Lesson 8 RDS

Michael Yang

RDB and RDS

- Relational databases focus on data consistency and guarantee ACID (atomicity, consistency, isolation, and durability) database transactions. A typical task is storing and querying structured data, such as the accounts and transactions in an accounting application.
- The Amazon Relational Database Service (Amazon RDS) offers ready-to-use relational databases such as PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server.
- RDS is a managed service. The managed service provider—in this case, AWS—is responsible for providing a defined set of services—in this case, operating a relational database system.

RDS vs DB on EC2

Managed service RDS vs. a self-hosted database on virtual machines

	Amazon RDS	Self-hosted on virtual machines
Cost for AWS services	Higher because RDS costs more than virtual machines (EC2)	Lower because virtual machines (EC2) are cheaper than RDS
Total cost of ownership	Lower because operating costs are split among many customers	Much higher because you need your own manpower to manage your database
Quality	AWS professionals are responsible for the managed service.	You'll need to build a team of professionals and implement quality control yourself.
Flexibility	High, because you can choose a relational database system and most of the configuration parameters	Higher, because you can control every part of the relational database system you installed on virtual machines

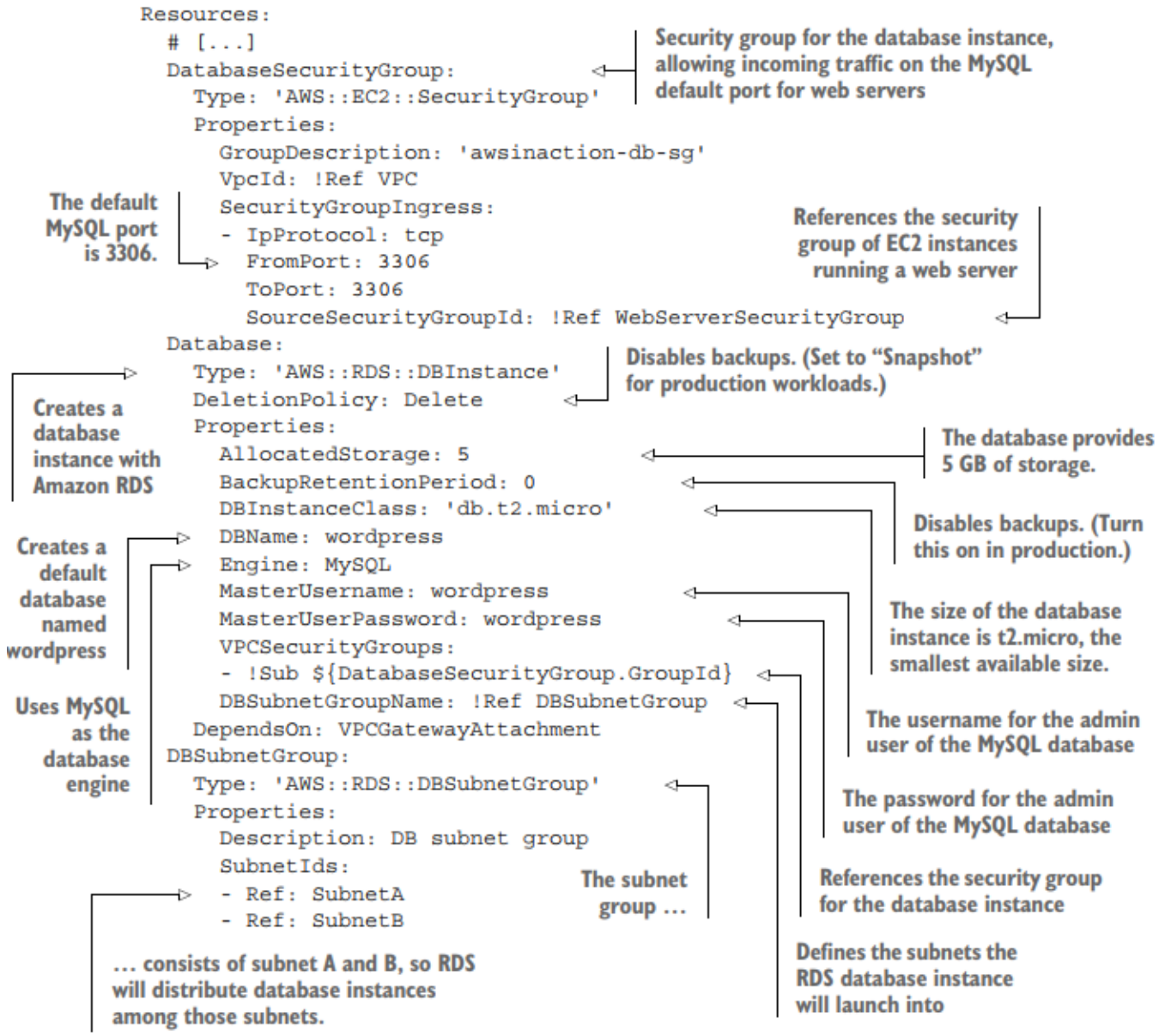
Launch RDS

- Launching a database consists of two steps:
  1 Launching a database instance
  2 Connecting an application to the database endpoint

- Attribute needed to connect to an RDS database

Attribute	Description
`AllocatedStorage`	Storage size of your database in GB
`DBInstanceClass`	Size (also known as instance type) of the underlying virtual machine
`Engine`	Database engine (Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, or Microsoft SQL Server) you want to use
`DBName`	Identifier for the database
`MasterUsername`	Name for the admin user
`MasterUserPassword`	Password for the admin user

CloudFormation Template



- Databases on Amazon RDS are priced according to the size of the underlying virtual machine and the amount and type of allocated storage.

Monthly costs for a medium-sized RDS instance

Description	Monthly price
Database instance `db.t4g.medium`	$94.17 USD
50 GB of general purpose (SSD)	$11.50 USD
Additional storage for database snapshots (100 GB)	$9.50 USD
Total	$115.17 USD

Importing Data

- A database without data isn't useful. In many cases, you'll need to import data into a new database by importing a dump from the old database. The process is similar for all other database engines (Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server). Here we learn through the process of importing a MySQL database dump to an RDS database with a MySQL engine.

Importing Data

- To import a database from your on-premises environment to Amazon RDS, follow these steps:

1. Export the database. (using mysqldump)
2. Start a virtual machine in the same region and VPC as the RDS database.
3. Upload the database dump to the virtual machine.
4. Run an import of the database dump to the RDS database on the virtual server.

Importing Data

- Theoretically, you could import a database to RDS from any machine from your on-premises or local network, but the higher latency over the internet or VPN connection will slow down the import process dramatically. Because of this, we recommend adding a second step: upload the database dump to a virtual machine running in the same AWS region and VPC, and import the database into RDS from there.

Importing Data

- To do so, we'll guide you through the following steps:

1 Connect to the virtual machine that is running.
2 Download a database dump from S3 to the VM. (If you are using your own database dump, we recommend uploading it to S3 first.)
3 Import the database dump into the RDS database from the virtual machine.

Backup and Restore

- How to use RDS snapshots to do backup and restore
  1. Configuring the retention period and time frame for automated snapshots
  2. Creating snapshots manually
  3. Restoring snapshots by starting new database instances based on a snapshot
  4. Copying a snapshot to another region for disaster recovery or relocation

Configure Automated Snapshot

- BackupRetentionPeriod
- Automated snapshots are created once a day during the specified time frame. If no time frame is specified, RDS picks a random 30-minute time frame during the night. A new random time frame will be chosen each night.

Configure Automated Snapshot

```
aws cloudformation update-stack --stack-name wordpress --template-url \
 https://s3.amazonaws.com/awsinaction-code3/chapter10/\
 template-snapshot.yaml \
 --parameters ParameterKey=WordpressAdminPassword,UsePreviousValue=true \
 --capabilities CAPABILITY_IAM
```

Configure Automated Snapshot

```
Database:
  Type: 'AWS::RDS::DBInstance'
  DeletionPolicy: Delete
  Properties:
    AllocatedStorage: 5
    BackupRetentionPeriod: 3
    PreferredBackupWindow: '05:00-06:00'
    DBInstanceClass: 'db.t2.micro'
    DBName: wordpress
    Engine: MySQL
    MasterUsername: wordpress
    MasterUserPassword: wordpress
    VPCSecurityGroups:
    - !Sub ${DatabaseSecurityGroup.GroupId}
    DBSubnetGroupName: !Ref DBSubnetGroup
  DependsOn: VPCGatewayAttachment
```

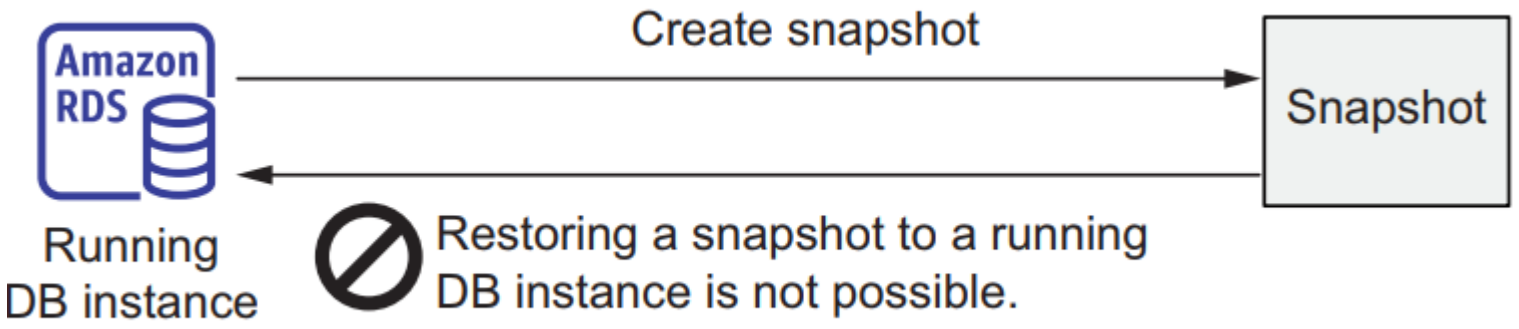
Keeps snapshots for three days

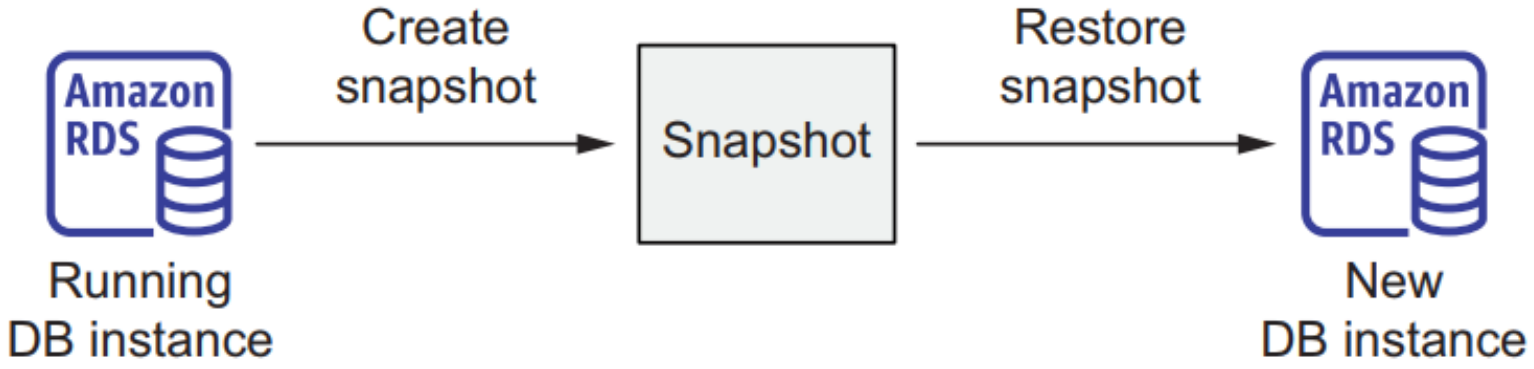
Creates snapshots automatically between 05:00 and 06:00 UTC

Backup Manually

- To create a snapshot, you have to know the instance identifier. The following command extracts the instance identifier from the first RDS database instance:
  `$ aws rds describe-db-instances --output text --query "DBInstances[0].DBInstanceIdentifier"`
- The next command creates a manual snapshot called `wordpress-manual-snapshot`. Replace `$DBInstanceIdentifier` with the output of the previous command:
  `$ aws rds create-db-snapshot --db-snapshot-identifier wordpress-manual-snapshot --db-instance-identifier $DBInstanceIdentifier`

Restoring Database





Restoring Database

- To create a new database in the same VPC as the WordPress platform you started in, you need to find out the existing database's subnet group.

```
$ aws cloudformation describe-stack-resource \
➥ --stack-name wordpress --logical-resource-id DBSubnetGroup \
➥ --query "StackResourceDetail.PhysicalResourceId" --output text
```

- You're now ready to create a new database based on the manual snapshot you created at the beginning of this section. Execute the following command, replacing `$SubnetGroup` with the output of the previous command:

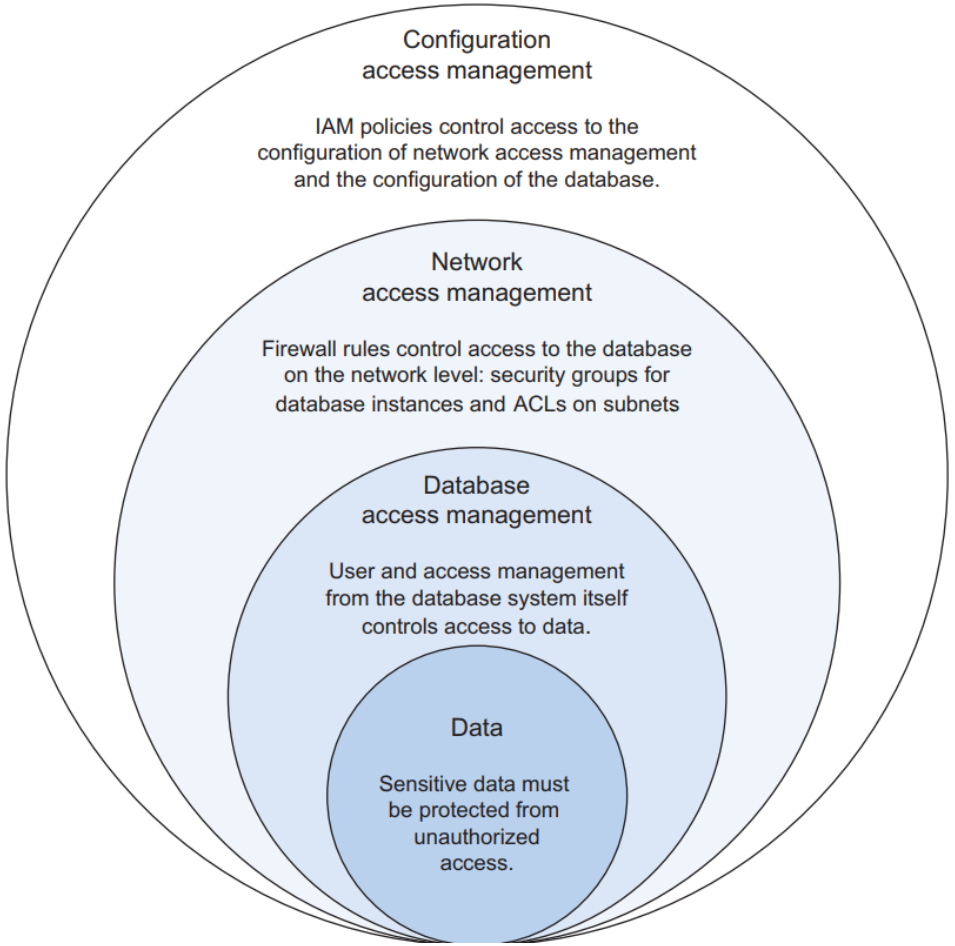
```
$ aws rds restore-db-instance-from-db-snapshot \
➥ --db-instance-identifier awsinaction-db-restore \
➥ --db-snapshot-identifier wordpress-manual-snapshot \
➥ --db-subnet-group-name $SubnetGroup
```

Restoring Automated Snapshot

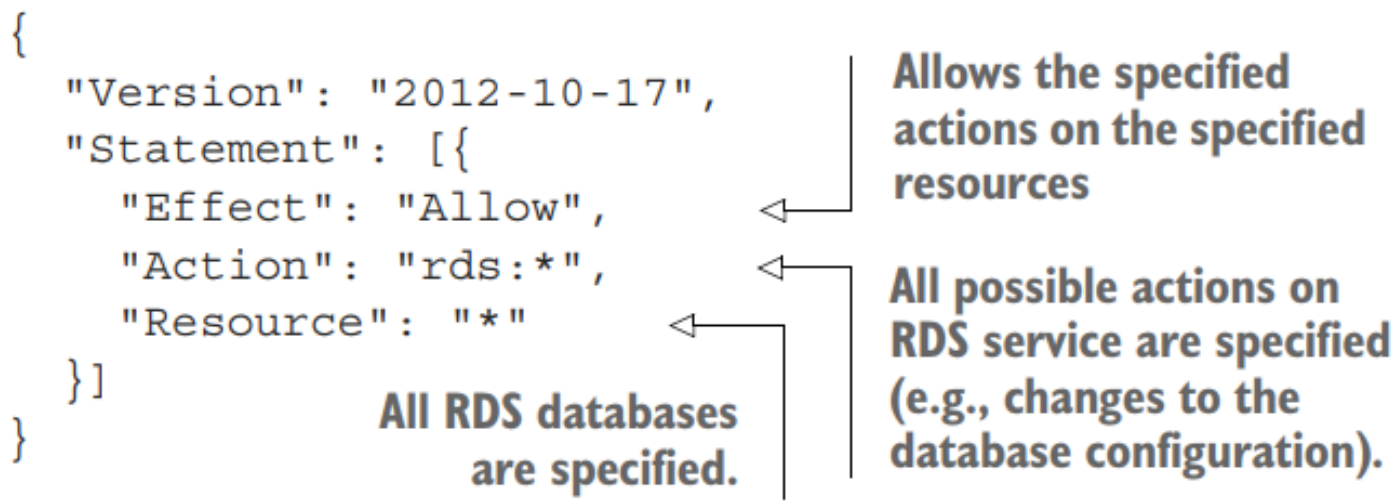
- If you're using automated snapshots, you can also restore your database from a specified moment, because RDS keeps the database's change logs. This allows you to jump back to any point in time from the backup retention period to the last five minutes.

```
$ aws rds restore-db-instance-to-point-in-time \
➥ --target-db-instance-identifier awsinaction-db-restore-time \
➥ --source-db-instance-identifier $DBInstanceIdentifier \
➥ --restore-time $Time --db-subnet-group-name $SubnetGroup
```

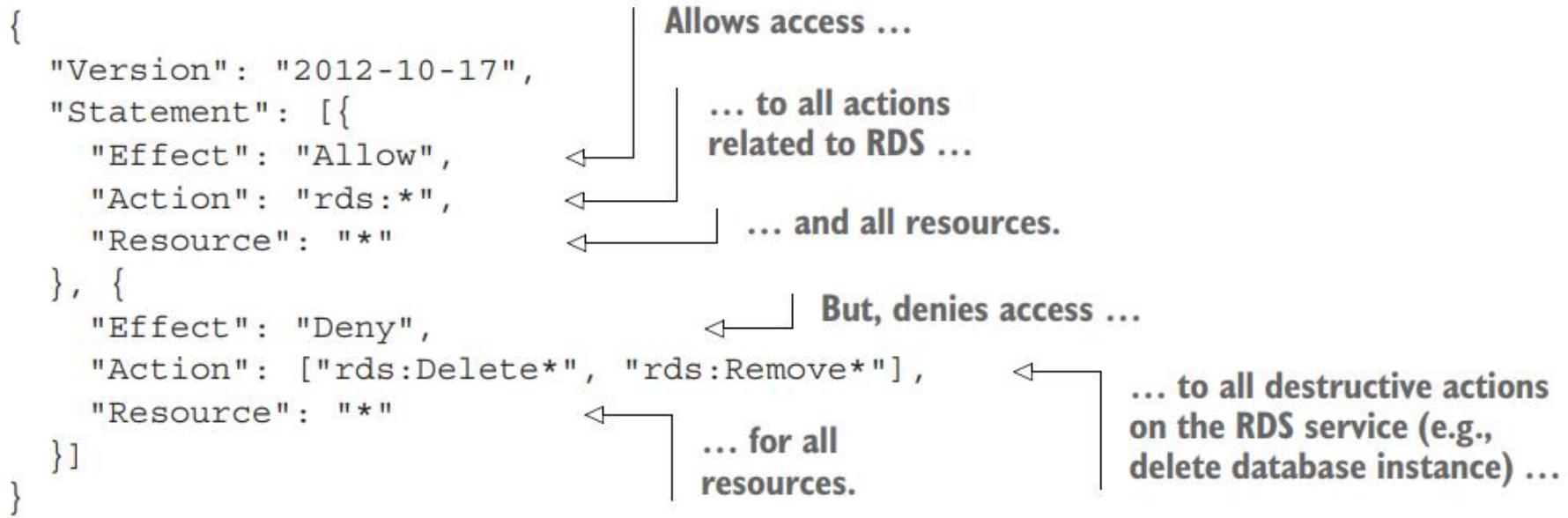
Controlling Access to RDS



Controlling Persmissons



Controlling Permissions



Controlling Network

```
DatabaseSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'awsinaction-db-sg'
    VpcId: !Ref VPC
    SecurityGroupIngress:
    - IpProtocol: tcp
      FromPort: 3306
      ToPort: 3306
      SourceSecurityGroupId: !Ref WebServerSecurityGroup
```

The security group for the database instance, allowing incoming traffic on the MySQL default port for web servers

The default MySQL port is 3306.

References the security group for web servers

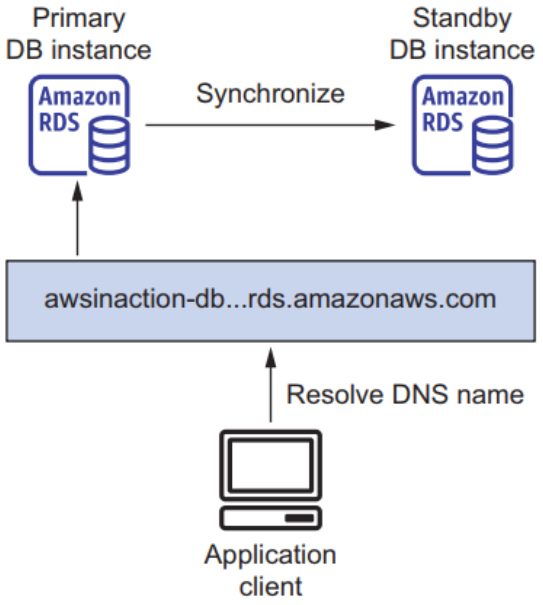
Controlling Data Access

- A database engine also implements access control itself. User management of the database engine has nothing to do with IAM users and access rights; it's only responsible for controlling access to the database. For example, you typically define a user for each application and grant rights to access and manipulate tables as needed. In the WordPress example, a database user called wordpress is created. The WordPress application authenticates itself to the database engine (MySQL, in this case) with this database user and a password.

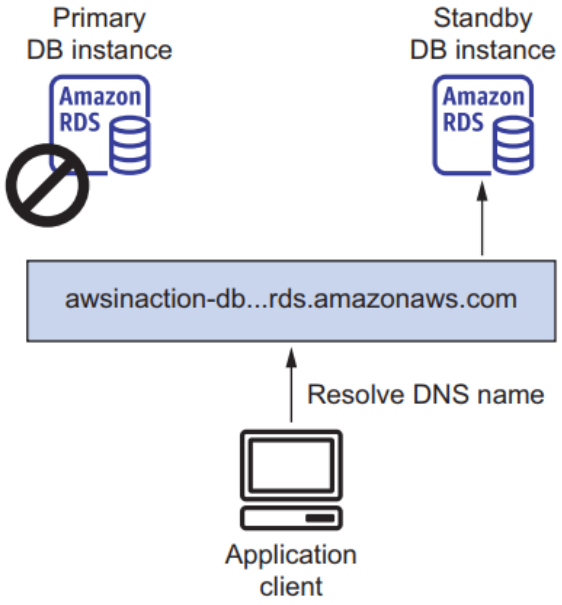
Controlling Data Access

- Typical use cases follow:
  1Limiting write access to a few database users (e.g., only for an application)
  2 Limiting access to specific tables to a few users (e.g., to one department in the organization)
  3 Limiting access to tables to isolate different applications (e.g., hosting multiple applications for different customers on the same database)

HA Database



The primary database is replicated to the standby database when running in high-availability mode.



The client fails over to the standby database if the primary database fails, using DNS resolution.

Aurora

- Aurora is an exception to the way that highly available databases operate in AWS. It does not store your data on a single EBS volume. Instead, Aurora stores data on a cluster volume. A cluster volume consists of multiple disks, with each disk having a copy of the cluster data. This implies that the storage layer of Aurora is not a single point of failure. But still, only the primary Aurora database instance accepts write requests. If the primary goes down, it is automatically re-created, which typically takes less than 10 minutes. If you have replica instances in your Aurora cluster, a replica is promoted to be the new primary instance, which usually takes around one minute and is much faster than primary re-creation.