AWS RDS

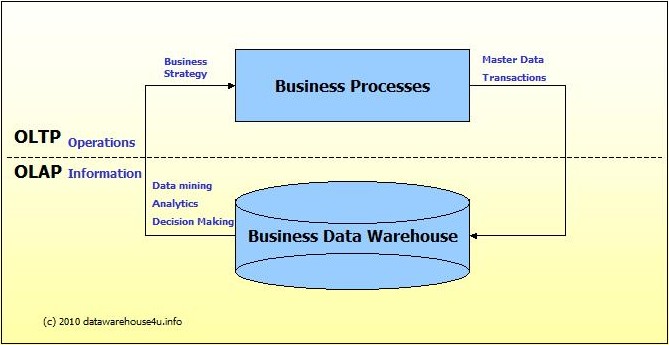
Capgemini | [Company address]

Study

Khan, Viquar

2017

**OLTP vs. OLAP**  
We can divide IT systems into transactional (OLTP) and analytical (OLAP). In general we can assume that OLTP systems provide source data to data warehouses, whereas OLAP systems help to analyze it. 



- **OLTP (On-line Transaction Processing)** is characterized by a large number of short on-line transactions (INSERT, UPDATE, DELETE). The main emphasis for OLTP systems is put on very fast query processing, maintaining data integrity in multi-access environments and an effectiveness measured by number of transactions per second. In OLTP database there is detailed and current data, and schema used to store transactional databases is the entity model (usually 3NF).   
  
- **OLAP (On-line Analytical Processing)** is characterized by relatively low volume of transactions. Queries are often very complex and involve aggregations. For OLAP systems a response time is an effectiveness measure. OLAP applications are widely used by Data Mining techniques. In OLAP database there is aggregated, historical data, stored in multi-dimensional schemas (usually star schema).

|  |  |  |
| --- | --- | --- |
|  | **OLTP System  Online Transaction Processing  (Operational System)** | **OLAP System  Online Analytical Processing  (Data Warehouse)** |
| Source of data | Operational data; OLTPs are the original source of the data. | Consolidation data; OLAP data comes from the various OLTP Databases |
| Purpose of data | To control and run fundamental business tasks | To help with planning, problem solving, and decision support |
| What the data | Reveals a snapshot of ongoing business processes | Multi-dimensional views of various kinds of business activities |
| Inserts and Updates | Short and fast inserts and updates initiated by end users | Periodic long-running batch jobs refresh the data |
| Queries | Relatively standardized and simple queries Returning relatively few records | Often complex queries involving aggregations |
| Processing Speed | Typically very fast | Depends on the amount of data involved; batch data refreshes and complex queries may take many hours; query speed can be improved by creating indexes |
| Space Requirements | Can be relatively small if historical data is archived | Larger due to the existence of aggregation structures and history data; requires more indexes than OLTP |
| Database Design | Highly normalized with many tables | Typically de-normalized with fewer tables; use of star and/or snowflake schemas |
| Backup and Recovery | Backup religiously; operational data is critical to run the business, data loss is likely to entail significant monetary loss and legal liability | Instead of regular backups, some environments may consider simply reloading the OLTP data as a recovery method |

## **General**

* Amazon RDS (Relational Database Service) helps users easily set up, operate, and then scale relational databases in a cloud.  With RDS, you have access to database capabilities which work seamlessly with existing databases. RDS patches and backs up database and stores backups for a period defined by the user. RDS gives the user flexibility to scale resources or storage capacity easily with an API call or the AWS management console. RDS also provides easy replication so that users can improve database availability, enhance data durability, and scale beyond constraints of the database. No up-front fees are required and users pay for resources used.
* A database (DB) instance is basically an environment within the cloud; you can specify the compute and storage resources you need. You can manage DB instances via **RDS APIs, command lines, and the AWS management console**. Multiple instances can be run simultaneously; additionally, each instance is able to support multiple databases or schemas, all dependent on the engine type.
* RDS manages work required in setting up a database. Once running, RDS is able to manage common administrative tasks automatically. Additionally, RDS can manage data replication and automatic failover for Multi-AZ deployments. The user must manage the settings specific to applications. Also, users must create the relational schema for the use case and are responsible for performance turning.
* Many database options are available for developers. RDS runs full relational databases and offloads administration.**Amazon DynamoDB is a NoSQL database** offering fast, consistent performance and scalability. Developers also have the option of relational database AMIs in EC2 to operate databases.
* In order to sign up for RDS, developers must have or create an Amazon Web Services account. Once signed up, there is documentation and a getting started guide. Launching a DB instance takes just minutes with either AWS management console or RDS APIs.
* To start a DB instances with AWS management console, users would choice RDS followed by Launch DB Instance under the Instances tab. This allows the user to identify parameters. Users can modify the instance’s scheduled maintenance window, backup retention policy, and preferred backup window. Users can also use **CreateDBInstance** API or the **rds-create-db-instance** command line to launch the instance.
* Available DB instances can be retrieved with the AWS management console or **DescribeDBInstance** API. This will allow the user to create the connection string needed to connect the instance with the database tool or programming language desired. You must authorize network access for the DB instance.
* Forty RDS DB instances are allowed. Under license included model, ten can be Oracle or SQL instances. All forty are allowable for PostgreSQL, Oracle SQL Server, MariaDB, MySQL, and Aurora under BYOL model (**Bring Your Own License** (**BYOL**) **Model AWS)**. For applications needing more than the default instances, users can request more instances.
* There are no limits imposed for the following: Amazon Aurora, MySQL, MariaDB, and PostgreSQL. **Oracle is allowed 1 database per instance** but have no limits on schemas per database. **SQL Server are allowed 30 databases per instance**.
* Users have many ways to import data. For MySQL use **mysqldump** or **mysqlimport**. Data pump, SQL loader, or import/export can be used for Oracle. For SQL server, use import/export wizard or BCP. PostgreSQL users have pg\_dump.
* The following engines are supported: MariaDB 10.0 and 10.1, MySQL 5.5-5.7 with InnoDB, 2008 R2, SQL SP2 and SP4, Oracle Database 11gR2 and 12c, and PostgreSQL 93.-9.5. MySQL, Oracle, and SQL users have engine version management to control various options of the DB instance.
* Maintenance windows allow the user to control when modifications and software patching takes place. Maintenance for a given week will be completed at some point in the window and will last 30 minutes.

Scale compute operations and software patching require the instance to be offline. Needed patching is scheduled automatically and only when security or durability related patches are required. This patching happens very rarely and shouldn’t require much of your maintenance window. Users without a maintenance window selected will be assigned a default value of 30 minutes. Maintenance time can be modified via the AWS management console or the **ModifyDBInstance API**. You can set maintenance windows differently for each DB instance. You can also run a multi-AZ deployment to decrease maintenance event impact.

* Amazon RDS has guidelines for the support of new engine versions and elimination of engines currently supported. New engines supported will depend on frequency and content of patches and releases from the engine vendor. Also, the database engineering team works to vet new releases and patches. The outcomes assist in adding new engine releases. RDS plans to support major releases for at least 3 years and minor releases for at least 1 year. Occasionally, versions may be deprecated. When this happens, Amazon RDS will allow for a grace period of three months after the deprecation has been announced. **Un-upgraded instances will automatically be upgraded once the grade period ends.**
* If queries run slow, users have options depending on the engine they use. MySQL keeps a slow query log which users can set the parameters on. Oracle has a trace file to determine slow queries. On the client side of SQL Server, you can trace to find slow queries. **CloudWatch** also provides metrics for CPU utilization users can check. When seeing high levels of utilization, query performance can be affected and you should consider scaling the instance class.
* Costs for each database engine vary. Costs for database engines include operational components besides licensing. Amazon RDS works hard to decrease cost and save customers money.

## **Billing**

* Amazon RDS only bills for resources used. No minimum fee or setup fee is required. Fees per month are based on: DB instance hours, GB storage, I/O requests, and provisioned IOPS. There are also fees for backup storage and data transfer.
* Billing for DB instances begin as soon the instance is made available and continues until the instance is terminated.
* DB Instance hours are billed per hour the instance is in an available running state. To avoid billing, terminate the DB instance.
* The storage allowed for DB instances primary data exists within a single zone and is replicated across available zones to guarantee greater data durability. Any backup storage beyond the free allocation to account for the replication is subject to extra fees.
* **Multi-AZ DB** Instance deployments are billed based on the multi-AZ DB instance hours, provisioned storage, I/O requests, backup storage, and data transfer.
* Prices for DB Instances and Multi-AZ DB Instance hours do not include taxes. Japanese billing address which use Asia Pacific Region are subject to the Japanese Consumption Tax

## **Free Tier**

* Amazon RDS offers an AWS Free Tier. This allows users with Single-AZ Micro DB instances through MySQL, MariaDB, PostgreSQL, Oracle BYOL, and SQL Server Express. **Free tier usage is allowed 750 instance hours each month**. Also, customers will get 20 GB of database and backup storage and 10 million I/Os free each month.
* For the first 12 months of new AWS accounts, users have AWS free tier access.
* Multiple Single-AZ Micro DB instances are allowed and eligible for usage provided with free tier. Instances over 750 are billed at the standard rate.
* Instances under the free tier are allowed only on one engine and cannot be shared across engines.
* The Free Tier offer is available for users in all regions but GovCloud in the US.

## **Reserved Instances**

* Reserved Instances allow users to reserve predicted capacity and discount the hourly charge for instances with the same configuration as the RI. Users have three payment choices: **No Upfront, Partial Upfront, or All Upfront.**
* RIs (Reserved Instances) work the same way as On-Demand Instances; they are billed differently though. RIs reserve capacity in a datacenter and provide a lower hourly rate during the term.
* RIs can be purchased via the AWS Management Console (Purchase Reserved DB Instances) or API tools (**PurchaseReservedDBInstancesOffering**). Creating a RI works the same way as launching an On-Demand Instance. Users have the options of using **rds-create-db-instance** command, **CreateDBInstance API**, or the Launch DB Instance option in AWS Management Console. If the purchase is successful, the discount will automatically apply to the hourly fee for eligible DB Instances.
* RIs are always available to be purchased even if capacity is limited in a zone. The reservation can be used in different zones within the same region.
* **Users are allowed 40 RIs**. Running more than this require the user to submit a request.
* If you would like to convert an existing DB instance to a RI, you can purchase a DB Instance reservation. You must ensure the class, engine and license model, and region are the same. Once purchased, the discount applies to existing DB Instance.
* Once a request for a RI is received, pricing changes go into effect while payment is processed. Status of RI purchased can be viewed on the AWS Account Activity page or the **DescribeReservedDBInstances API**. One-time payments not successfully authorized by the following billing period will not benefit from the discounted price. After the term ends, the RI will return to the appropriate On-demand usage fee for the class and region.
* **The system automatically applies the lowest eligible hourly rate.**
* Reservations are associated with the DB engine, instance class, deployment type, license model, and region. Reservations only apply to DB instances that contain the same attributes. Any modifications affect the hourly rates applied to running DB instances with the same attributes.
* **RIs are linked with a single region but can be used across zones within the region.**
* RIs can be purchased for multi-AZ deployments. When viewing RI offerings, look for the multi-AZ options. You can also find RIs with synchronous replication across multiple zones in the offerings.
* A DB Instance RI can apply to a Read Replica if it is the same class and in the same region. The system applies the appropriate billing fee.
* **RIs are not able to be cancelled and the fee is non-refundable**.
* Upfront payments require the entire fee for the term at the time of purchase. No upfront requires no startup fee; the fee will be spread across all the hours of the term even if no activity exists. Partial upfront requires a small upfront fee and the rest of the fees are spread across the hours of the term even is no activity exists.

## **Hardware and Scaling**

* Assessing compute, memory, and storage needs help users determine which DB Instance class and storage capacity to choose.
* Users can scale resources via **ModifyDBInstance AP**I or the AWS Management Console. Changing DB Instance class modifies memory and CPU resources. Storage available is modified through the storage allocation. These changes will take place during the selected maintenance window. You also have the option to apply immediately with that flag. All other pending changes will be applied when using this flag as well. **Monitoring utilization can be done via CloudWatch**. **RDS does not support increasing storage for SQL Server instances.**
* RDS uses EBS for storage. RDS automatically stores data across multiple volumes if needed for enhanced performance. For those using MySQL and Oracle, scaling up storage for an existing DB Instance can improve I/O capacity. Scaling storage capacity can be done via the AWS Management Console, an **rds-modify-db-instance** command, or the **ModifyDBInstance** API. SQL Server storage increase is not supported.
* When increasing storage of an available DB instance, the database will be unavailable for a few minutes while the class is modified which will happen during the maintenance window unless the user applies the change immediately.
* **If you need to scale a DB Instance beyond the maximum allowed capacity, you can use partitioning to spread the data across multiple DB Instances.**
* SSD Storage works for a broad number of database workloads needing moderate I/O requirements. **SSD storage has a baseline of 3 IOPS/GB and can burst up to 3000 IOPS**. It will give predictable performance for most applications.
* Provisioned IOPS (SSD) Storage delivers predictable, consistent, and fast I/O performance. If using this storage, the user chooses the IOPS rate when creating an instance. That rate is provisioned for the life of the instance. This storage is best suited for I/O-intensive, OLTP workloads.
* Magnetic storage is best for small workloads which access data infrequently.

## **Automated Backups and Database Snapshots**

* You can back up your instance with automated backups or DB snapshots. Automated backup enables recovery of the instance at a certain point of time. When a user enables automated backups, the system performs a fully snapshot of the data and obtains the transaction logs. Any time a recovery is initiated, the system applies transaction logs to the appropriate daily backup to restore instances to the time requested.
* Backups are kept for a user selected period which is referred to as the retention period. **The default is a single day but can be as long as 35 days**. Point in time restorations can be set during the retention period. The **DescribeDBInstance** API allows the user to restore the DB instance time, usually in the last five minutes. Last restore time can be viewed in the AWS Management Console under the description tab. On the other hand, **DB Snapshots are initiated by the user and allows you to back up your instance as frequently as you like and restore the instance to a specific state any time. DB Snapshots are created in the AWS** Management Console or with **CreateDBSnapshot** API. DB Snapshots are kept unless you delete them. Users can copy the snapshot to bring back functionality. They are denoted as automated snapshot type. You can also use the Snapshot Created Time field to find out at which time the snapshot was taken (time is presented in UTC).
* RDS provides free automated backups with a single retention period. The storage for free backup is limited to the size of the user’s provisioned database and only includes active instances. Increasing the retention period can be done with the **CreateDBInstance** API or **ModifyDBInstance** API, depending if the instance exists or not.
* Users can define a period preferred during which instances are backed up. Using these backups and transaction logs allows users to restore instances during the retention period up until the **LatestRestorableTime**, the last few minutes of the period. Storage I/O may be temporarily paused during the backup window for the backup to initialize which may result in elevated latency briefly. No I/O suspension occurs for multi-AZ DB deployments.
* S3 stores DB snapshots and automated backups. The AWS management console or **ModifyDBInstance** API can be used to modify the time period for backups to be retained. Although not recommended, setting the retention period to 0 would turn off backups. DB Snapshots can be managed through DB Snapshots section in the AWS Management Console. A list of user-created snapshots for an instance can be view using **DescribeDBSnapshots** API and snapshots can be deleted with **DeleteDBSnapshots** API.
* Before deleting an instance, users can create a last snapshot. Doing this will allow the user to restore the deleted instance later. RDS will keep all user-created snapshots even after the instance is deleted. However, when an instance is deleted, the automated backup is also deleted.

## **Security**

* Amazon Virtual Private Cloud (VPC) allows users to create virtual networking environments within a private section of the cloud. You have complete control over the aspects of the virtual network including IP address ranges, subnets, network gateways, and routing tables. Users are able to define the network topology and modify network setting to act like a traditional IP network.
* RDS works the same despite using VPC or the EC2-Classic platform. RDS manages the features within a VPC or classic platform. The difference is that instances deployed in a classic platform get an external IP address and instances run within a VPC will receive a private IP address. VPC can be configured to make an instance publicly accessible.
* If using a VPC, you may need a **DB Subnet group**. This is a group of subnets you can designate for instances in a VPC. DB subnet groups require a minimum of one subnet for each zone in a region. When creating a new instance, you have to choose a DB subnet group. RDS will then use the group and your preferred zone to select the subnet and IP address for the instance. RDS will associate an Elastic Network Interface with the instance assigned the IP address. If you create Multi-AZ deployments, you can define a subnet for all zones in a region. This will let RDS create a standby in another zone if needed. This must be done for single-AZ deployments as well in case you need to convert those to multi-AZ deployments later.
* To create an instance in a VPC, users need a VPC with at least a subnet set up in all zones of a region you want to deploy instances into. Additionally, users must set up a DB subnet group and a DB security group for your VPC. Large enough CIDR blocks need to be allocated for each subnet to ensure enough spare IP addressed are available.
* VPC security groups exist to help users secure their instances within their VPC. Also, entering and exiting traffic for each subnet can be controlled via ACLs. You can inspect traffic via IPsec VPN by on-premise security.
* You can access instances deployed within a VPC by EC2. Publicly-accessible instances can be accessed via the internet. Instances deployed in your VPC are accessible via a VPN or bastion host or the internet. Using a bastion host requires setup of a public subnet with an internet gateway and routing rules. Public connectivity access requires set up with the publicly accessible option to be set. No configuration of a VPN or bastion host is needed for this. Users can also set up a VPN to extend the corporate network into the VPC.
* If you’d like to move an instance outside of a VPC into the another VPC, you can take a snapshot of the instance to restore within the VPC or use the Restore to Point in Time option. However, **directly moving instances within a VPC outside of the VPC is not supported**. This is for security reasons. If you need to do this, you must export data from the instance to the target instance outside of the VPC.
* Users must ensure they have modified routing tables and ACLs of the VPC to make sure the instance is reachable by client instances. If a failover occurs and a multi-AZ deployment is relaunched, the client instance and RDS instance may be in different zones. If this happens, you must configure the settings to allow communication between the two.
* Users can update existing subnet groups to add more subnets. Removing subnets can cause instances to become unavailable depending on if their zone was removed from the group. Currently, updating a group does not change the subnet of a deployed instance. Changing the subnet group of a deployed instance is not allowable.
* Using RDS requires an AWS developer account. A master account is different and is only used within RDS. It controls the access allowed to your DB instances. The master account is connected to the DB instance. The user chooses the username and password to associate with each instance that is created. When the instance is created, the user can use the master credentials to connect to the database. Creating additional accounts lets the user restrict access to the instance.
* Each engine has default privileges set for the master account which are viewed in the RDS user guide.
* You may configure access to a database in your own data center to work with RDS. You control access for specific IP addresses, IP ranges, and subnets within your own server.
* Encryption of connections is supported in MySQL, SQL Server, MariaDB, Oracle, and PostgreSQL. RDS will create a SSL certificate for every instance. When an encrypted connection is created, the data transferred is also encrypted. SSL encryption can increase the latency of the connection and should not be relied on for securing the instance itself.
* Using the AWS KMS, data at rest can be encrypted for all engines. If a DB instance is running with encryption, data, automated backups, read replicas, and snapshots are encrypted. You cannot encrypt an existing instance. Encryption must be set when the instance is created. For Oracle and SQL Server, transparent data encryption is supported.
* Through AWS IAM, you can control the actions allowed on your resources by users and groups. Also, you can tag resources which allows categorization of resources as well as take actions allowed by the AWS IAM policies for the tags.
* A history of AWS API calls is stored by CloudTrail and then the service delivers a log to you.

## **DB Parameter Groups**

* RDS chooses a default set of parameters for optimal performance for an instance by considering the resources and storage capacity needed. Users can modify these with management APIs.
* DB Parameter Groups set a group of engine configurations which can be applied to your instances. Creating an instance without a group set means a default group will be assigned by RDS. Running an instance with a customized group means your instances will run in the most optimized way.
* Current parameters can be viewed via the AWS Management Console, APIs or command line tools.

## **Multi-AZ Deployments and Read Replicas**

* Users have two replication options. Replication for increasing database availability and protecting from unexpected outages is best gained through a multi-AZ deployment. When doing this, RDS will create a standby replica in a different zone. If there’s a failure, RDS can failover to the standby. Multi-AZ deployments use synchronous replication. The database writes on both the primary and standby in the event of a failover. MySQL, Oracle, PostgreSQL, and SQL Server support Multi-AZ deployments.
* Read Replicas are available for users to scale past capacity constraints on a single instance for read-heavy workloads. Read Replicas can be created using AWS Management Console or the **CreateDBInstnaceReadReplica** API. Traffic can be distributed among multiple read replicas. These are supported by MySQL and PostgreSQL. Read Replicas do not offer the data durability in the event of an unexpected failure.
* Multi-AZ deployments and read replicas can be used in concert to get the best benefits of both.

## **Enhanced Monitoring**

* Enhanced monitoring allows users to get a better look at the health of their instances. You can turn on the monitoring for an instance and set the granularity to collect metrics and process information.
* Enhanced monitoring stores information on CPU, memory, file system and disk I/O metrics as well as others.
* All database engines support enhanced monitoring.
* With the exception of t1.micro and m1.small, all instance types support enhanced monitoring. For instances medium or higher, it is recommended that you set higher granular levels.
* Metrics and information can be view on the console. Also, users can manage the metrics to be monitored and can customize how the dashboard appears.
* Different granular levels can be defined for different instances. Also, users choose which instances to monitor.
* Metrics are stored for an hour with a granularity up to 1 second depending on the user’s setting.
* Metrics are delivered to CloudWatch Logs. Metric filers can be created in CloudWatch Logs. The metrics can be displayed as graphs on the dashboard.
* If you wish to view historical data, you need to use CloudWatch. You can also use CloudWatch to monitor the health of your entire AWS stack. CloudWatch currently supports 1 minute or less granularities.
* Using CloudWatch, you can set alarms to send notifications when states change. You can specify which metric to monitor over a specified time period. You can also set the alarm to perform a certain action.
* You can use CloudWatch Subscriptions to set up a feed for metrics to be used with third-party applications. Also, you can use filters in CloudWatch logs to bridge metrics with your application.
* Retention periods can be set in CloudWatch Logs. The default is 30 days.
* Enhanced monitoring fees are based on transfer and storage rates once the free tier amount is exceeded. Granularities can be set to manage costs.

## **MySQL**

### **DB Engine Version Management**

* Users can control upgrades of the relational database software when new supported versions become available. This allows you flexibility to maintain compatibility with versions, test new versions, and perform upgrades when you want. Unless specified, instances will be upgraded to minor versions when they are supported by RDS. The patching happens during your maintenance window. These upgrades are announced in advance in the forum. Turning off automatic version upgrades can be done by setting **AutoMinorVersionUpgrade** to false. Major upgrades must be started by the customer. While you have the control for upgrades, if a security vulnerability is discovered, RDS may patch the instance.
* The **CreateDBInstance** API can be used to specify the version of the engine to be used. If a user doesn’t specify a version, RDS will default usually to the most recent version. Manually initiating an upgrade can be done via the **ModifyDBInstance** API. The upgrade will take place at the next maintenance window unless the apply immediately flag is set.
* Users can create a snapshot of the existing instance, restore it to a new instance, and initiate the upgrade to test it. After experimenting, you can decide whether to upgrade or not.
* A version upgrade would be major if the version or release level changes.

### **Storage Engines**

* **Using point in time or snapshot restoration with MySQL requires crash-recoverable engine. They are supported only for InnoDB.**

## **Oracle**

### **Licensing and Support**

* BYOL (Bring Your Own License) and License Included are the licensing options available.
* BYOL allows you to use existing licenses to run deployments. Using this method requires you having the appropriate license for the instance class and edition you want to run.
* Using license included means you don’t need already purchased licenses. Pricing for this method includes software, underlying hardware resources, and management capabilities.
* BYOL is available for SE2, SE1, SE, and EE. License included is available for SE1 and SE2.
* If using BYOL, you can use your active support account and should contact Oracle for database specific issues. Users with a premium support account, you can contact premium support for issues relating to RDS.
* If using license included, users with a premium support account will contact them for all issues.
* Changing the licensing option can be done but users must delete the current instance with an ending snapshot and create a new instance with the snapshot and desired licensing option.  
  DB Engine Version Management
* Major version releases are those that change the version or release level.
* **RDS supports versions 11.2 and 12c**.
* You have control over version upgrades. This allows you flexibility to keep compatibility, test new versions, and upgrade on your own time.
* Unless specified, minor upgrades will be scheduled automatically and patching happens during your next maintenance window. These will be announced in advance on the RDS forum. Turning auto upgrades can be done by setting this field to no. Major upgrades are not done due to potential compatibility risk; this are started by the user. If a security vulnerability is discovered RDS may patch on your behalf. If using license included, software update license fee is embedded in the hourly price. BYOL you should have update support from Oracle.
* When creating a new instance, you can choose which version to use.
* You can initate a manual upgrade via the **ModifyDBInstance** API via the **EngineVersion** parameter. This upgrade will take place during the next window unless the apply immediately flag is set.
* Before upgrading to a new version, users can create a snapshot of an existing instance, restoring from this snapshot to create a new instance and then start the version upgrade. This will allow you to test the compatibility before making the upgrade permanent.

### **Scaling**

Options / Features  
• RDS for Oracle allows Multi-AZ deployments for both license models.

• Enterprise Edition uses Oracle Data Guard to protect data. Other editions use a synchronous replication technology to provide Multi-AZ deployments. It also has automatic failover functionality. Multi-AZ deployments are supported by all editions.

• If using BYOL, you should plan on using twice as many licenses. But review your licensing agreement to ensure you’re complying with policies.

• RAC is not supported by RDS.

• Under the BYOL model, advanced security, partitioning, management packs, advanced compression, and total recall are options of Enterprise Edition that are supported.

• RDS supports thirty character sets in Oracle. Setting the desired character set can be done when creating a new instance.