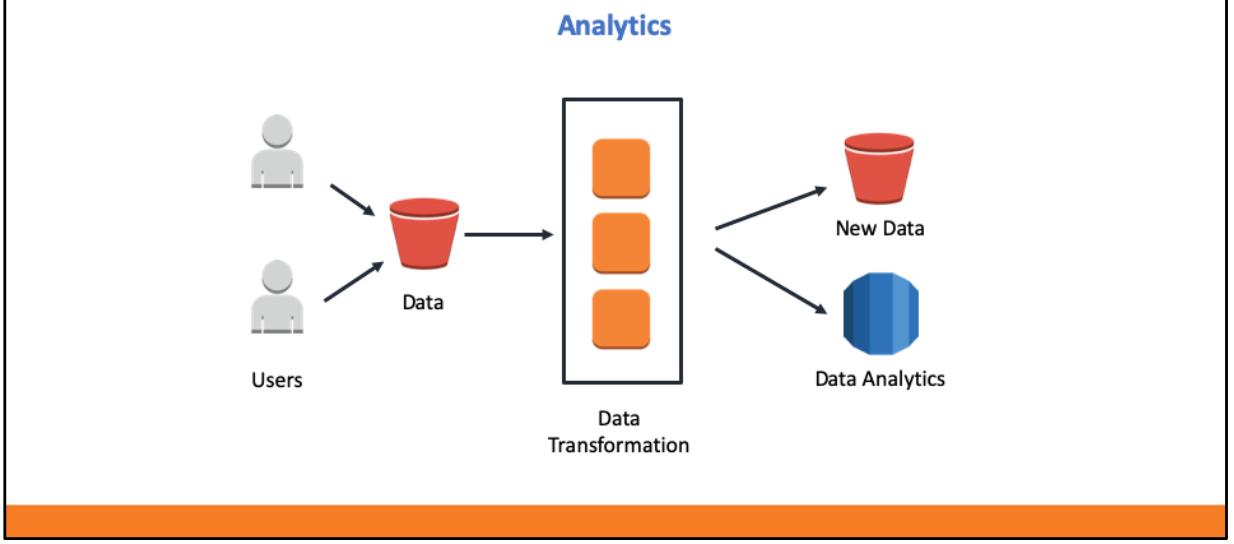
Develop Intelligence

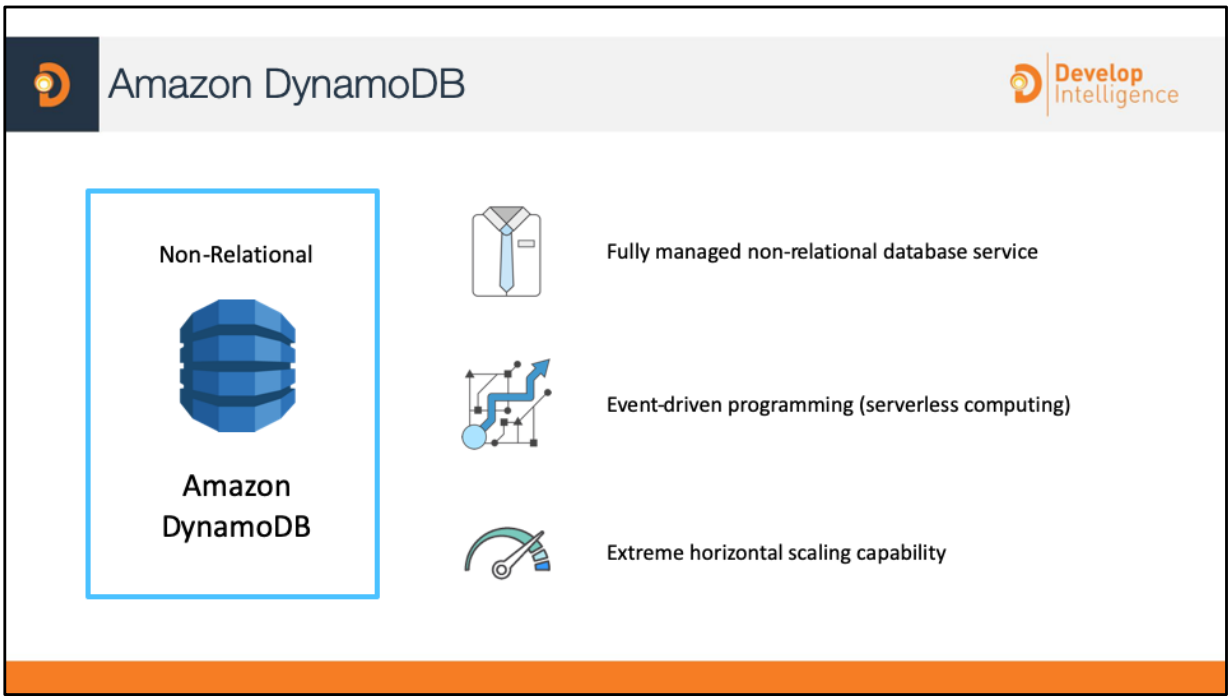
Amazon Aurora is a fully managed, MySQL- and PostgreSQL-compatible, relational database engine.

- Up to five times the throughput of MySQL
- Up to three times the throughput of PostgreSQL
- Replicates data six ways across three Availability Zones
- Requires very little change to your existing application

Amazon RDS Use Case

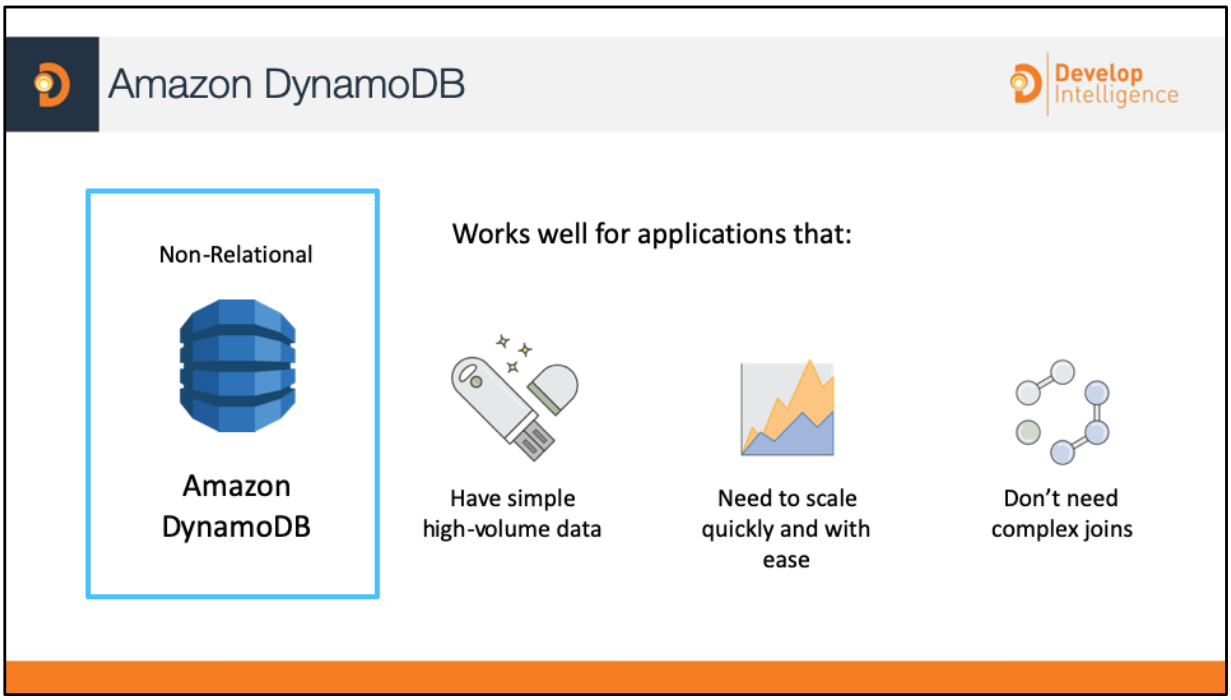
Develop Intelligence



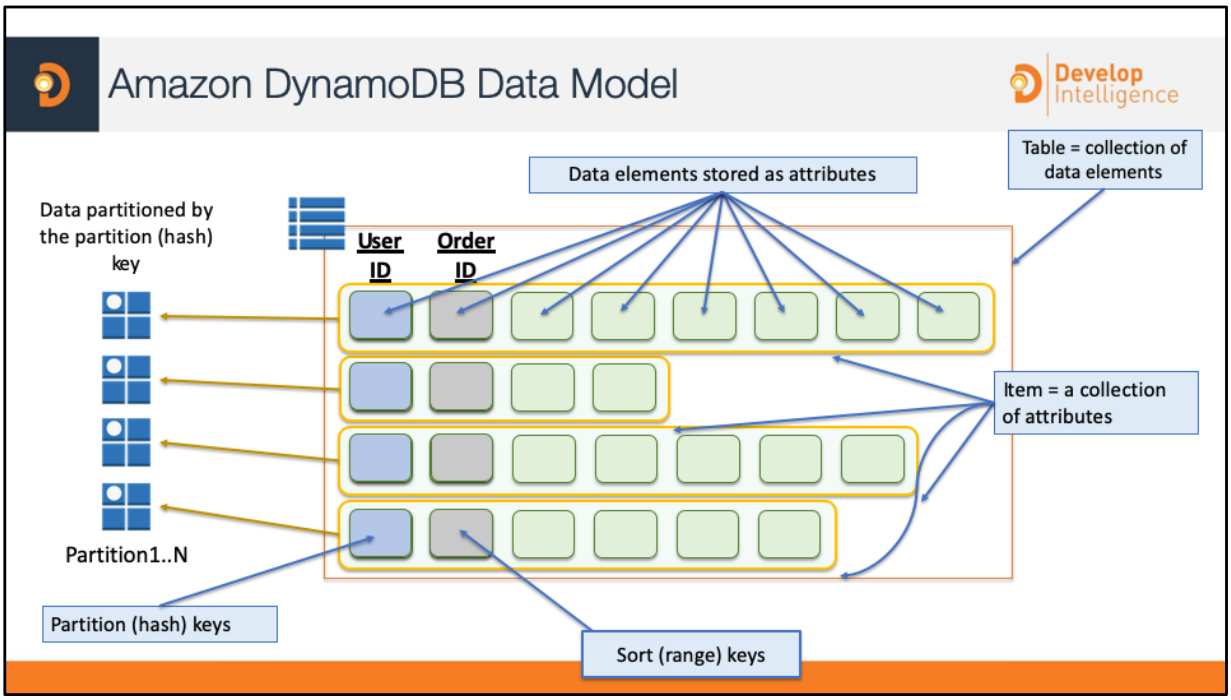


Amazon DynamoDB is a fast, NoSQL database service that makes it simple and cost-effective to store data. Its throughput and single-digit millisecond latency make it a great fit for gaming, ad tech, mobile, and many other applications. DynamoDB delivers seamless throughput and storage scaling via API and an easy-to-use management console, so you can easily scale up or down to meet your needs. Many AWS customers have, with the click of a button, created DynamoDB deployments in a matter of minutes that are able to serve trillions of database requests per year.

DynamoDB tables do not have fixed schemas, and each item may have a different number of attributes. You can add secondary indexes to increase flexibility in the queries you can perform, without affecting performance. Performance, reliability, and security are built in, with SSD (solid state drive) storage and automatic three-way replication. DynamoDB uses proven cryptographic methods to securely authenticate users and prevent unauthorized data access.



DynamoDB transactions provides ACID across one or more tables within a **single AWS account and region.** Can be used with applications that require coordinated inserts, deletes, or updates to multiple items.
https://aws.amazon.com/blogs/aws/new-amazon-dynamodb-transactions/



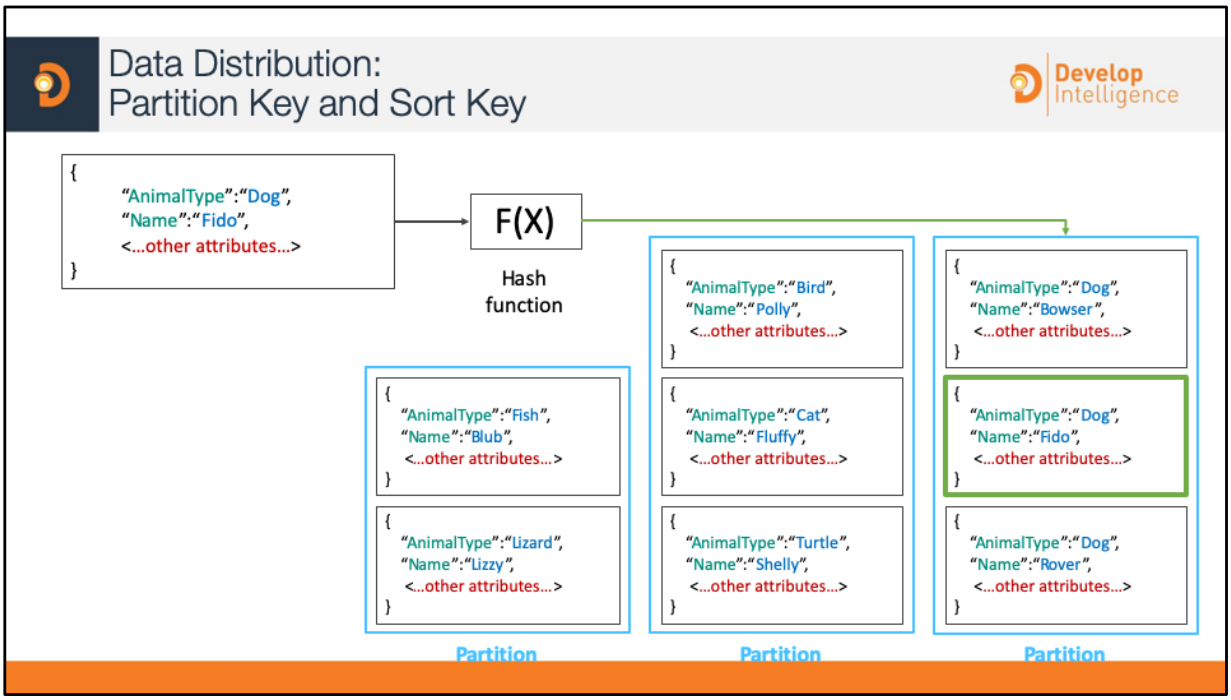
In the world of DynamoDB, there is no schema. In a table, you have items. An item has *variable attributes* that have key-value pairs (similar to associative array or hash). In DynamoDB, they are called *key* and *attribute*.

DynamoDB supports key-value GET/PUT operations using a user-defined primary key (partition key, also known as a hash key). The primary key is the only required attribute for items in a table and it uniquely identifies each item. You specify the primary key when you create a table. In addition, DynamoDB provides flexible querying by allowing queries on non-primary key attributes using Global Secondary Indexes and Local Secondary Indexes.

A primary key can be a single-attribute partition key or a composite partition-sort key. A composite partition-sort key is indexed as a partition key element and sort (also known as range) key element. This multi-part key could be a combination of “UserID” (partition) and “Timestamp” (sort). Holding the partition key element constant, you can search across the sort key element to retrieve items. For example, you can retrieve all items for a single UserID across a range of timestamps.

Auto-partitioning occurs with data set size growth and as provisioned capacity

increases.



If the table has a composite primary key (partition key and sort key), DynamoDB calculates the hash value of the partition key in the same way as described in Data Distribution: Partition Key—but it stores all of the items with the same partition key value physically close together, ordered by sort key value.

To write an item to the table, DynamoDB calculates the hash value of the partition key to determine which partition should contain the item. In that partition, there could be several items with the same partition key value, so DynamoDB stores the item among the others with the same partition key, in ascending order by sort key.

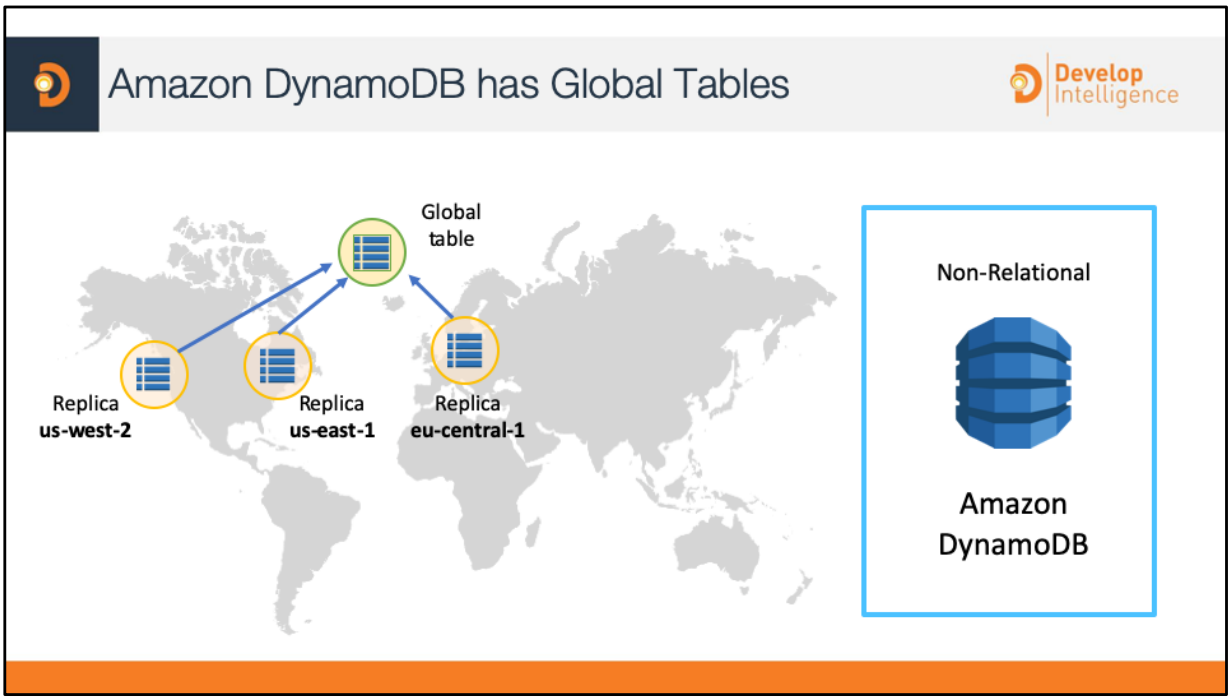
To read an item from the table, you must specify its partition key value and sort key value. DynamoDB calculates the partition key's hash value, yielding the partition in which the item can be found.

You can read multiple items from the table in a single operation (Query), provided that the items you want have the same partition key value. DynamoDB returns all of the items with that partition key value. Optionally, you can apply a condition to the sort key so that it returns only the items within a certain range of values.

Suppose that the *Pets* table has a composite primary key consisting of *AnimalType* (partition key) and *Name* (sort key). The following diagram shows DynamoDB writing an item with a partition key value of *Dog* and a sort key value of *Fido*.

To read that same item from the *Pets* table, DynamoDB calculates the hash value of *Dog*, yielding the partition in which these items are stored. DynamoDB then scans the sort key attribute values until it finds Fido.

To read all of the items with an *AnimalType* of *Dog*, you can issue a Query operation without specifying a sort key condition. By default, the items are returned in the order that they are stored (that is, in ascending order by sort key). Optionally, you can request descending order instead.



A *global table* is a collection of one or more DynamoDB tables, all owned by a single AWS account, identified as replica tables. A *replica table* (or *replica*, for short) is a single DynamoDB table that functions as a part of a global table. Each replica stores the same set of data items. Any given global table can only have one replica table per region, and every replica has the same table name and the same primary key schema.

Amazon DynamoDB global tables provide a fully managed solution for deploying a multi-region, multi-master database, without having to build and maintain your own replication solution. When you create a global table, you specify the AWS regions where you want the table to be available. DynamoDB performs all of the necessary tasks to create identical tables in these regions, and propagate ongoing data changes to all of them.

Amazon DynamoDB Use Cases

Develop Intelligence

Leaderboards and Scoring

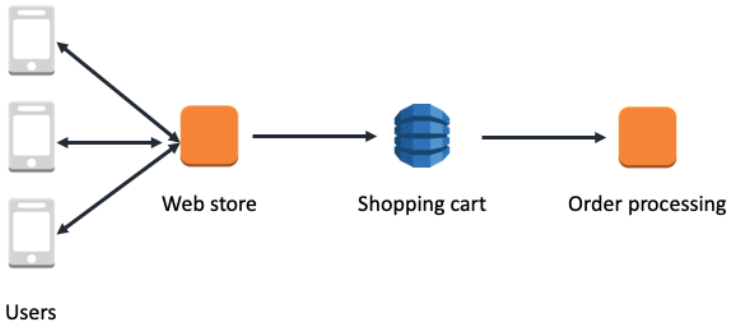
GameScores

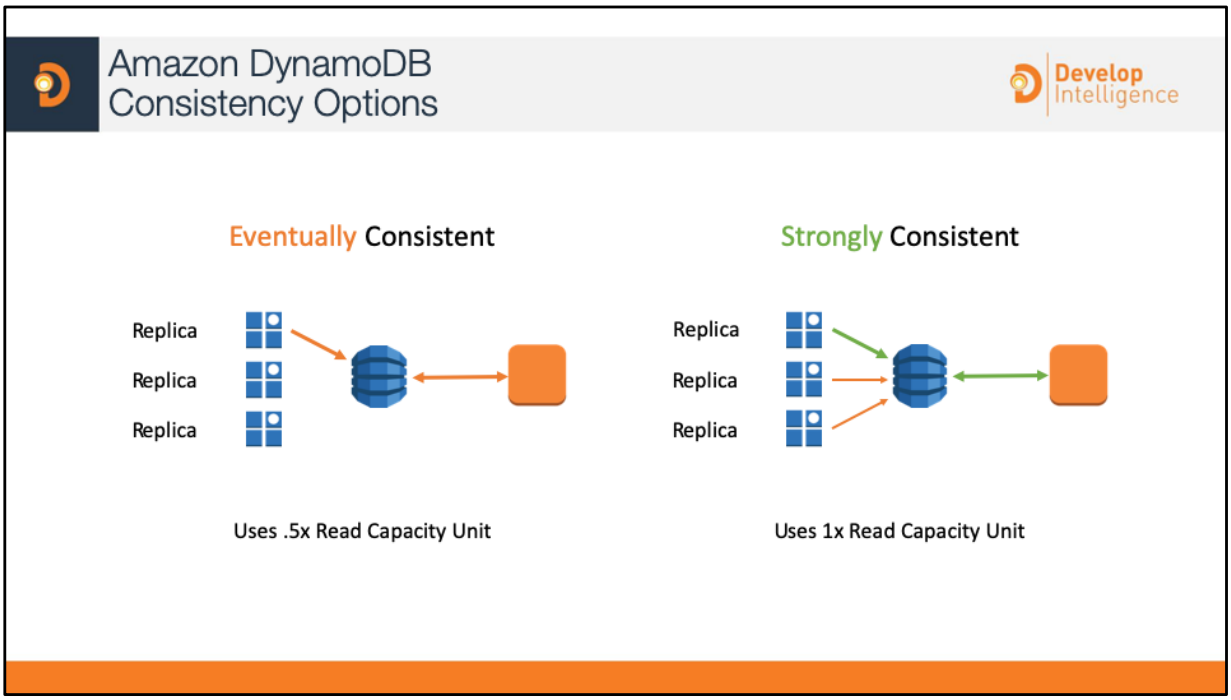
UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	...
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	...
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	...
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	...
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	...
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	...
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	...
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	...
"103"	"Starship X"	42	"2015-07-11:06:53:00"	4	19	...
...	...	...	...	...	...	

Amazon DynamoDB Use Cases

Develop Intelligence

Temporary Data (Online Cart)



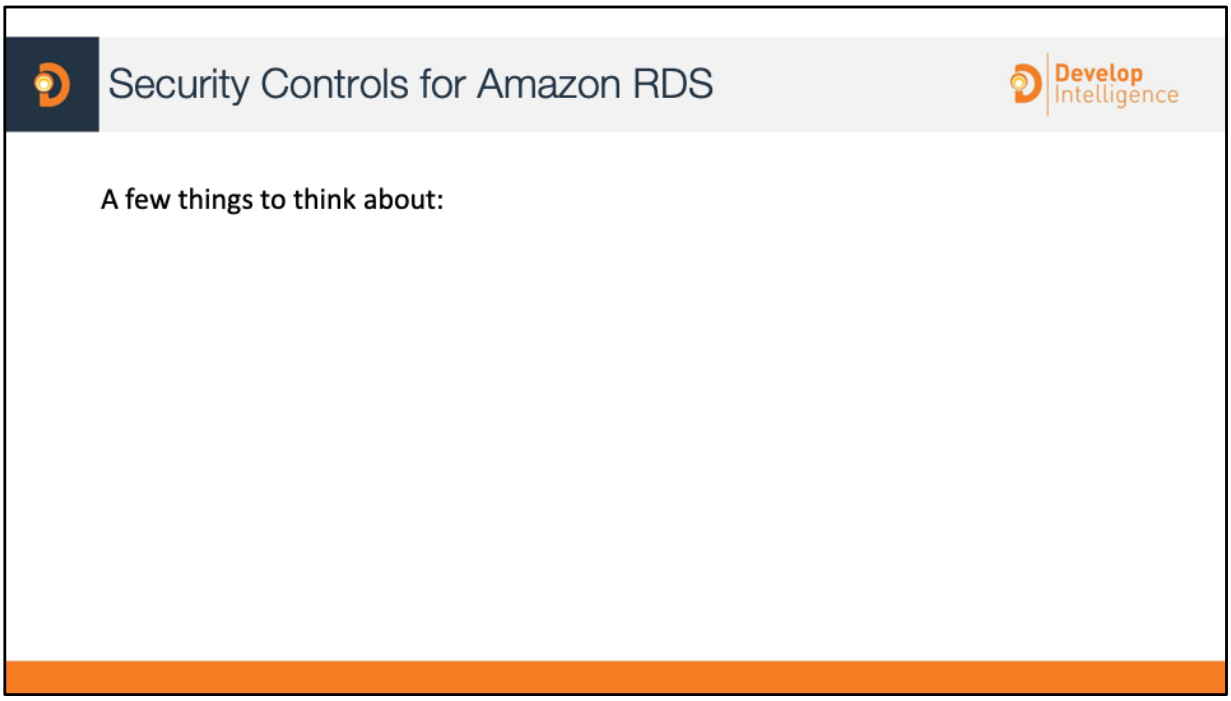


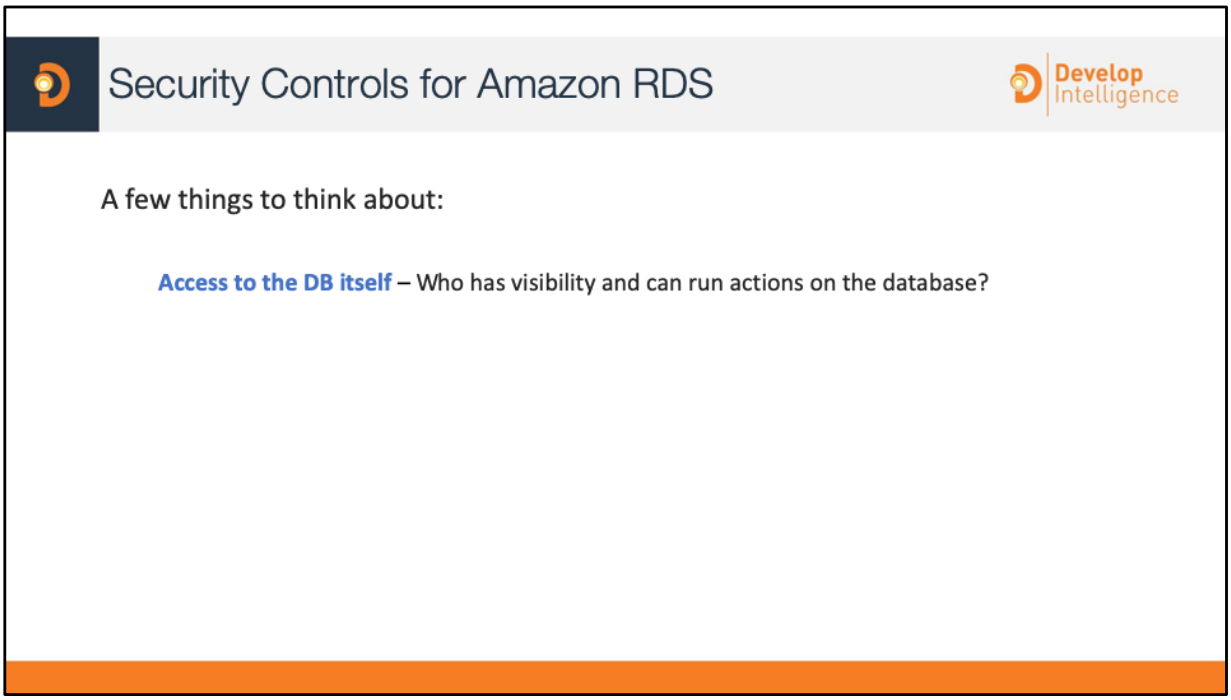
Read consistency represents the manner and timing in which the successful write or update of a data item is reflected in a subsequent read operation of that same item. Amazon DynamoDB exposes logic that enables you to specify the consistency characteristics you desire for each read request within your application.

Eventually Consistent Reads
When you read data from a DynamoDB table, the response might not reflect the results of a recently completed write operation. The response might include some stale data. If you repeat your read request after a short time, the response should return the latest data.

Strongly Consistent Reads
When you request a strongly consistent read, DynamoDB returns a response with the most up-to-date data, reflecting the updates from all prior write operations that were successful. A strongly consistent read might not be available if there is a network delay or outage.

DynamoDB uses eventually consistent reads, unless you specify otherwise. Read operations (such as GetItem, Query, and Scan) provide a ConsistentRead parameter. If you set this parameter to true, DynamoDB uses strongly consistent reads during the operation.

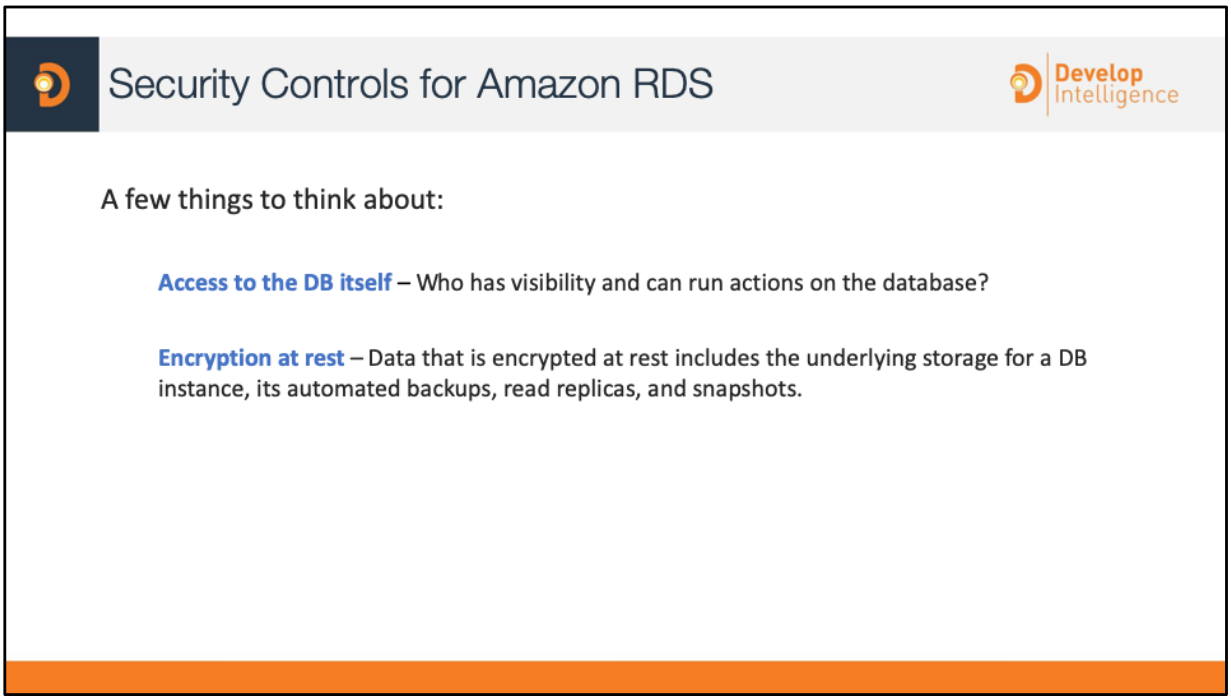

Security Controls for Amazon RDS
Develop Intelligence
A few things to think about:



Access Control

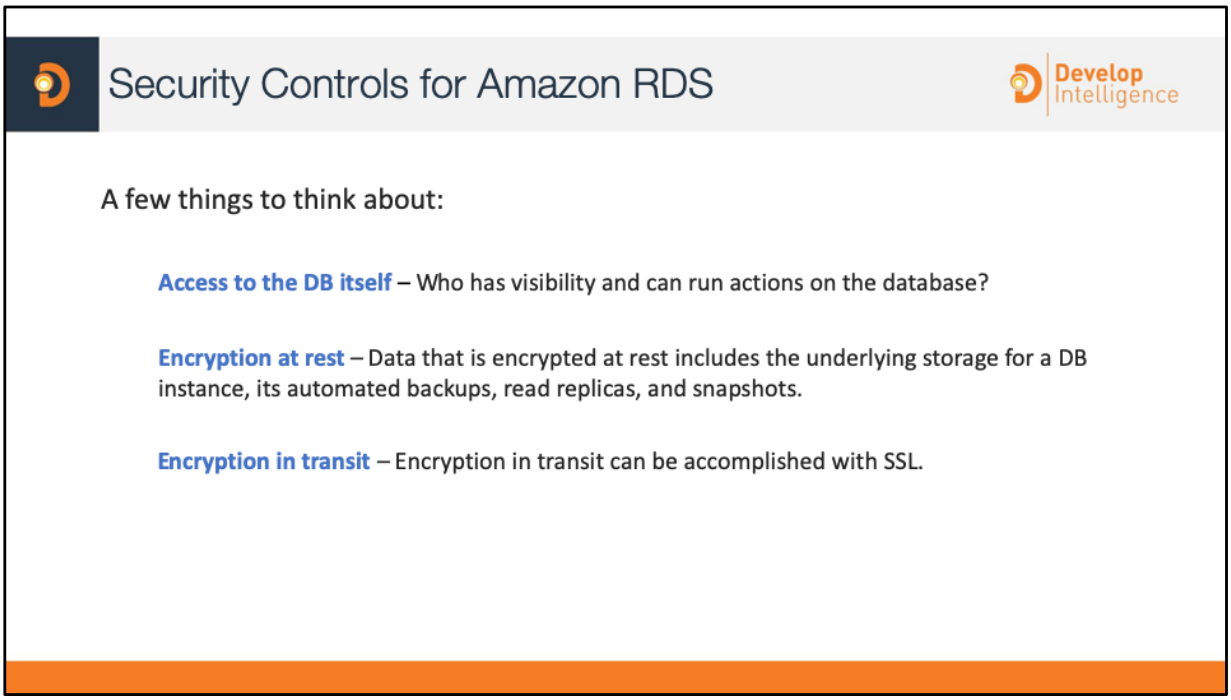
When you first create a DB instance within Amazon RDS, you create a master user account, which is used only within the context of Amazon RDS to control access to your DB instances. The master user account is a native database user account that allows you to log on to your DB instance with all database privileges. You can specify the master user name and password you want associated with each DB instance when you create the DB instance. After you create your DB instance, you can connect to the database using the master user credentials. Subsequently, you can create additional user accounts so that you can restrict who can access your DB instance.

You can control Amazon RDS DB instance access via DB security groups, which are similar to Amazon EC2 security groups but not interchangeable. DB security groups act like a firewall that controls network access to your DB instance. Database security groups default to a “deny all” access mode, and customers must specifically authorize network ingress. There are two ways to do this: authorize a network IP range, or authorize an existing Amazon EC2 security group. DB security groups only allow access to the database server port (all others are blocked) and can be updated without restarting the Amazon RDS DB instance, which gives a customer seamless control of their database access. Using AWS IAM, you can further control access to your RDS DB instances. AWS IAM enables you to control which RDS operations each individual AWS IAM user has permission to call.



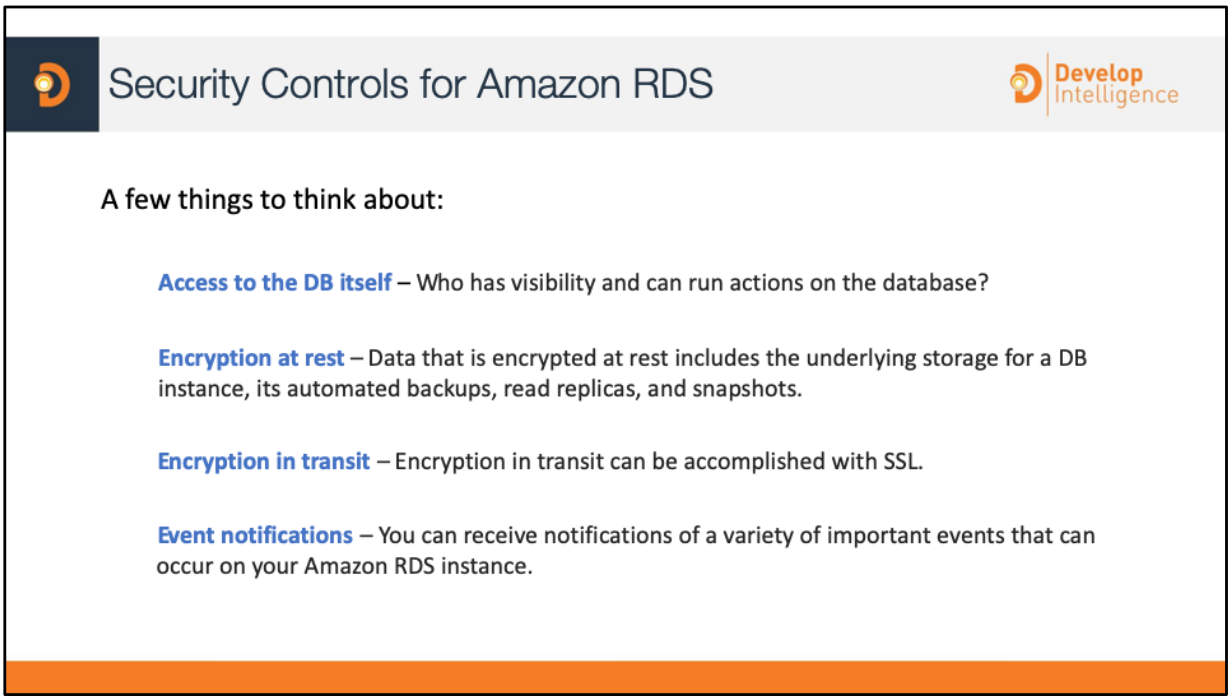
Encryption at Rest

Amazon RDS DB instances and snapshots at rest can be encrypted. When encryption has been enabled, the instance's automated backups, Read Replicas, and snapshots are encrypted using AES-256. Amazon RDS then handles authentication of access and decryption of this data with a minimal impact on performance, and with no need to modify your database client applications. This encryption is available for Amazon RDS with MySQL, PostgreSQL, Oracle, and SQL Server DB instances; however, it is not available for DB instances in the AWS GovCloud (us-gov-west-1) Region.



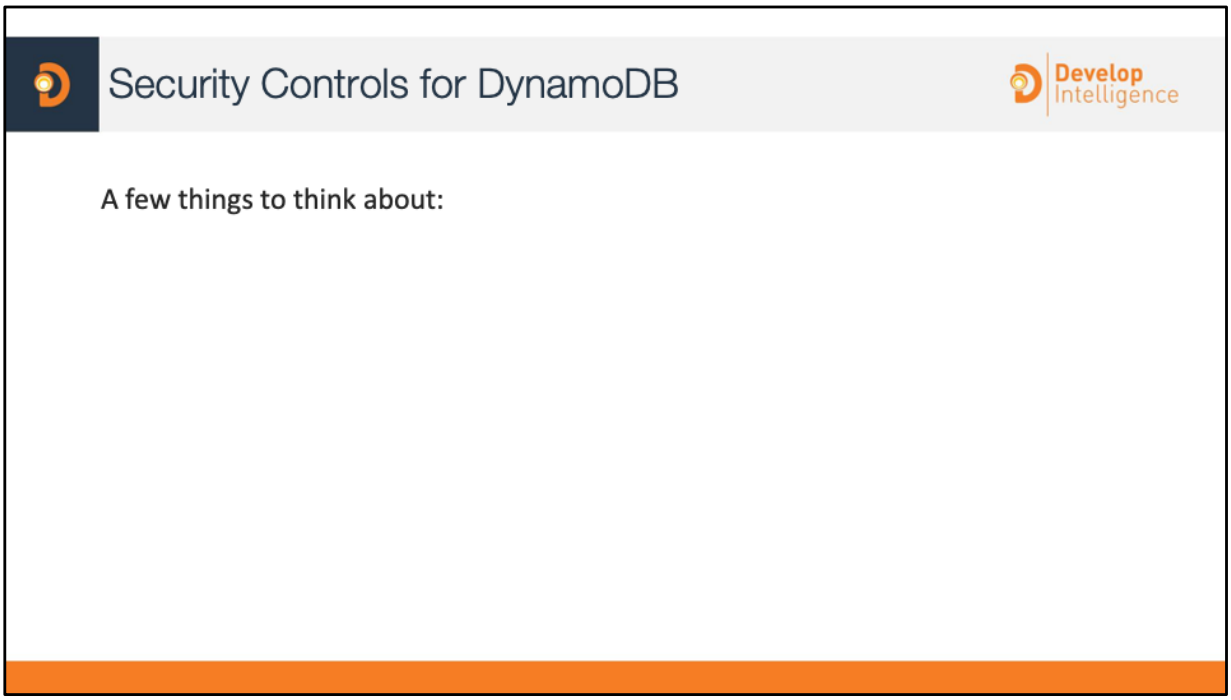
Encryption in Transit

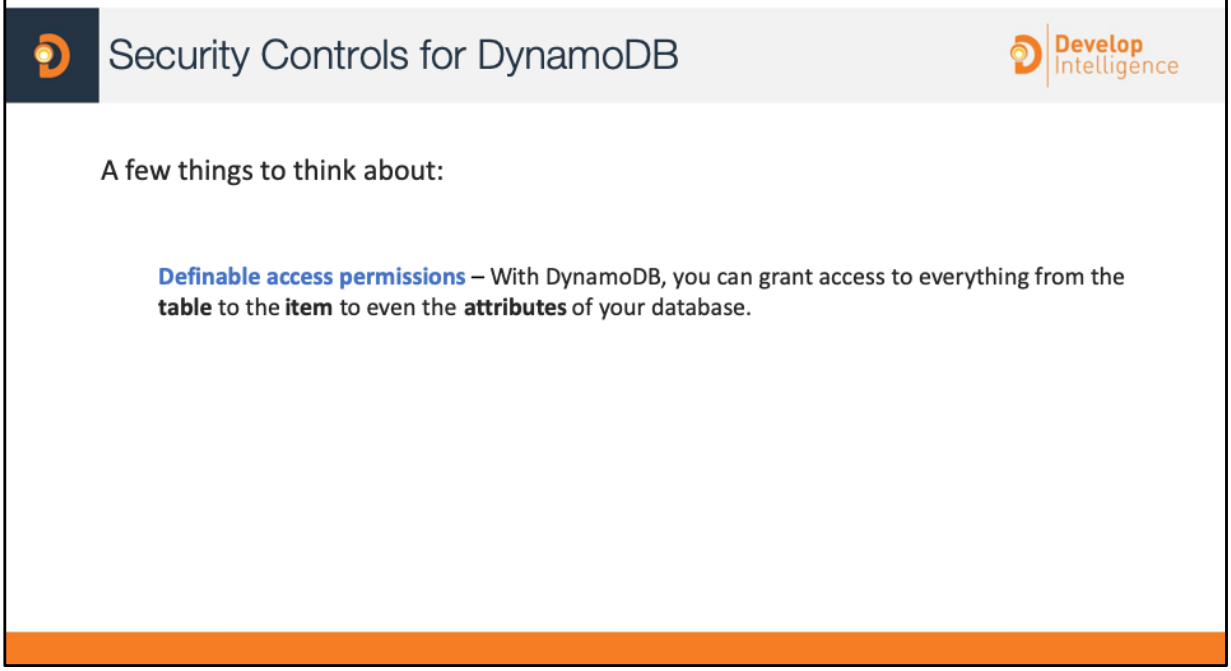
You can use SSL to encrypt connections between your application and your DB instance. For MySQL and SQL Server, RDS creates an SSL certificate and installs the certificate on the DB instance when the instance is provisioned. For MySQL, launch the MySQL client using the --ssl_ca parameter to reference the public key in order to encrypt connections. For SQL Server, download the public key and import the certificate into your Windows operating system. Oracle RDS uses Oracle native network encryption with a DB instance. You simply add the native network encryption option to an option group and associate that option group with the DB instance. When an encrypted connection is established, data transferred between the DB instance and your application will be encrypted during transfer. You can also require your DB instance to only accept encrypted connections. Note that SSL support within Amazon RDS is for encrypting the connection between your application and your DB instance; it should not be relied on for authenticating the DB instance itself. Although SSL offers security benefits, be aware that SSL encryption is a compute-intensive operation and will increase the latency of your database connection.

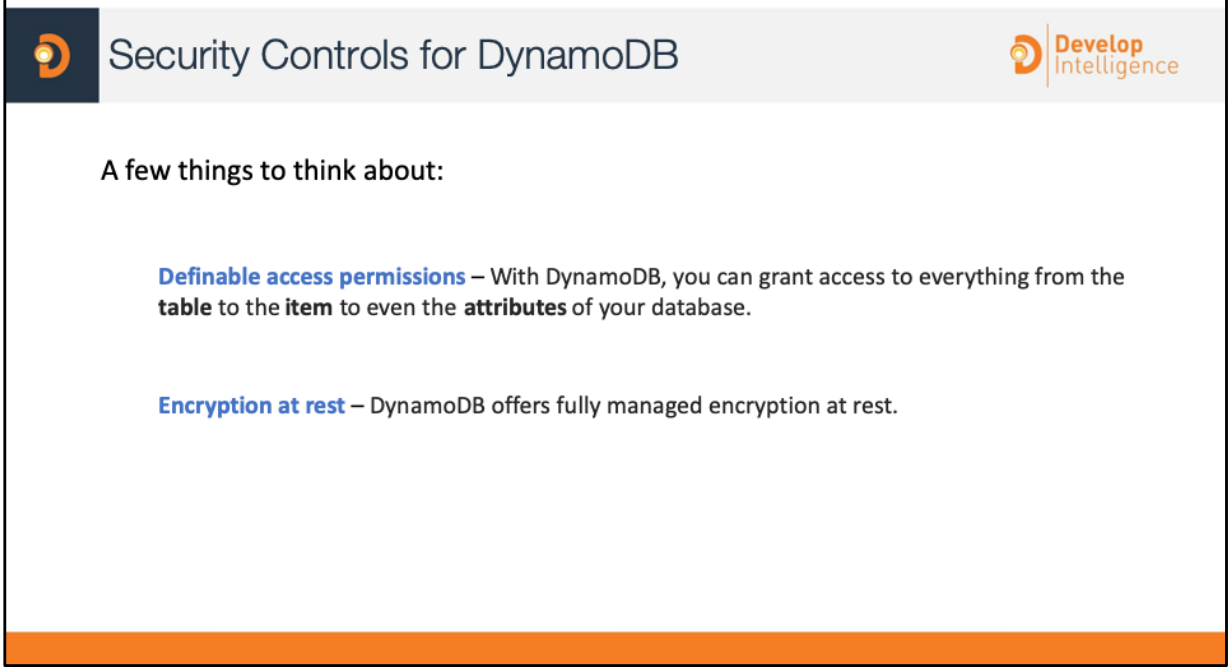


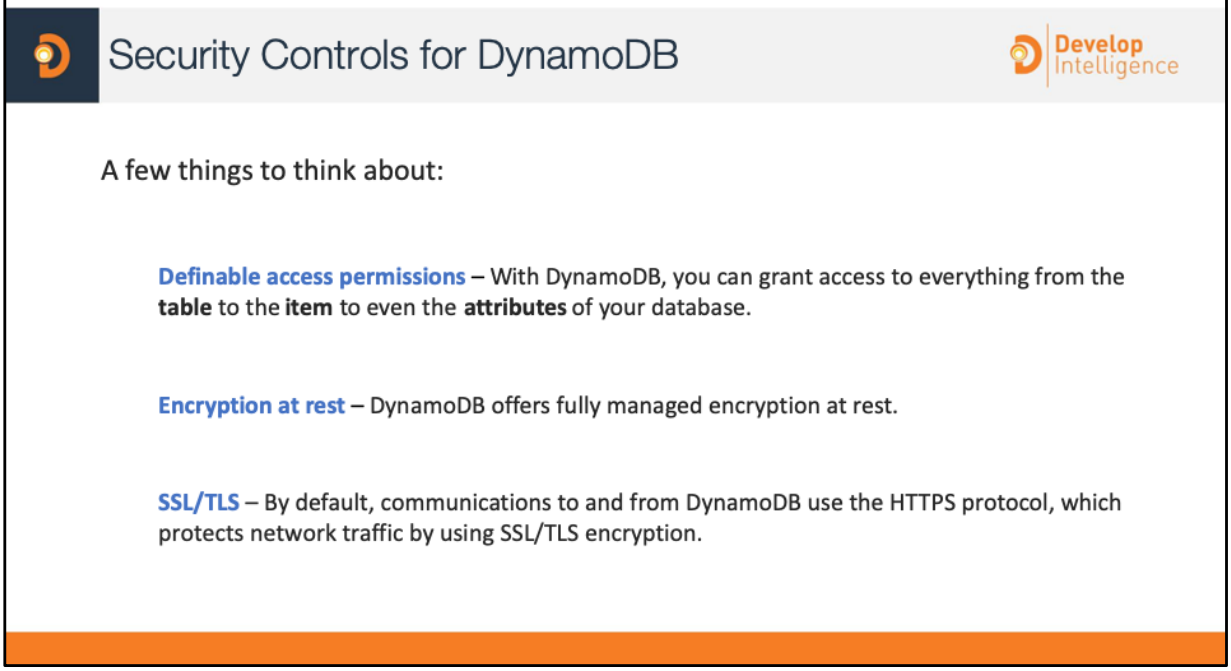
Event Notification

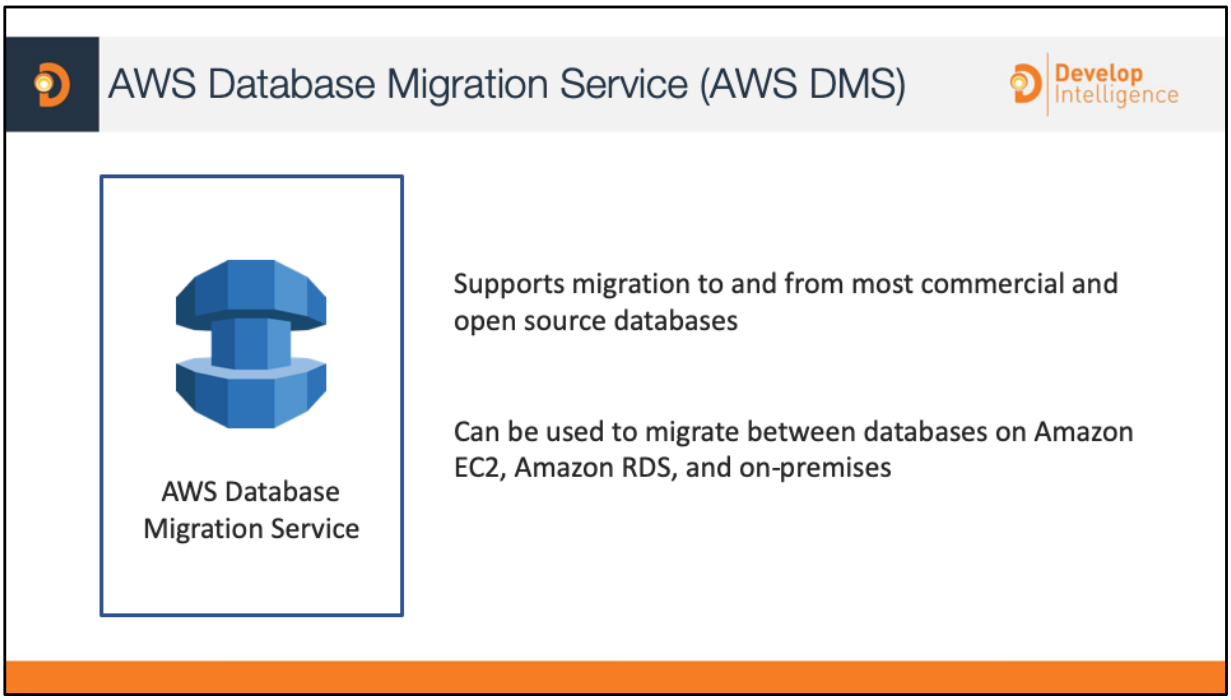
You can receive notifications of a variety of important events that can occur on your RDS instance, such as whether the instance was shut down, a backup was started, a failover occurred, the security group was changed, or your storage space is low. Amazon RDS groups events into categories that you can subscribe to so that you can be notified when an event in that category occurs. You can subscribe to an event category for a DB instance, DB snapshot, DB security group, or DB parameter group. Amazon RDS events are published via Amazon SNS and sent to you as an email or text message.


Security Controls for DynamoDB
Develop Intelligence
A few things to think about:


Security Controls for DynamoDB
Develop Intelligence
A few things to think about:
Definable access permissions – With DynamoDB, you can grant access to everything from the table to the item to even the attributes of your database.


Security Controls for DynamoDB
Develop Intelligence
A few things to think about:
Definable access permissions – With DynamoDB, you can grant access to everything from the table to the item to even the attributes of your database.
Encryption at rest – DynamoDB offers fully managed encryption at rest.


Security Controls for DynamoDB
Develop Intelligence
A few things to think about:
Definable access permissions – With DynamoDB, you can grant access to everything from the table to the item to even the attributes of your database.
Encryption at rest – DynamoDB offers fully managed encryption at rest.
SSL/TLS – By default, communications to and from DynamoDB use the HTTPS protocol, which protects network traffic by using SSL/TLS encryption.



AWS DMS supports migration between the most widely used databases (Oracle, PostgreSQL, Microsoft SQL Server, Amazon Redshift, Amazon Aurora, MariaDB, and MySQL). It also supports homogenous (same engine) and heterogeneous (different engines) migrations.

The service can be used to migrate between databases on Amazon EC2, Amazon RDS and on-premises.

- This includes migrating from Amazon EC2 or RDS databases to on-premises.
- Either the target or the source database must be located in Amazon EC2 (cannot migrate between two on-premises databases).

It automatically handles formatting of the source data for consumption by the target database.

It does not perform schema or code conversion.

- For homogenous migrations, you can use native tools to perform these conversions.
- For heterogeneous migrations, you can use the **AWS Schema Conversion Tool**.

For more information, see http://docs.aws.amazon.com/dms/latest/userguide/Welcome.html.

Migration Options

Develop Intelligence

AWS Database Migration Service

One-time migration

Ongoing migration

Using AWS Snowball Edge with AWS DMS

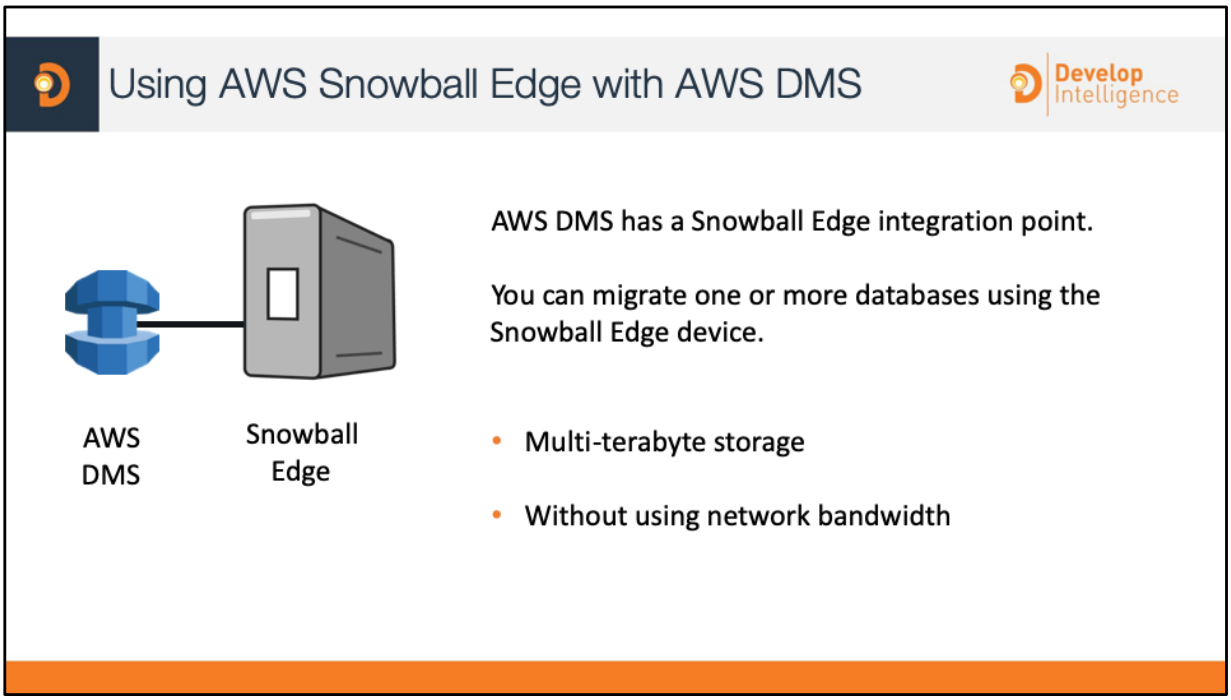
Develop Intelligence

AWS Database Migration Service

When migrating data is unfeasible:

- Database is too large
- Connection is too slow
- Privacy and security concerns

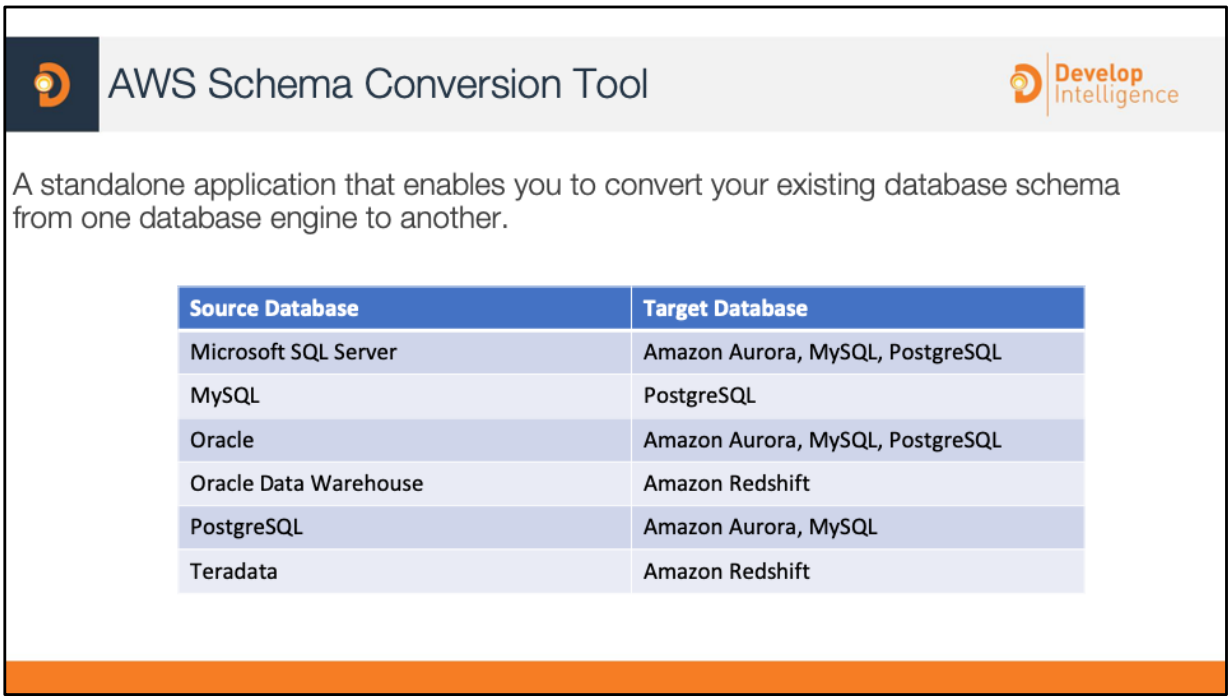
We recommend AWS Snowball Edge



You get the ability to physically attach a secure, hardened device directly inside your data center rather than opening ports from the outside.
You can now move very large databases from on-premises to AWS Cloud.
This integration provides a “push” model to migrate databases instead of a “pull” model.
You can migrate one or more databases using the same AWS Snowball Edge device without a need to upgrade your network infrastructure and consume dedicated bandwidth.
While migrating these multi-terabyte and multi-petabyte databases to AWS, your on-premises databases remain online. They can be decommissioned when the AWS Snowball Edge appliance is shipped back to AWS and is automatically loaded onto your target Amazon RDS– or Amazon EC2–based database.
You can migrate existing data (one time) or optionally perform ongoing data replication to the target database.

A few notes about working with AWS Snowball Edge and AWS DMS:

- The version of AWS Snowball required for integration with AWS DMS is AWS Snowball Edge.
- Currently, you need a Linux host to run the DMS Snowball agent.
- The AWS Snowball Edge must be on the same network as your source database.



For more on the AWS Schema Conversion Tool, go here: http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/Welcome.html