Cloud Computing

By

Dr. P. Sreedhar

AWS RDS
(Relational database service)

AWS RDS

- Amazon RDS is a fully managed relational database service.
- Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the cloud.
- It provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching and backups.
- It frees you to focus on your applications so you can give them the fast performance, high availability, security and compatibility they need.
- Amazon RDS is available on several **database instance types** - optimized for memory, performance or I/O - and provides you with six familiar database engines to choose from, including
  - Amazon Aurora,
  - PostgreSQL,
  - MySQL,
  - MariaDB,
  - Oracle Database, and
  - SQL Server.
- You can use the **AWS Database Migration Service** to easily migrate or replicate your existing databases to Amazon RDS.

Amazon *RDS*

Amazon RDS engines

Commercial

ORACLE

Microsoft SQL Server

Open source

MySQL

PostgreSQL

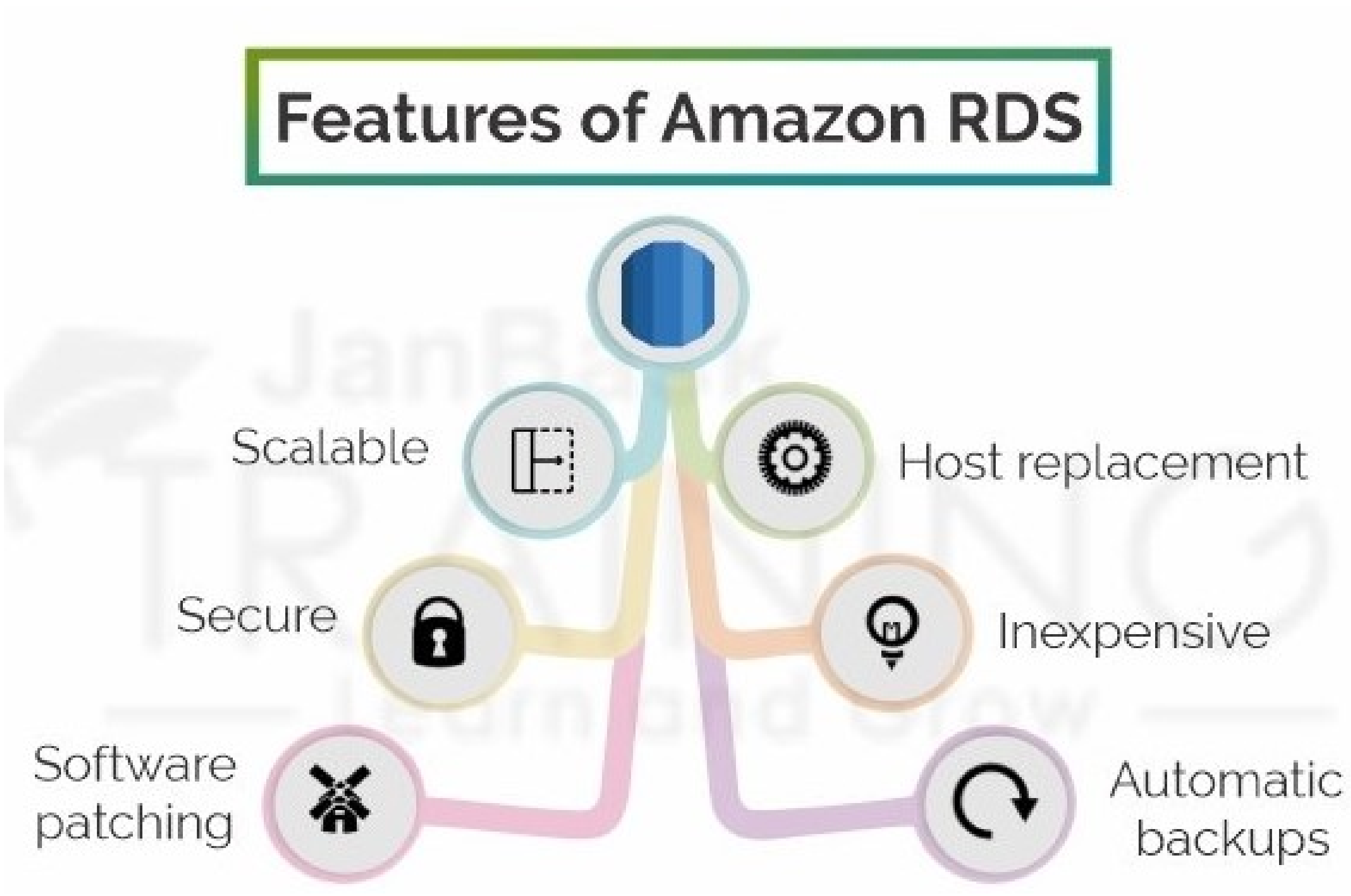
MariaDB

Cloud native

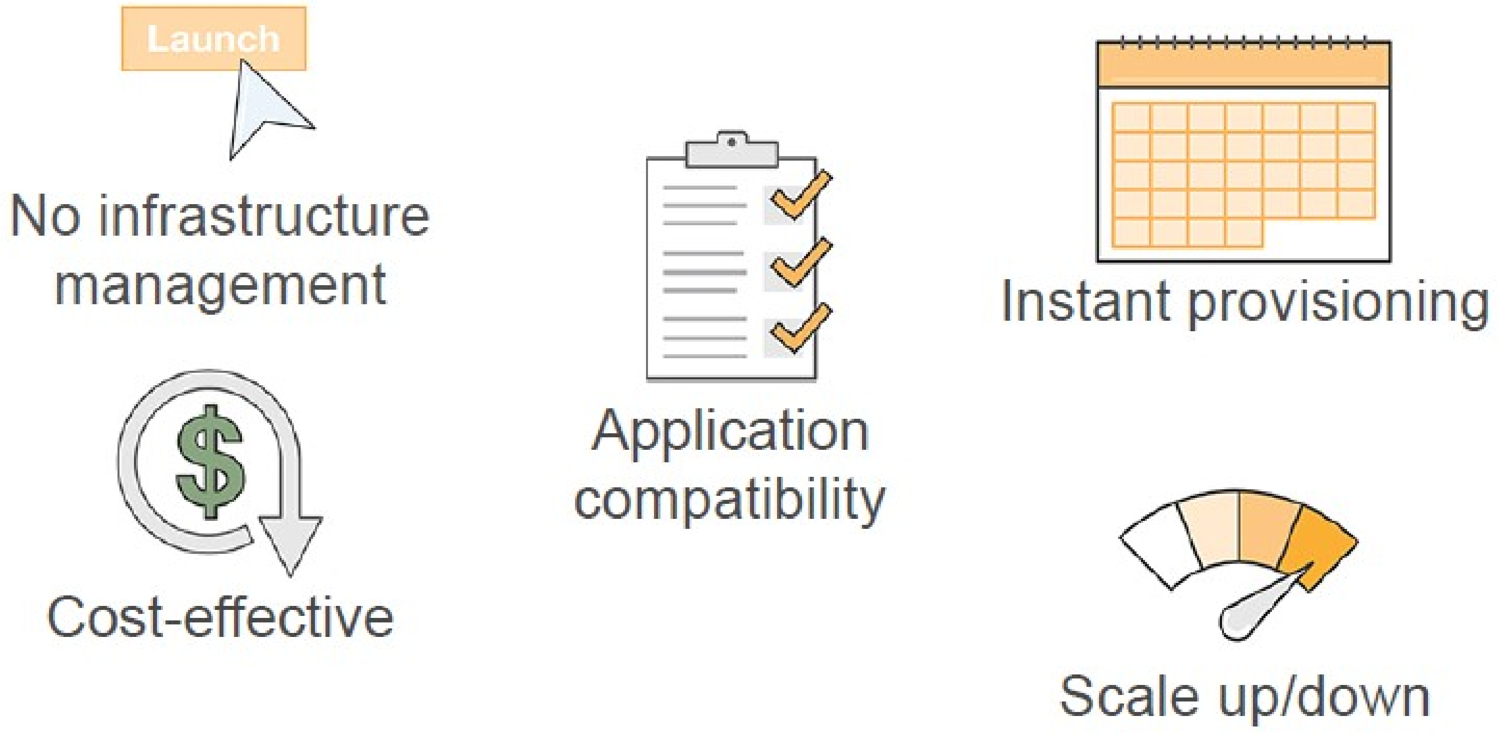
Amazon Aurora

AWS RDS

•


Features of Amazon RDS
Scalable
Host replacement
Secure
Inexpensive
Software patching
Automatic backups

Amazon *RDS*


Launch
No infrastructure management
Application compatibility
Instant provisioning
Cost-effective
Scale up/down

AWS RDS features

- **Scalable**– Amazon RDS permits to scale the social database by utilizing the AWS Management Console or RDS-explicit API. We can either increase or decrease your RDS prerequisites within minutes.
- **Inexpensive**– Using Amazon RDS, we pay just for the features that we actually use. There is no forthcoming and long-term payment.
- **Secure**– Amazon RDS gives unlimited authority over the system to get to their database and their related services.
- **Automatic backups**– Amazon RDS backs up everything in the database including exchange logs up to most recent five minutes and furthermore oversees programmed backup timings.
- **Software patching**– Automatically gets all the most recent patches for the database programming. We can likewise indicate when the product ought to be fixed utilizing DB Engine Version Management.
-

AWS RDS benefits

- https://console.aws.amazon.com/rds/ -→ AWS CONSOLE

Benefits:

- Easy to administer
- Highly scalable
- Available and durable
- Fast
- Secure
- Inexpensive
- reliable

Easy to administer:

- Amazon RDS makes it easy to go from project conception to deployment. Use the Amazon RDS Management Console, the AWS RDS Command-Line Interface, or simple API calls to access the capabilities of a production-ready relational database in minutes.
- No need for infrastructure provisioning, and no need for installing and maintaining database software.

Highly scalable

- You can scale your database's compute and storage resources with only a few mouse clicks or an API call, often with no downtime.
- Many Amazon RDS engine types allow you to launch one or more Read Replicas to offload read traffic from your primary database instance.

AWS RDS benefits

Available and durable:

- Amazon RDS runs on the same highly reliable infrastructure used by other Amazon Web Services.
- When you provision a Multi-AZ DB Instance, Amazon RDS synchronously replicates the data to a standby instance in a different Availability Zone (AZ). Amazon RDS has many other features that enhance reliability for critical production databases, including automated backups, database snapshots, and automatic host replacement.

Fast

- Amazon RDS supports the most demanding database applications. You can choose between two SSD-backed storage options: one optimized for high-performance OLTP applications, and the other for cost-effective general-purpose use.
- In addition, Amazon Aurora provides performance on par with commercial databases at 1/10th the cost.

AWS RDS benefits

Secure

- Amazon RDS makes it easy to control network access to your database. Amazon RDS also lets you run your database instances in Amazon Virtual Private Cloud (Amazon VPC), which enables you to isolate your database instances and to connect to your existing IT infrastructure through an industry-standard encrypted IPsec VPN.
- Many Amazon RDS engine types offer encryption at rest and encryption in transit.

Inexpensive

- You pay very low rates and only for the resources you actually consume.
- In addition, you benefit from the option of On-Demand pricing with no up-front or long-term commitments, or even lower hourly rates via our Reserved Instance pricing.

Deploying with Amazon *RDS*

The normal process for deploying a DBMS is:

- Choose the system
- Obtain the software:
  - If commercial software, purchase license for estimated usage
  - If open source software, choose the version and download the software
- Specify, purchase, install, and configure the platform (hardware and disk storage)
- Install the DBMS
- Configure the administrative user and the initial database
- Secure the system

Setup Amazon *RDS*

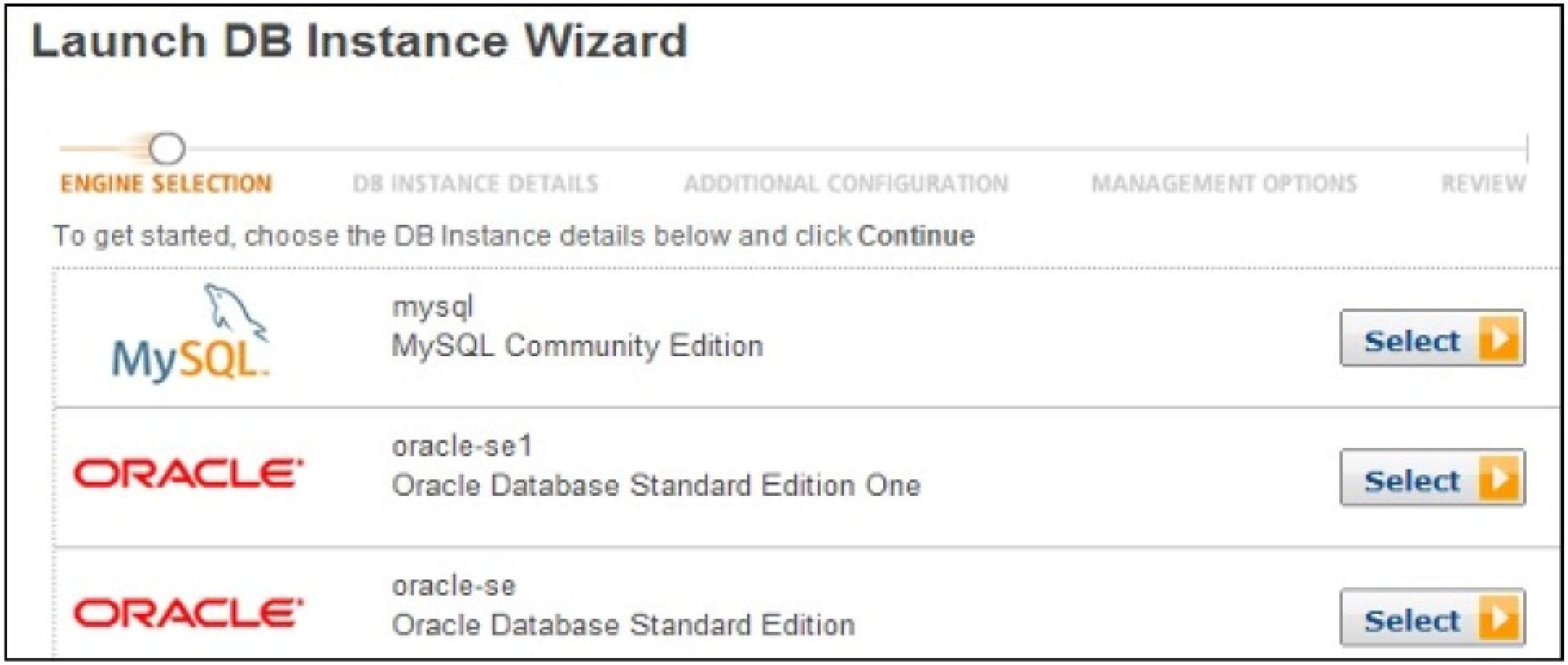
Steps:

1. Login to AWS Mgt. Console. Using the link: https://console.aws.amazon.com/rds/

2. Select the region where the db instance is to be created, at top right corner of the rds console.

3. Select instances in navigation pane. Then click 'Launch DB instance' button. The Launch DB instance wizard opens.

4. Select the type of instance and click Select button.



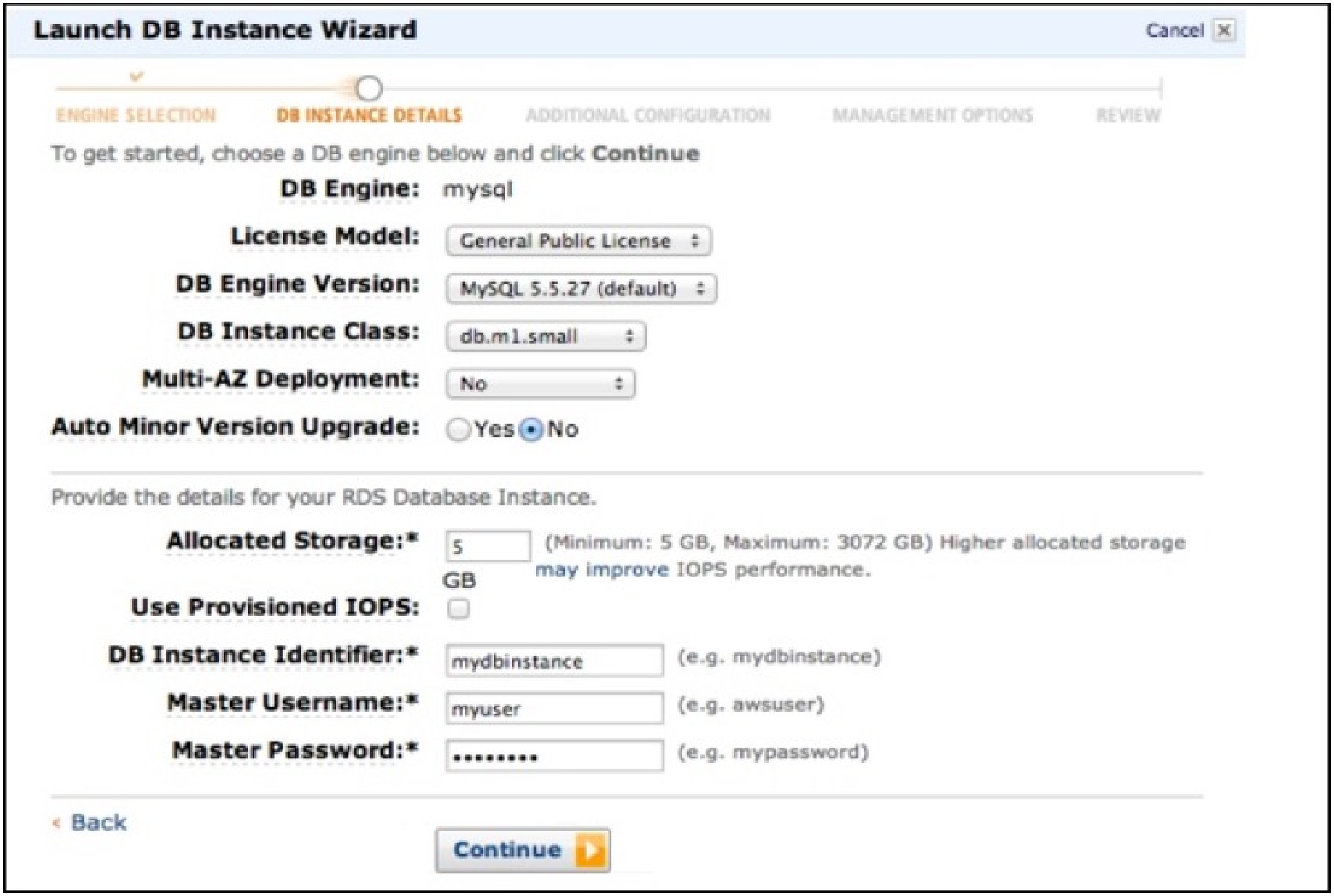
Launch DB Instance Wizard

To get started, choose the DB Instance details below and click **Continue**

MySQL	mysql MySQL Community Edition	Select
ORACLE	oracle-se1 Oracle Database Standard Edition One	Select
ORACLE	oracle-se Oracle Database Standard Edition	Select

Setup Amazon *RDS*

5. Give the details in 'Launch DB instance Wizard' dialog:



To get started, choose a DB engine below and click **Continue**

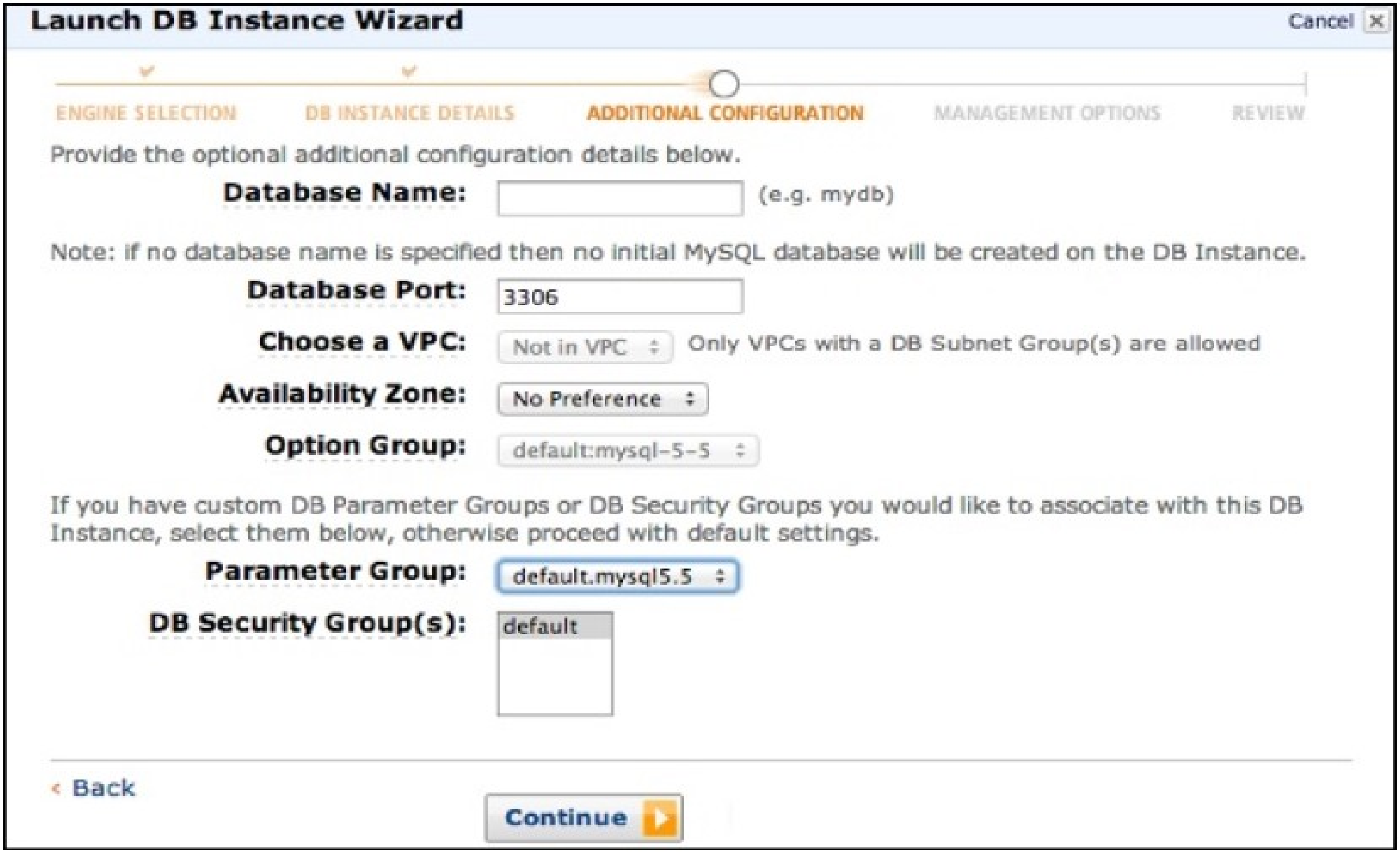
DB Engine:	mysql
License Model:	General Public License
DB Engine Version:	MySQL 5.5.27 (default)
DB Instance Class:	db.m1.small
Multi-AZ Deployment:	No
Auto Minor Version Upgrade:	Yes [✓ No]

Provide the details for your RDS Database Instance.

Allocated Storage:*	5 GB	(Minimum: 5 GB, Maximum: 3072 GB) Higher allocated storage may improve IOPS performance.
Use Provisioned IOPS:	☐	
DB Instance Identifier:*	mydbinstance	(e.g. mydbinstance)
Master Username:*	myuser	(e.g. awsuser)
Master Password:*	••••••••	(e.g. mypassword)

Setup Amazon *RDS*

Step 6: On the 'Additional Configuration' page, provide additional information required to launch MySql db instance.



Launch DB Instance Wizard

Provide the optional additional configuration details below.

Database Name: (e.g. mydb)

Note: if no database name is specified then no initial MySQL database will be created on the DB Instance.

Database Port: 3306

Choose a VPC: Not in VPC Only VPCs with a DB Subnet Group(s) are allowed

Availability Zone: No Preference

Option Group: default:mysql-5-5

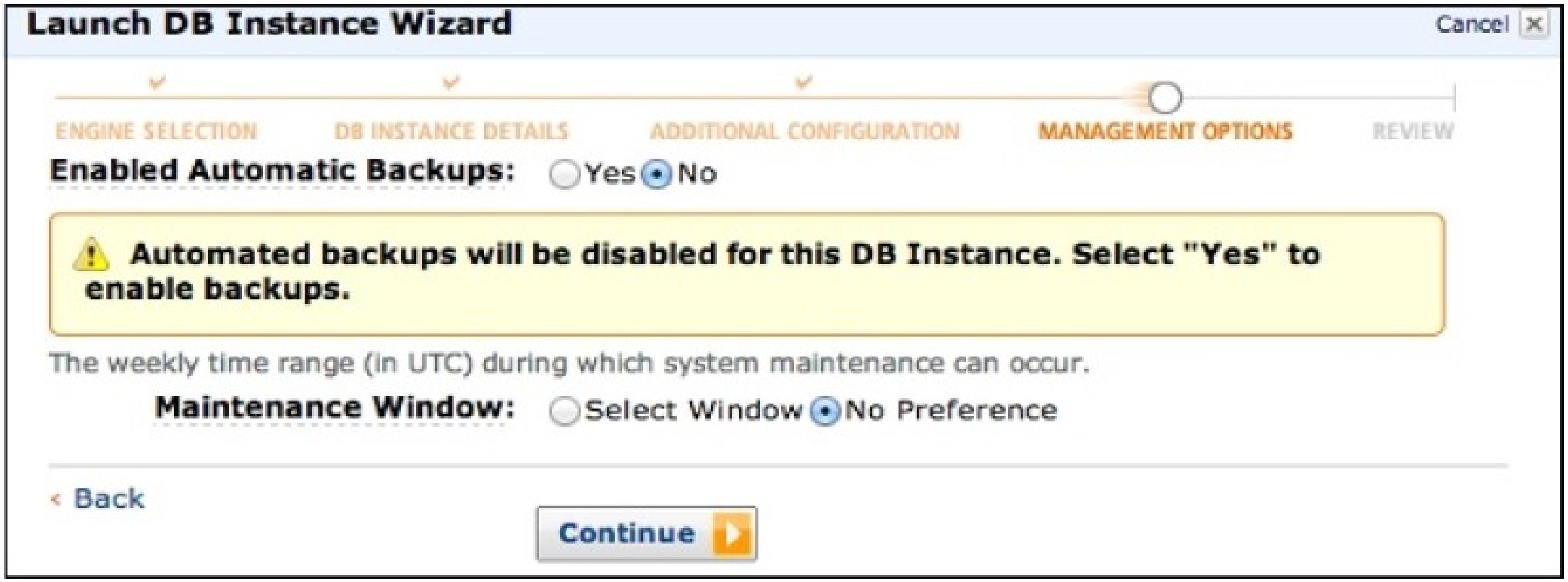
If you have custom DB Parameter Groups or DB Security Groups you would like to associate with this DB Instance, select them below, otherwise proceed with default settings.

Parameter Group: default.mysql5.5

DB Security Group(s): default

Setup Amazon *RDS*

Step 7: On 'Management Options' page,select options and click 'Continue'.



Enabled Automatic Backups: Yes [✓ No]

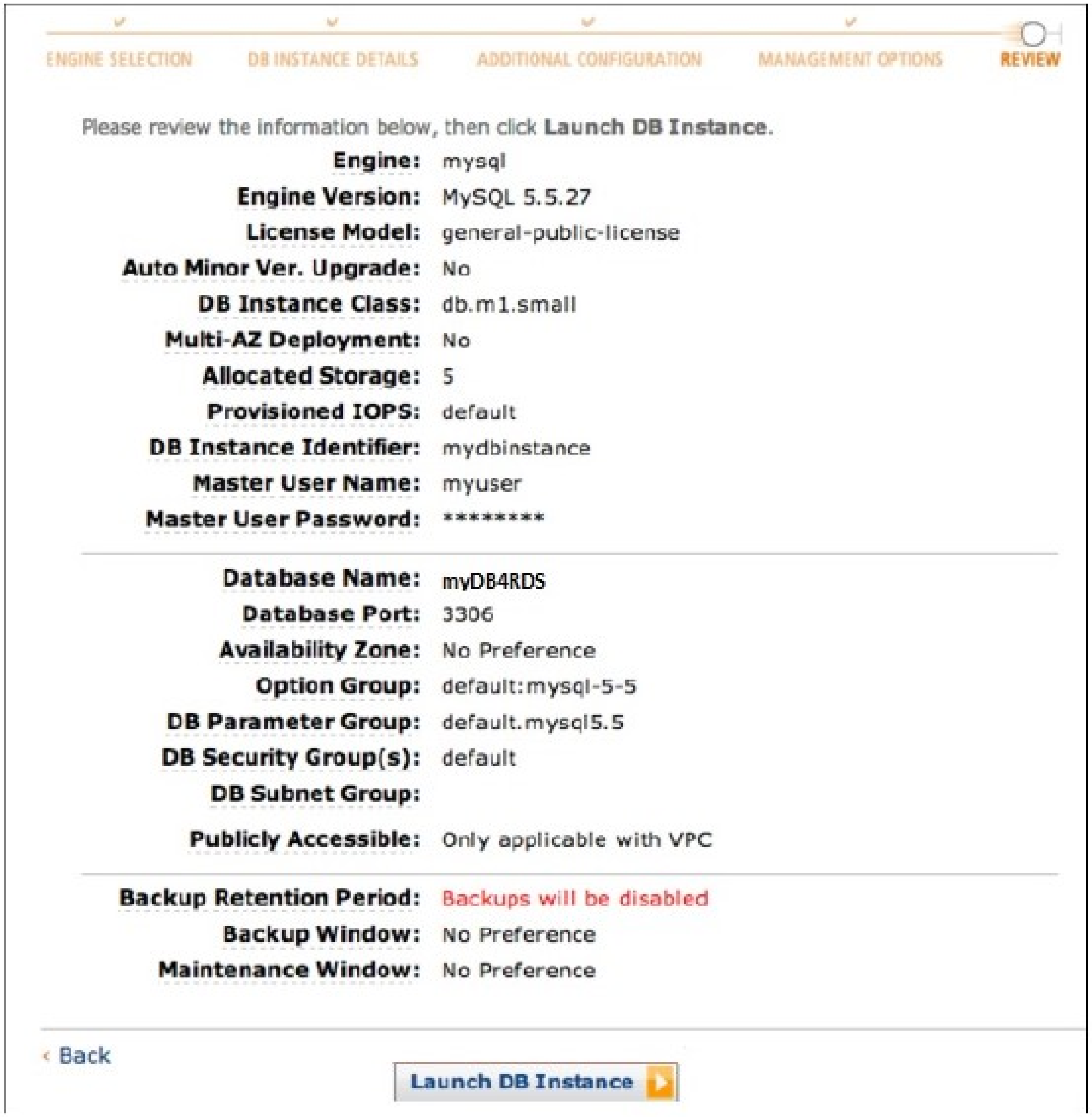
Automated backups will be disabled for this DB Instance. Select "Yes" to enable backups.

The weekly time range (in UTC) during which system maintenance can occur.

Maintenance Window: Select Window [✓ No Preference]

Setup Amazon *RDS*

Step 8: Review and click 'Launch DB instance' button to Finish.



Please review the information below, then click **Launch DB Instance**.

Engine:	mysql
Engine Version:	MySQL 5.5.27
License Model:	general-public-license
Auto Minor Ver. Upgrade:	No
DB Instance Class:	db.m1.small
Multi-AZ Deployment:	No
Allocated Storage:	5
Provisioned IOPS:	default
DB Instance Identifier:	mydbinstance
Master User Name:	myuser
Master User Password:	********
Database Name:	myDB4RDS
Database Port:	3306
Availability Zone:	No Preference
Option Group:	default:mysql-5-5
DB Parameter Group:	default.mysql5.5
DB Security Group(s):	default
DB Subnet Group:	
Publicly Accessible:	Only applicable with VPC
Backup Retention Period:	Backups will be disabled
Backup Window:	No Preference
Maintenance Window:	No Preference

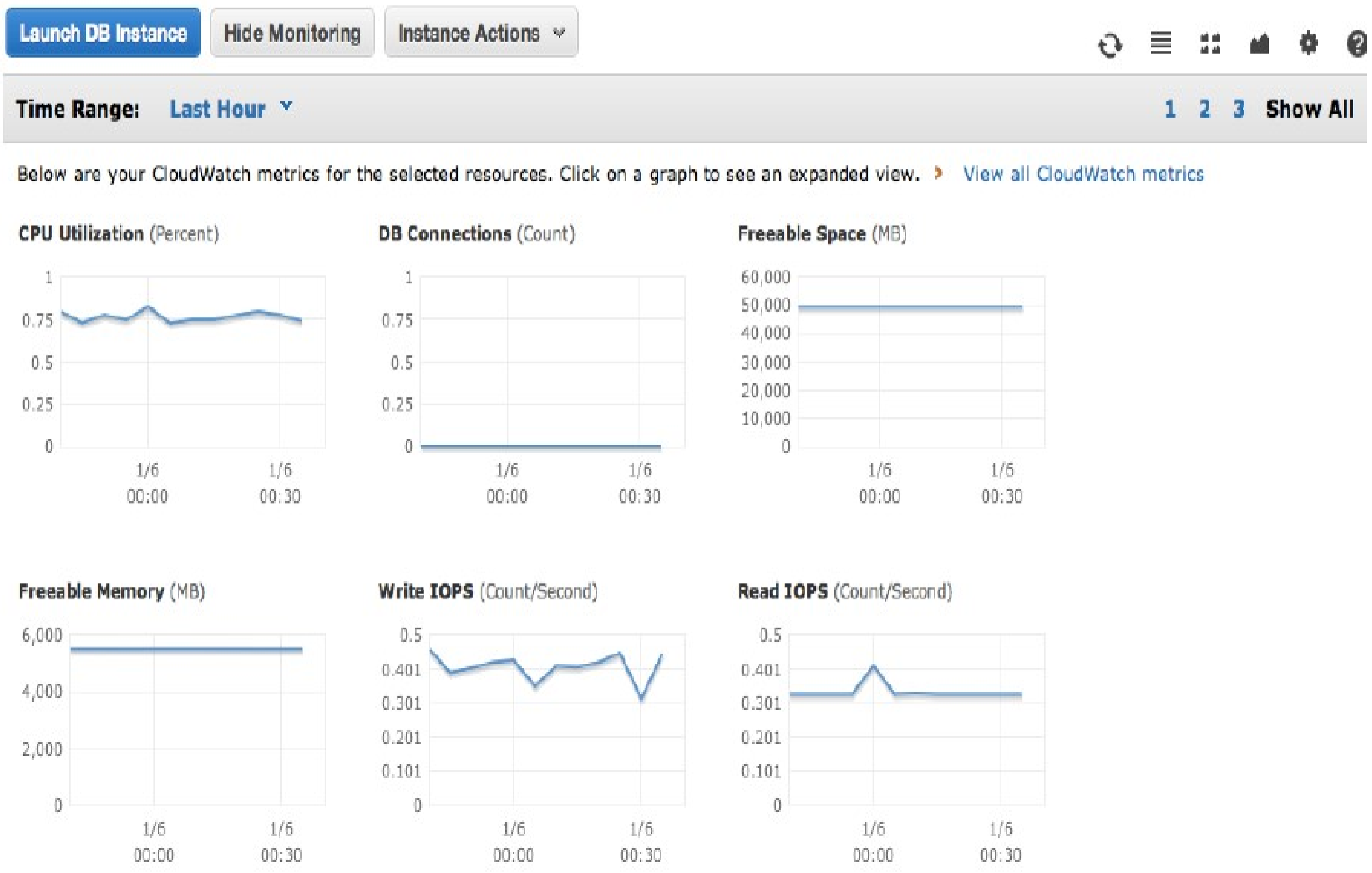
Cost of Amazon *RDS*

Parameters:

1. Instance Class
2. Running Time
3. Storage
4. I/O requests per month
5. Backup storage

•

Amazon *RDS*



Below are your CloudWatch metrics for the selected resources. Click on a graph to see an expanded view. › View all CloudWatch metrics

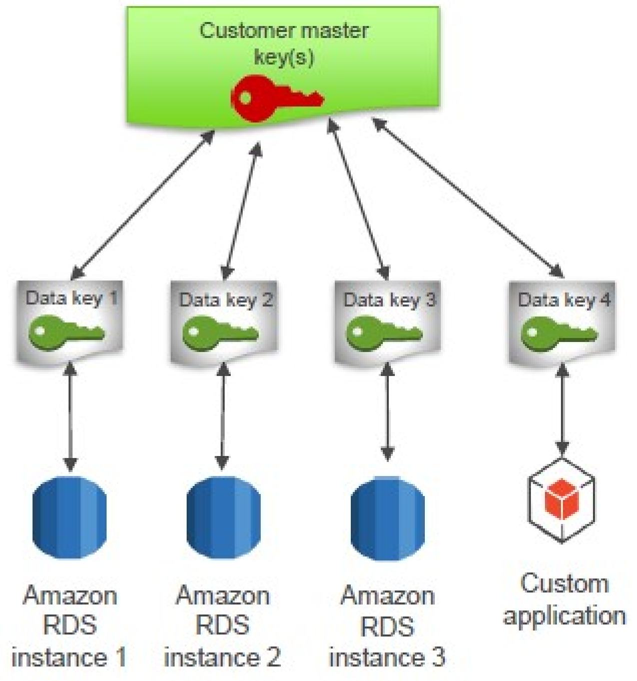
At Rest Encryption for all RDS Engines AWS Key Management Service (KMS)

Two-tiered key hierarchy using envelope encryption:

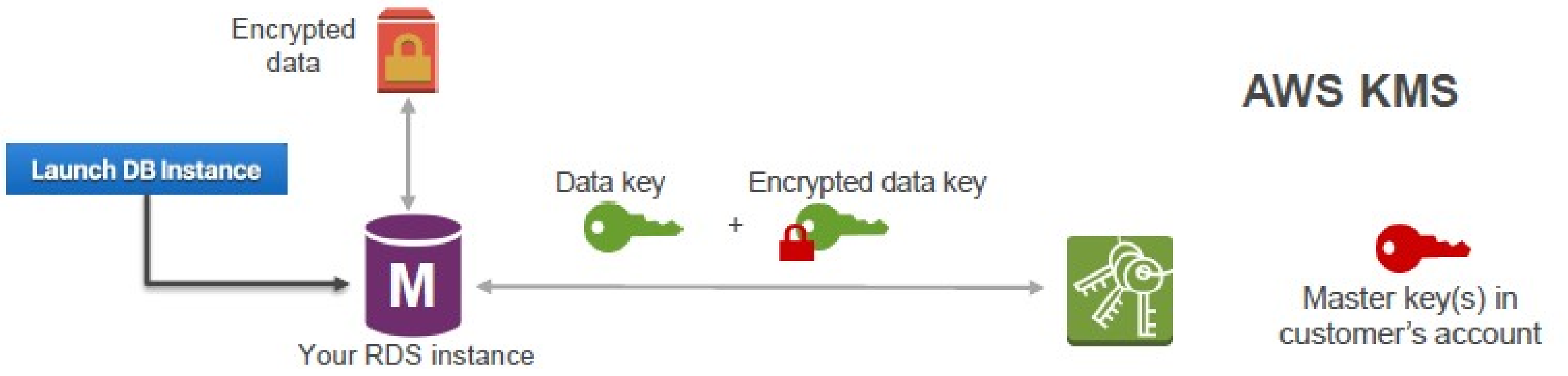
- Unique data key encrypts customer data
- AWS KMS master keys encrypt data keys
- Available for **ALL** RDS engines

Benefits:

- Limits risk of compromised data key
- Better performance for encrypting large data
- Easier to manage small number of master keys than millions of data keys
- Centralized access and audit of key activity

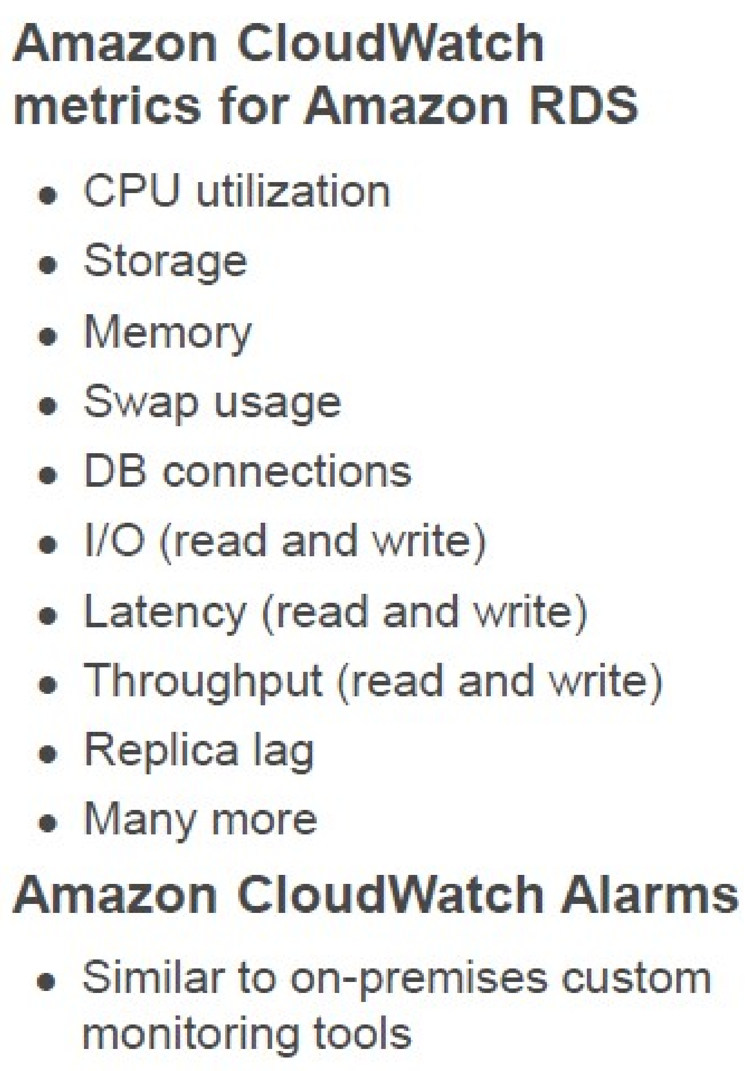


How keys are used to protect your data



1. Launch your RDS instance
2. RDS instance requests encryption key to use to encrypt data, passes reference to master key in account
3. Client request authenticated based on permissions set on both the user and the key
4. A unique data encryption key is created and encrypted under the KMS master key
5. Plaintext and encrypted data key returned to RDS
6. Plaintext data key stored in memory and used to encrypt/decrypt RDS data

Amazon *RDS*



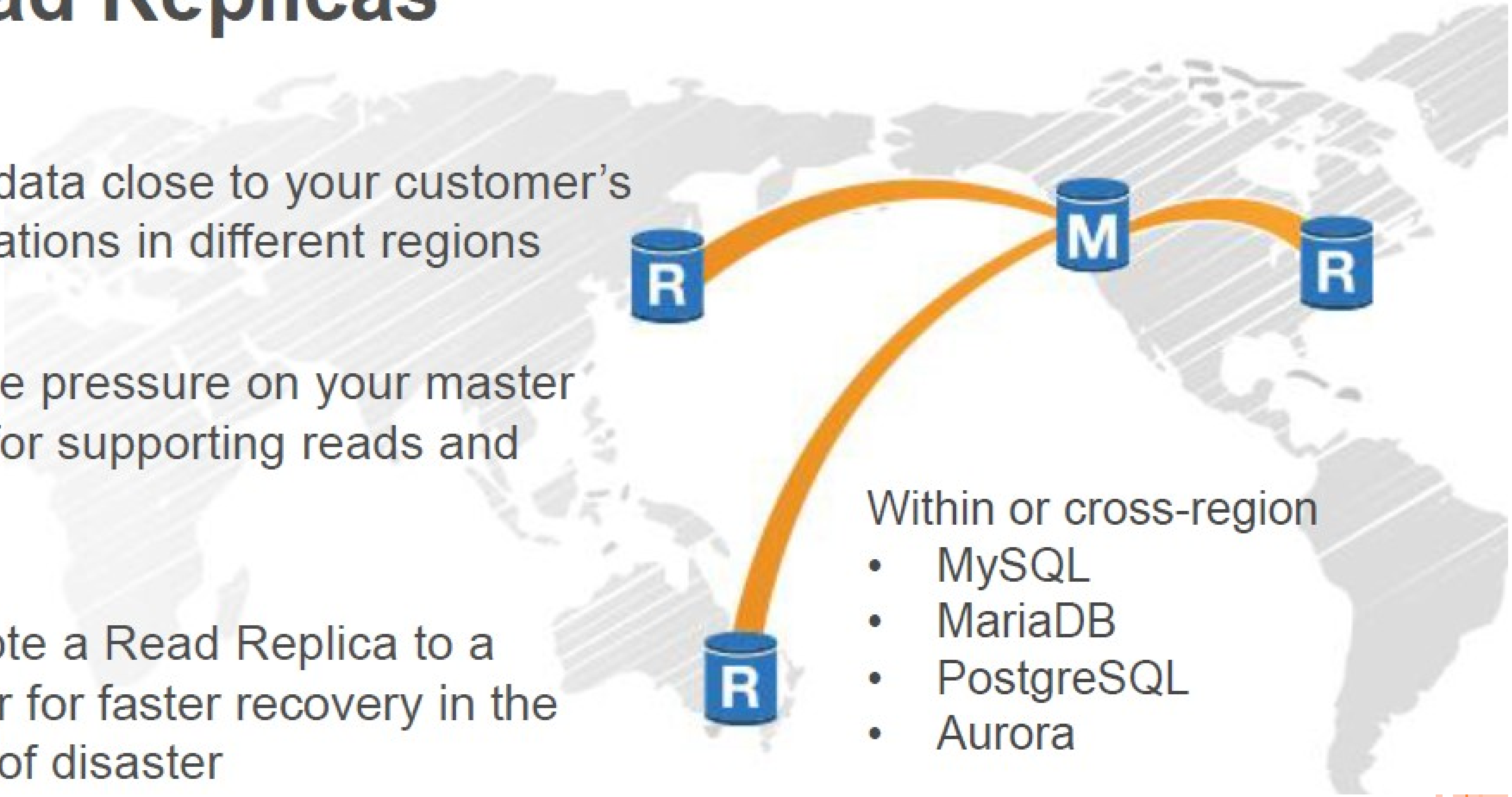
Amazon CloudWatch metrics for Amazon RDS

- CPU utilization
- Storage
- Memory
- Swap usage
- DB connections
- I/O (read and write)
- Latency (read and write)
- Throughput (read and write)
- Replica lag
- Many more

Amazon CloudWatch Alarms

- Similar to on-premises custom monitoring tools

Read Replicas



Bring data close to your customer's applications in different regions

Relieve pressure on your master node for supporting reads and writes

Promote a Read Replica to a master for faster recovery in the event of disaster

Within or cross-region
- MySQL
- MariaDB
- PostgreSQL
- Aurora

Amazon *RDS*

Automated Backups

MySQL, PostgreSQL, MariaDB, Oracle, SQL Server

- Scheduled daily volume backup of entire instance
- Archive database change logs
- 35-day retention
- Taken from standby when running multi-AZ

Aurora

- Automatic, continuous, incremental backups
- No impact on database performance
- 35-day retention

Amazon *RDS*

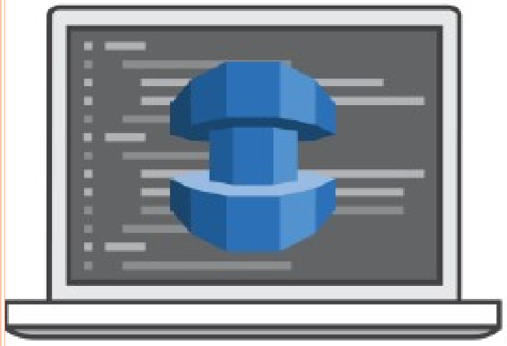
PostgreSQL ORACLE Amazon Aurora

AMAZON REDSHIFT MySQL MariaDB Microsoft SQL Server

AWS Database Migration Service

- ✓ Move data to the same or different database engine
- ✓ Keep your apps running during the migration
- ✓ Start your first migration in 10 minutes or less
- ✓ Replicate within, to, or from Amazon EC2 or RDS

Amazon *RDS*



AWS Schema Conversion Tool

- ✓ Migrate from Oracle and SQL Server
- ✓ Move your tables, views, stored procedures, and data manipulation language (DML) to MySQL, MariaDB, and Aurora
- ✓ Highlight where manual edits are needed

AWS RDS:

AWS RDS

Why do you want a managed relational database service? Because Amazon RDS takes over many of the difficult and tedious management tasks of a relational database:

- ✓ When you buy a server, you get CPU, memory, storage, and IOPS, all bundled together. With Amazon RDS, these are split apart so that you can **scale** them independently. If you need more CPU, less IOPS, or more storage, you can easily allocate them.
- ✓ Amazon RDS manages **backups, software patching, automatic failure detection, and recovery.**
- ✓ To deliver a managed service experience, Amazon RDS **doesn't provide shell access to DB instances.** It also restricts access to certain system procedures and tables that require advanced privileges.
- ✓ You can have **automated backups** performed when you need them, or manually create your own backup snapshot. You can use these backups to restore a database. The Amazon RDS restore process works reliably and efficiently.
- ✓ You can use the **database products** you are already familiar with: MySQL, MariaDB, PostgreSQL, Oracle, Microsoft SQL Server.
- ✓ You can get **high availability** with a primary instance and a synchronous secondary instance that you can fail over to when problems occur. You can also use MariaDB, Microsoft SQL Server, MySQL, Oracle, and PostgreSQL read replicas to increase read scaling.
- ✓ In addition to the **security** in your database package, you can help control who can access your RDS databases by using **AWS Identity and Access Management (IAM)** to define users and permissions. You can also help protect your databases by putting them in a virtual private cloud.

AWS RDS:

AWS RDS - MySQL DB

***1. Crating an Amazon RDS DB instance** (CreateDBInstance)*

- *~~using Console~~*
- *~~using AWS CLI~~*
- *~~using RDS API~~*

~~2. Connecting to an Amazon RDS DB instance~~

- *~~Finding the connection information for a MySQL DB instance~~*
- ~~Connecting from the MySQL client~~
- *~~using MySQL client (Mysql workbench)~~*
- *~~using RDS API~~*
- *~~MySQL commandline tool (mysql shell)/ AWS CLI~~*

~~3. Getting DescribeDBInstances information~~

- *~~using Console~~*
- *~~using AWS CLI~~*
- *~~using RDS API~~*

4. Importing and Exporting data

5. Managing and RDS DB Instance

- *Starting, Stopping, Rebooting, Modifying, Renaming, Deleting DB instances*

6. Working with RDS events

AWS RDS - Connecting to Db

Reasons for connection to db failure:

- The RDS DB instance is in a state other than available, so it can't accept connections.
- The source you use to connect to the DB instance is missing from the sources authorized to access the DB instance in your security group, network access control lists (ACLs), or local firewalls.
- The wrong DNS name or endpoint was used to connect to the DB instance.
- The Multi-AZ DB instance failed over, and the secondary DB instance uses a subnet or route table that doesn't allow inbound connections.
- The user authentication is incorrect.

Sdk urls:

https://docs.aws.amazon.com/code-samples/latest/catalog/javav2-rds-src-main-java-com-example-rds-CreateDBInstance.java.html

https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/rds/AmazonRDSClient.html

https://docs.aws.amazon.com/code-samples/latest/catalog/javav2-rds-src-main-java-com-example-rds-DescribeDBInstances.java.html

AWS RDS - Connecting to MySql Db instance

- The connection information for a DB instance includes its **endpoint, port, and a valid database user, such as the master user.**
- For example, suppose that an endpoint value is mydb.123456789012.us-east-1.rds.amazonaws.com. In this case, the port value is 3306, and the database user is admin.
- Given this information, you specify the following values in a connection string:
  - For host or **host name or** DNS name, specify mydb.123456789012.useast1.rds.amazonaws.com.
  - For **port**, specify 3306.
  - For **user**, specify admin.
- To connect to a DB instance, use any client for a DB engine. For example, you might use the mysql utility to connect to a MariaDB or MySQL DB instance.

AWS RDS - Connecting to MySql Db instance

Getting Connection Information:

Console

To find the connection information for a DB instance in the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases** to display a list of your DB instances.
3. Choose the name of the MySQL DB instance to display its details.
4. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.

Ex: From Windows command prompt, give command to get connection information:

```
aws rds describe-db-instances --filters "Name=engine,Values=mysql" --query
"*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

AWS RDS - Connecting to MySql Db instance

To connect to a DB instance using the MySQL client, enter the following command at a Windows command prompt to connect to a DB instance using the MySQL client.

- For the -h parameter, substitute the DNS name (endpoint) for your DB instance.
- For the -P parameter, substitute the port for your DB instance.
- For the -u parameter, substitute the user name of a valid database user, such as the master user. Enter the master user password when prompted.

Ex: To connect with mydb1id mysql database

```
C:\Users\bh>mysql -h mydb1id.c90djrnrrzvh.us-east-1.rds.amazonaws.com -P 3306 -u admin -p
```

Output:

```
Enter password: **********
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 193
Server version: 8.0.20 Source distribution
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MySQL [(none)]>
```

AWS RDS - Connecting to MySql Db instance

using AWS JAVA API:

AWS RDS - Connecting to MySql DB instance FROM MYSQL WORKBENCH

using AWS JAVA API:

To connect from MySQL Workbench

1. Download and install MySQL Workbench at Download MySQL Workbench
2. Open MySQL Workbench.
3. From **Database**, choose **Manage Connections**.
4. In the **Manage Server Connections** window, choose **New**.
5. In the **Connect to Database** window, enter the following information:
    - **Stored Connection** – Enter a name for the connection, such as **MyDB.**
    - **Hostname** – Enter the DB instance endpoint.
    - **Port** – Enter the port used by the DB instance.
    - **Username** – Enter the user name of a valid database user, such as the master user.
    - **Password** – Optionally, choose **Store in Vault** and then enter and save the password for the user.

6. Optionally, choose **Test Connection** to confirm that the connection to the DB instance is successful.

7. Choose **Close**.

8. From **Database**, choose **Connect to Database**.

9. From **Stored Connection**, choose your connection.

10. Choose **OK**.

AWS RDS - Data import

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/MySQL.Procedural.Importing.Other.html

- You can use several different techniques to import data into an Amazon RDS for MySQL DB instance. The best approach depends on the source of the data, the amount of data, and whether the import is done one time or is ongoing.
- If you are migrating an application along with the data, also consider the amount of downtime that you are willing to experience.
- ***mysqldump:***
- For Windows, the following command needs to be run in a command prompt that has been opened by right-clicking **Command Prompt** on the Windows programs menu and choosing **Run as administrator**:
- ***mysqldump*** *-u localuser*
- *--databases world*
- *--single-transaction*
- *--compress*
- *--order-by-primary*
- *-plocalpassword | mysql -u rdsuser*
- *--port=3306*
- *--host=myinstance.123456789012.us-east-1.rds.amazonaws.com*
- *-prdspassword*

AWS RDS - Data import

```
mysqldump -u local_user \
  --databases database_name \
  --single-transaction \
  --compress \
  --order-by-primary  \
  -plocal_password | mysql -u RDS_user \
     --port=port_number \
     --host=host_name \
     -pRDS_password
```

AWS RDS

The parameters used are as follows:

-u local_user – Use to specify a user name. In the first usage of this parameter, you specify the name of a user account on the local MySQL or MariaDB database identified by the --databases parameter.

--databases database_name – Use to specify the name of the database on the local MySQL or MariaDB instance that you want to import into Amazon RDS.

--single-transaction – Use to ensure that all of the data loaded from the local database is consistent with a single point in time. If there are other processes changing the data while mysqldump is reading it, using this option helps maintain data integrity.

--compress – Use to reduce network bandwidth consumption by compressing the data from the local database before sending it to Amazon RDS.

--order-by-primary – Use to reduce load time by sorting each table's data by its primary key.

-plocal_password – Use to specify a password. In the first usage of this parameter, you specify the password for the user account identified by the first -u parameter.

-u RDS_user – Use to specify a user name. In the second usage of this parameter, you specify the name of a user account on the default database for the MySQL or MariaDB DB instance identified by the --host parameter.

AWS RDS

--port port_number – Use to specify the port for your MySQL or MariaDB DB instance. By default, this is 3306 unless you changed the value when creating the instance.
--host host_name – Use to specify the DNS name from the Amazon RDS DB instance endpoint, for example, myinstance.123456789012.us-east-1.rds.amazonaws.com. You can find the endpoint value in the instance details in the Amazon RDS Management Console.
-pRDS_password – Use to specify a password. In the second usage of this parameter, you specify the password for the user account identified by the second -u parameter.

AWS CLI for RDS

- To find the connection information for a DB instance by using the AWS CLI, call the describe-dbinstances command. In the call, query for the DB instance ID, endpoint, port, and master user name.

- aws rds describe-db-instances --query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"

Managing RDS DB Instance:

Starting, Stopping, Rebooting, Modifying, Renaming, Deleting DB instances

AWS RDS – Stopping DB Instance

Managing an RDS DB Instance:

- *Starting, Stopping, Rebooting, Modifying, Renaming, Deleting DB instances*

Stopping an RDS DB Instance:

- If you use a DB instance intermittently, for temporary testing, or for a daily development activity, you can stop your Amazon RDS DB instance temporarily to save money.
- While your DB instance is stopped, you are charged for provisioned storage (including Provisioned IOPS) and backup storage (including manual snapshots and automated backups within your specified retention window), but not for DB instance hours.

You can stop a DB using the AWS Management Console, the AWS CLI, or the RDS API.

Limitations:

1. You can't stop a DB instance that has a **read replica**, or that is a read replica.
2. You can't stop an Amazon RDS for SQL Server DB instance in a **Multi-AZ configuration**.
3. You can't modify a stopped DB instance.
4. You can't delete an **option group** that is associated with a stopped DB instance.
5. You can't delete a DB **parameter group** that is associated with a stopped DB instance.

AWS RDS – Stopping DB Instance

Using AWS Console:

To stop a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to stop.

3. For **Actions**, choose **Stop**.

4. (Optional) In the **Stop DB Instance** window, choose **Yes** for **Create Snapshot?** and enter the

snapshot name for **Snapshot name**. Choose **Yes** if you want to create a snapshot of the DB instance before stopping it.

5. Choose **Yes, Stop Now** to stop the DB instance, or choose **Cancel** to cancel the operation.

***using RDS API*:**

To stop a DB instance by using the Amazon RDS API, call the StopDBInstance operation with the following parameter:

• DBInstanceIdentifier – the name of the DB instance.

Using AWS CLI:

aws rds **stop-db-instance** --db-instance-identifier *mydbinstance.*

Where:

DBInstanceIdentifier → the name of the DB instance.

AWS RDS – Starting DB Instance

Starting an Amazon RDS DB instance that was previously stopped:

- After you stop your DB instance, you can restart it to begin using it again.
- When you start a DB instance that you previously stopped, the DB instance retains the ID, Domain Name Server (DNS) endpoint, parameter group, security group, and option group. When you start a stopped instance, you are charged a full instance hour.

To start a DB instance using Console:

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to start.

3. For **Actions**, choose **Start**.

Using AWS CLI:

aws rds start-db-instance --db-instance-identifier *mydbinstance*

Using RDS API:

To start a DB instance by using the Amazon RDS API, call the StartDBInstance operation with the

following parameter:

• DBInstanceIdentifier – The name of the DB instance.

AWS RDS – Modifying DB Instance

Using AWS Console:

To modify a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://

console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.

3. Choose **Modify**. The **Modify DB Instance** page appears.

4. Change any of the settings that you want.

5. When all the changes are as you want them, choose **Continue** and check the summary of

modifications.

6. (Optional) Choose **Apply immediately** to apply the changes immediately. Choosing this option can cause downtime in some cases.

7. On the confirmation page, review your changes. If they are correct, choose **Modify DB Instance** to save your changes.

Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS RDS – Modifying DB Instance

Using AWS CLI:

To modify a DB instance by using the AWS CLI, call the modify-db-instance command. Specify the DB instance identifier and the values for the options that you want to modify.

Modifying an Amazon RDS DB instance:

```
aws rds modify-db-instance ^
--db-instance-identifier mydbinstance ^
--backup-retention-period 7 ^
--deletion-protection ^
--no-apply-immediately
```

Using RDS API:

- To modify a DB instance by using the Amazon RDS API, call the ModifyDBInstance operation. Specify the DB instance identifier, and the parameters for the settings that you want to modify

AWS RDS - Renaming DB Instance

Renaming DB Instance using Console:

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://

console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Databases**.

3. Choose the DB instance that you want to rename.

4. Choose **Modify**.

5. In **Settings**, enter a new name for **DB instance identifier**.

6. Choose **Continue**.

7. To apply the changes immediately, choose **Apply immediately**. Choosing this option can cause an outage in some cases.

8. On the confirmation page, review your changes. If they are correct, choose **Modify DB Instance** to save your changes.

AWS RDS - Renaming DB Instance

Renaming DB instance Using using CLI:

```
aws rds modify-db-instance ^
--db-instance-identifier DBInstanceIdentifier ^
--new-db-instance-identifier NewDBInstanceIdentifier
```

Using RDS API:

To rename a DB instance, call Amazon RDS API operation ModifyDBInstance with the following parameters:

- DBInstanceIdentifier — existing name for the instance
- NewDBInstanceIdentifier — new name for the instance

AWS RDS - Rebooting DB Instance

using Console:

To reboot a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://

console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to reboot.

3. For **Actions**, choose **Reboot**.

The **Reboot DB Instance** page appears.

4. (Optional) Choose **Reboot with failover?** to force a failover from one AZ to another.

5. Choose **Reboot** to reboot your DB instance. Alternatively, choose **Cancel**.

using CLI:

To reboot a DB instance by using the AWS CLI, call the reboot-db-instance command.

```
aws rds reboot-db-instance ^
--db-instance-identifier mydbinstance
```

Using RDS API:

To reboot a DB instance by using the Amazon RDS API, call the RebootDBInstance operation

NoSQL

No SQL

- NoSQL databases enable you to store data with flexible schema and a variety of data models. These databases are relatively easy for developers to use, and have the high performance and functionality needed for modern applications.
- NoSQL is a new breed of database management systems that fundamentally differ from relational database systems. NoSQL database is a highly scalable and flexible database management system. NoSQL database allows the user to store and process unstructured data and semi-structured data; this feature is not possible in RDBMS tools.
- NoSQL database is a type of non-relational database, and it is capable of processing structured, semi-structured and unstructured data.
- NoSQL databases, also referred to as “non-SQL” and “Not Only SQL” databases, are mainly used for unstructured data.
- **Data is not stored in tabular format** but is stored mainly **in documents, key-value pairs, graphs, or wide column stores format.** As NoSQL databases are schema agnostic, **unstructured data such as blog articles, photos, videos, or other content can be stored very easily.**
- NoSQL is an approach to database design that can accommodate a wide variety of data models, including key-value, document, columnar and graph formats.
- NoSQL systems don’t generally provide the same level of data consistency as SQL databases.
- SQL databases have traditionally sacrificed scalability and performance for the ACID properties.
- NoSQL databases guarantee high speed and scalability performance.
- NoSQL systems have the architecture in such a way to operate at high speed and wider flexibility towards the developer side.
- Some of the most common NoSQL databases are MongoDB, MarkLogic, CloudDB, and Dynamo DB.
- *Unlike SQL databases, NoSQL databases are more horizontally scalable which means that the load can be distributed by adding more database servers in the pool. This elastic scalability is a big advantage in providing optimal support for the MapReduce programming model, which makes NoSQL the perfect candidate for big-data applications.*

No SQL - Characteristics

- **Multi-Model:** This feature of NoSQL databases makes them extremely flexible when it comes to handling data.
- **Easily Scalable:** This feature of NoSQL databases easy scales to adapt to huge volumes and complexity of cloud applications. This scalability also improves performance, allowing for continuous availability and very high read/write speeds.
- **Flexible:** This feature of NoSQL databases allows you to process all varieties of data. It can process structured, semi-structured and unstructured data. It works on many processors—NoSQL systems allow you to store your database on multiple processors and maintain high-speed performance.
- **Less Downtime:** The elastic nature of NoSQL allows for the workload to automatically be spread across any number of servers.
-

No SQL

- Models of NoSQL Databases Offered on AWS
  - Key-Value Databases
  - Document Databases
  - Wide Column Databases
  - Graph Databases
  - Time Series Databases
  - Ledger Databases
- AWS NoSQL Databases Services
  - Amazon DynamoDB
  - Amazon ElastiCache
  - Amazon Neptune
  - Amazon Timestream
  - Amazon QLDB
  - Amazon DocumentDB
  - Amazon Keyspaces

SQL dbs

- SQL databases use the ACID database properties to ensure that the database transactions are reliable.
- ACID stands for
  - **Atomicity** (An “all or nothing” approach for the data that is committed to be saved),
  - **Consistency** (Interrupted changes are rolled back),
  - **Isolation** (Intermediate state of a transaction is not visible to other transactions), and
  - **Durability** (Completed transactions retain their state even in system failure).

SQL vs *No SQL*

You would choose an SQL database when:

- You need a database with a predefined schema so that applications adhere to that schema.
- You are designing an application that requires multi-row transactions.
- You require a database that has no room for error and is very consistent, for example in the case of data warehousing systems.

You would choose an NoSQL database when:

- You need a database that accounts for exponential growth with no clear schema definitions.
- You require a database which can accommodate variable data structures and plays well with big data platforms such as Hadoop.
- You need a distributed database system that scales easily and inexpensively.

No SQL

SQL	NoSQL
Relational Data Base Management System (RDBMS)	Non-relational or distributed database system. (Non – RDBMS)
These databases have fixed or static or predefined schema	They have dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows **ACID** (Automocity, Consistency, Isolation and Durability) property	Follows **CAP**(consistency, availability, partition tolerance)

SQL vs No SQL

1. Type:

- SQL databases are primarily called as Relational Databases (RDBMS); whereas
- NoSQL database are primarily called as non-relational or distributed database.

2. Language difference:

- **SQL requires you to use predefined schemas** to determine the structure of your data before you work with it. Also all of your data must follow the same structure. This can require significant up-front preparation which means that a change in the structure would be both difficult and disruptive to your whole system.
- **A NoSQL database has dynamic schema for unstructured data**. Data is stored in many ways which means it can be document-oriented, column-oriented, graph-based or organized as a KeyValue store. This flexibility means that documents can be created without having defined structure first. Also each document can have its own unique structure. The syntax varies from database to database, and you can add fields as you go.

3. The Scalability:

- In almost all situations **SQL databases are vertically scalable**. This means that you can increase the load on a single server by increasing things like RAM, CPU or SSD.
- **NoSQL databases are horizontally scalable**. This means that you handle more traffic by sharing, or adding more servers in your NoSQL database. It is similar to adding more floors to the same building versus adding more buildings to the neighbourhood. Thus NoSQL can ultimately become larger and more powerful, making these databases the preferred choice for large or ever-changing data sets.

SQL vs No SQL

4. The Structure:

- **SQL databases are table-based**
- **NoSQL databases are either key-value pairs, document-based, graph databases or wide-column stores.** This makes relational SQL databases a better option for applications that require multi-row transactions such as an accounting system or for legacy systems that were built for a relational structure.

5. Property followed:

- **SQL databases follow ACID properties** (Atomicity, Consistency, Isolation and Durability) whereas
- **NoSQL database follows the Brewers CAP theorem** (Consistency, Availability and Partition tolerance).

6. Support:

- **Great support is available for all SQL database from their vendors.** Also a lot of independent consultations are there who can help you with SQL database for a very large scale deployments
- **NoSQL database you still have to rely on community support** and only limited outside experts are available for setting up and deploying your large scale NoSQL deployments.
- Some **examples of SQL databases include PostgreSQL, MySQL, Oracle and Microsoft SQL Server.**
- **NoSQL database examples include Redis, RavenDB Cassandra, MongoDB, BigTable, HBase, Neo4j and CouchDB, DynamoDB.**

No SQL db models offered on AWS

- Models of NoSQL Databases Offered on AWS
  - Key-Value Databases
  - Document Databases
  - Wide Column Databases
  - Graph Databases
  - Time Series Databases
  - Ledger Databases

AWS *No SQL databases*

Types of No-SQL Databases offered on AWS:

There are ***six types of NoSQL dataase models*** you can choose from in AWS.

1) Key-Value Databases
2) Document Databases
3) Wide Column Databases
4) Graph Databases
5) Time Series Databases
6) Ledger Databases

Key-Value Databases:

Key-value databases enable you to store data in pairs containing a unique ID and a data value. This provides a flexible storage structure since values are not assigned to a table and can hold any amount or structure of data. These databases can manage large volumes of data or requests.

- ✓ Use cases for key-value databases include gaming applications, eCommerce systems, and high traffic applications.

AWS service: Amazon DynamoDB

No SQL

Document Databases:

Document databases are structured similarly to key-value databases except that keys and values are stored in documents written in a markup language like JSON, XML, or YAML. You can use these databases to store hierarchies of data by linking documents. Use cases for document databases include user profiles, catalogs, and content management.

AWS service: Amazon DocumentDB, DynamoDB

Wide Column Databases:

Wide column databases are based on tables but without a strict column format. Rows do not need a value in every column and segments of rows and columns containing different data formats can be combined. Use cases for wide column databases include route optimization, fleet management, and industrial maintenance applications.

AWS service: Amazon Keyspaces (for Apache Cassandra)

Graph Databases

Graph databases are structured as collections of edges and nodes. Nodes are the individual data values and edges are the relationships between those values. These databases enable you to track intricately related data in an organic network rather than a structured table. Use cases for graph databases include recommendation engines, social networking, and fraud detection.

AWS service: Amazon Neptune

No SQL

Time Series Databases:

Time series databases store data in time ordered streams. Data is not sorted by value or ID but by the time of collection, ingestion, or other timestamps included in the metadata.

These databases enable you to manage and query data based on time intervals. Use cases for time series databases include industrial telemetry, DevOps, and Internet of things (IoT) applications.

AWS service: Amazon Timestream

Ledger Databases:

Ledger databases are based on logs that record events related to data values. These logs are transparent, immutable, and can be verified cryptographically to prove the authenticity and integrity of data. Use cases for ledger databases include banking systems, registrations, supply chains, and systems of record.

AWS service: Amazon Quantum Ledger Database (QLDB)

AWS *No SQL DB Services*

- AWS NoSQL Databases Services
  - Amazon DynamoDB
  - Amazon ElastiCache
  - Amazon Neptune
  - Amazon Timestream
  - Amazon QLDB
  - Amazon DocumentDB
  - Amazon Keyspaces

AWS No SQL db services

Amazon DynamoDB

Amazon DynamoDB is a document and key-value database. It is a fully managed service that includes features for backup and restore, in-memory caching, security, and multiregion, multimaster distribution. DynamoDB supports atomicity, consistency, isolation, durability (ACID) transactions and encryption by default.

Amazon ElastiCache

Amazon ElastiCache is an in-memory data store that you can use in place of a disk-based database. It provides fully managed support for Memcached and Redis, and enables scaling with memory sharding. It is designed to support sub-millisecond response times and is typically used for queuing, real-time analytics, caching, and session stores.

Amazon Neptune

Amazon Neptune is a graph database service that is fully managed and optimized for storing data on billions of relationships. It supports a range of graph models and query languages, including W3C's RDF, Property Graph, SPARQL, and TinkerPop Gremlin.

Neptune includes features for point-in-time recovery, multi-zone data replication, continuous backups, and read replicas. It supports ACID transactions and provides encryption in-transit and at-rest.

Amazon Timestream

Amazon Timestream is a fully managed time series database with an adaptive query processing engine. It is a serverless service and automatically manages hardware and software maintenance and provisioning for you.

Timestream includes features for automated data compression, tiering, retention, and rollups. It also includes built-in analytics for the approximation, smoothing, and interpolation of data.

AWS No SQL db services

Amazon QLDB

Amazon QLDB is a ledger database that you can use to track data changes. It is fully managed and designed to enable you to avoid complex setups required for managing ledger data with relational databases or blockchain.

QLDB provides a SQL-like API, full transactional support, and a flexible document data model. It includes features for automatic scaling, ACID compliant transactions, multizone availability, and data streaming with Kinesis Data Streams.

Amazon DocumentDB

Amazon DocumentDB is a fully managed document database that is compatible with MongoDB. DocumentDB architecture separates compute and storage resources for greater scalability and flexibility. It also includes support for up to 15 read replicas, data replication for durability across three availability zones, and free use of the AWS Database Migration Service.

Amazon Keyspaces

Amazon Keyspaces is a managed wide column database that is compatible with Apache Cassandra. You can use it to migrate Cassandra workloads and applications and continue to use Cassandra native code and tools. It includes features for autoscaling and enables you to select between on-demand or provisioned resources.

Amazon DynamoDB

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.** DynamoDB also offers **encryption**.
- With DynamoDB, you can create database tables that can store and retrieve **any amount of data and serve any level of request traffic.**
- You can use the AWS Management Console **to monitor resource utilization and performance metrics.**
- DynamoDB provides **on-demand backup** capability. It allows you to create full backups of your tables for long-term retention and archival for regulatory compliance needs.
- You can create on-demand backups and enable **point-in-time recovery** for your Amazon DynamoDB tables. Point-in-time recovery helps protect your tables from accidental write or delete operations. With point-in-time recovery, you can restore a table to any point in time during the last 35 days.
- DynamoDB allows you to delete expired items from tables automatically to help you reduce storage usage and the cost of storing data that is no longer relevant.

No SQL

High Availability and Durability:

- DynamoDB automatically spreads the data and traffic for your tables over a sufficient number of servers to handle your throughput and storage requirements, while maintaining consistent and fast performance.
- All of your data is stored on solid-state disks (SSDs) and is automatically replicated across multiple Availability Zones in an AWS Region, providing built-in high availability and data durability.
- You can use global tables to keep DynamoDB tables in sync across AWS Regions.

Dynamo DB

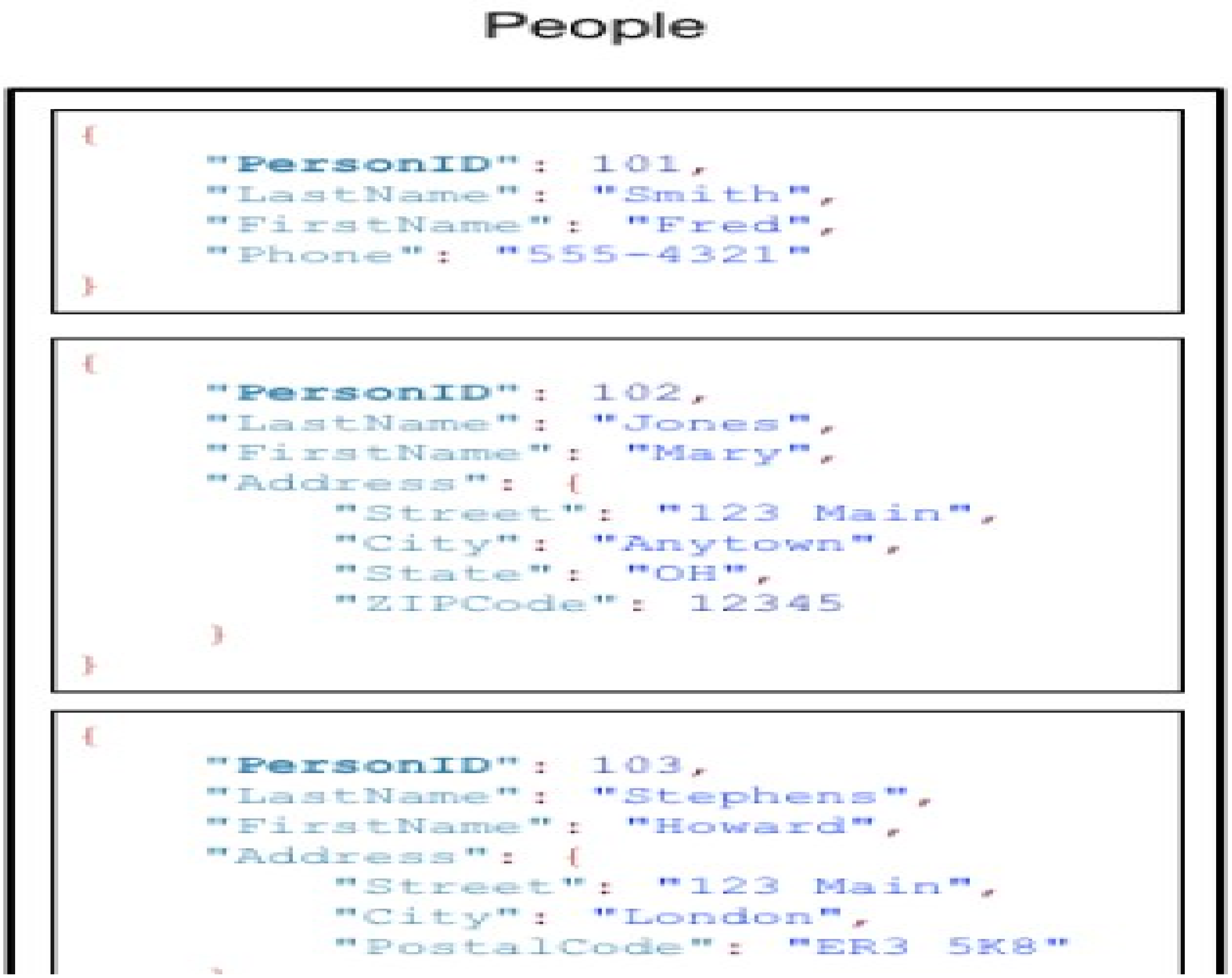
- In DynamoDB, tables, items, and attributes
  - are the core components that you work with.
- A *table* is a collection of *items*, and each item is a collection of *attributes*.
- DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility.

Tables, Items, and Attributes:

The following are the basic DynamoDB components:

- **Tables** – Similar to other database systems, **DynamoDB stores data in tables.** A *table* is a collection of data. For example, see the example table called *People* that you could use to store personal contact information about friends, family, or anyone else of interest. You could also have a *Cars* table to store information about vehicles that people drive.
- **Items** – Each table contains zero or more items. An *item* is a group of attributes that is uniquely identifiable among all of the other items. In a *People* table, each item represents a person. For a *Cars* table, each item represents one vehicle. Items in DynamoDB are similar in many ways to rows, records, or tuples in other database systems. In DynamoDB, there is no limit to the number of items you can store in a table.
- **Attributes** – Each item is composed of one or more attributes. An *attribute* is a fundamental data element, something that does not need to be broken down any further. For example, an item in a *People* table contains attributes called *PersonID*, *LastName*, *FirstName*, and so on. For a *Department* table, an item might have attributes such as *DepartmentID*, *Name*, *Manager*, and so on. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems.

Dynamo DB



People

```
{
      "PersonID": 101,
      "LastName": "Smith",
      "FirstName": "Fred",
      "Phone": "555-4321"
}
```

```
{
      "PersonID": 102,
      "LastName": "Jones",
      "FirstName": "Mary",
      "Address": {
            "Street": "123 Main",
            "City": "Anytown",
            "State": "OH",
            "ZIPCode": 12345
      }
}
```

```
{
      "PersonID": 103,
      "LastName": "Stephens",
      "FirstName": "Howard",
      "Address": {
            "Street": "123 Main",
            "City": "London",
            "PostalCode": "ER3 5K8"
```

Dynamo DB

Points:

- Note the following about the *People* table:
- Each item in the table has a unique identifier, or primary key, that distinguishes the item from all of the others in the table. In the *People* table, the primary key consists of one attribute (*PersonID*).
- Other than the primary key, the *People* table is schemaless, which means that neither the attributes nor their data types need to be defined beforehand. Each item can have its own distinct attributes.
- Most of the attributes are *scalar*, which means that they can have only one value. Strings and numbers are common examples of scalars.
- Some of the items have a nested attribute (*Address*). DynamoDB supports nested attributes up to 32 levels deep.

Dynamo DB

Primary Key:

- When you create a table, in addition to the table name, you must specify the primary key of the table.
- The primary key uniquely identifies each item in the table, so that no two items can have the same key.

DynamoDB supports two different kinds of primary keys:

Partition key – A simple primary key, composed of one attribute known as the *partition key*.

DynamoDB uses the partition key's value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored. In a table that has only a partition key, no two items can have the same partition key value.

Partition key and sort key – Referred to as a ***composite primary key*, this type of key is composed of**

two attributes. The first attribute is the *partition key*, and the second attribute is the *sort key*.

DynamoDB uses the partition key value as input to an internal hash function. The output from the hash function determines the partition (physical storage internal to DynamoDB) in which the item will be stored. All items with the same partition key value are stored together, in sorted order by sort key value.

In a table that has a partition key and a sort key, it's possible for two items to have the same partition key value. However, those two items must have different sort key values.