# Relation Database Service – RDS Overview

* Amazon Relational Database Service (RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the cloud.
* RDS provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.
* RDS features & benefits
  + CPU, memory, storage, and IOPS can be scaled independently.
  + manages backups, software patching, automatic failure detection, and recovery.
  + automated backups can be performed as needed, or manual backups can be triggered as well. Backups can be used to restore a database, and the RDS restore process works reliably and efficiently.
  + provides high availability with a primary instance and a synchronous secondary instance that can be failovered to seamlessly when a problem occurs.
  + provides elasticity & scalability by enabling MySQL, MariaDB, or PostgreSQL Read Replicas to increase read scaling.
  + supports MySQL, MariaDB, PostgreSQL, Oracle, Microsoft SQL Server, and the new, MySQL-compatible Amazon Aurora DB engine
  + in addition to the security in the database package, IAM users and permissions can help to control who has access to the RDS database service
  + databases can be further protected by putting them in a VPC, using SSL for data in transit and encryption for data in rest
  + However, **as it is a managed service, shell (root ssh) access to DB instances is not provided**, and this restricts access to certain system procedures and tables that require advanced privileges.

## RDS Components

* **DB Instance**
  + is a basic building block of RDS
  + is an isolated database environment in the cloud
  + each DB instance runs a DB engine. AWS currently supports MySQL, MariaDB, PostgreSQL, Oracle, and Microsoft SQL Server & Aurora DB engines
  + can be accessed from Amazon AWS command line tools, Amazon RDS  
    APIs, or the AWS Management RDS Console.
  + computation and memory capacity of an DB instance is determined by its DB instance class, which can be selected as per the needs
  + for each DB instance, 5 GB to 6 TB of associated storage capacity can be selected
  + storage comes in three types: Magnetic, General Purpose (SSD), and Provisioned IOPS (SSD), which differ in performance characteristics and price
  + each DB instance has a DB instance identifier, which is customer-supplied name and must be unique for that customer in an AWS region. It uniquely identifies the DB instance when interacting with the Amazon RDS API and AWS CLI commands.
  + each DB instance can host multiple databases, or a single Oracle database with multiple schemas.
  + can be hosted in an AWS VPC environment for better control
* **Regions and Availability Zones**
  + AWS resources are housed in highly available data center facilities in different areas of world, these data centers are called regions which further contain multiple distinct locations called Availability Zones
  + Each AZ is engineered to be isolated from failures in other AZs, and to provide inexpensive, low-latency network connectivity to other AZs in the same region
  + DB instances can be hosted in several AZs, an option called a Multi-AZ deployment.
    - Amazon automatically provisions and maintains a synchronous standby replica of the DB instance in a different AZ.
    - Primary DB instance is synchronously replicated across AZs to the standby replica
    - Provides data redundancy, failover support, eliminate I/O freezes, and minimize latency spikes during system backups.
* **Security Groups**
  + security group controls the access to a DB instance, by allowing access to the specified IP address ranges or EC2 instances
* **DB Parameter Groups**
  + A DB parameter group contains engine configuration values that can be applied to one or more DB instances of the same instance type
* **DB Option Groups**
  + Some DB engines offer tools that simplify managing the databases and making the best use of data.
  + Amazon RDS makes such tools available through option groups *for e.g. Oracle Application Express (APEX), SQL Server Transparent Data Encryption, and MySQL memcached support.*

## RDS Interfaces

* RDS can be interacted with multiple interfaces
  + AWS RDS Management console
  + Command Line Interface
  + Programmatic Interfaces which include SDKs, libraries in different languages, and RDS API

## RDS Pricing

* Instance class
  + Pricing is based on the class (e.g., micro, small, large, xlarge) of the DB instance consumed.
* Running time
  + Billed by the instance-hour, which is equivalent to a single instance running for an hour*for e.g., a single instance running for two hours = two instances running for one hour, both consume 2 instance-hours*.
  + if a DB instance runs for only part of an hour, full instance-hour is charged
* Storage
  + Storage capacity provisioned for the DB instance is billed per GB per month.
  + If the provisioned storage capacity is scaled within the month, the bill will be pro-rated.
* I/O requests per month
  + Total number of storage I/O requests made in a billing cycle.
* Backup storage
  + Automated backups & any active database snapshots consume storage
  + Increasing backup retention period or taking additional database snapshots increases the backup storage consumed by the database.
  + RDS provides backup storage up to 100% of the provisioned database storage at no additional charge *for e.g., if you have 10 GB-months of provisioned database storage, RDS provides up to 10 GB-months of backup storage at no additional charge.*
  + Most databases require less raw storage for a backup than for the primary dataset, so if multiple backups are not maintained, you will never pay for backup storage.
  + Backup storage is free only for active DB instances.
* Data transfer
  + Internet data transfer in and out of your DB instance.
* Reserved Instance
  + In addition to regular RDS pricing, reserved DB instances can be purchased

# AWS DynamoDB

* Amazon DynamoDB is a fully managed NoSQL database service that
  + makes it simple and cost-effective to store and retrieve any amount of data and serve any level of request traffic.
  + provides fast and predictable performance with seamless scalability
* DynamoDB enables customers to offload the administrative burdens of operating and scaling distributed databases to AWS, without having to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.
* DynamoDB tables do not have fixed schemas, and table consists of items and each item may have a different number of attributes.
* DynamoDB synchronously replicates data across three facilities in an AWS Region, giving high availability and data durability.
* DynamoDB supports fast in-place updates. A numeric attribute can be incremented or decremented in a row using a single API call
* DynamoDB uses proven cryptographic methods to securely authenticate users and prevent unauthorized data access
* Durability, performance, reliability, and security are built in, with SSD (solid state drive) storage and automatic 3-way replication.
* DynamoDB supports two different kinds of primary keys:
  + **Partition Key**(previously called the **Hash key**)
    - A simple primary key, composed of one attribute
    - DynamoDB uses the partition key’s value as input to an internal hash function; the output from the hash function determine the partition where the item will be stored.
    - No two items in a table can have the same partition key value.
  + **Partition Key and Sort Key**(previously called the **Hash and Range key**)
    - A composite primary key 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 where the item will be stored.
    - All items with the same partition key are stored together, in sorted order by sort key value.
    - It is possible for two items to have the same partition key value, but those two items must have different sort key values.
* DynamoDB Secondary indexes
  + add flexibility to the queries, without impacting performance.
  + are automatically maintained as sparse objects, items will only appear in an index if they exist in the table on which the index is defined making queries against an index very efficient
* DynamoDB throughput and single-digit millisecond latency makes it a great fit for gaming, ad tech, mobile, and many other applications
* ElastiCache can be used in front of DynamoDB in order to offload high amount of reads for non frequently changed data

## DynamoDB Performance

* Automatically scales horizontally
* runs exclusively on Solid State Drives (SSDs).
  + SSDs help achieve the design goals of predictable low-latency response times for storing and accessing data at any scale.
  + SSDs High I/O performance enables it to serve high-scale request workloads cost efficiently, and to pass this efficiency along in low request pricing
* allows provisioned table reads and writes
  + Scale up throughput when needed
  + Scale down throughput four times per UTC calendar day
* automatically partitions, reallocates and re-partitions the data and provisions additional server capacity as the
  + table size grows or
  + provisioned throughput is increased
* Global Secondary indexes (GSI)
  + can be created upfront or added later

## DynamoDB Consistency

* Each DynamoDB table is automatically stored in the three geographically distributed locations for durability
* Read consistency represents the manner and timing in which the successful write or update of a data item is reflected in a subsequent read operation of that same item
* DynamoDB allows user to specify whether the read should be eventually consistent or strongly consistent at the time of the request
  + **Eventually Consistent Reads** (Default)
    - Eventual consistency option maximizes the read throughput.
    - Consistency across all copies is usually reached within a second
    - However, an eventually consistent read might not reflect the results of a recently completed write.
    - Repeating a read after a short time should return the updated data.
  + **Strongly Consistent Reads**
    - Strongly consistent read returns a result that reflects all writes that received a successful response prior to the read
* Query, GetItem, and BatchGetItem operations perform eventually consistent reads by default
  + Query and GetItem operations can be forced to be strongly consistent
  + Query operations cannot perform strongly consistent reads on Global Secondary Indexes
  + BatchGetItem operations can be forced to be strongly consistent on a per-table basis

## DynamoDB Security

* Fine Grained Access Control (FGAC) gives a high degree of control over data in the table
* FGAC helps control *who* (caller) can access *which* items or attributes of the table and perform *what* actions (read/write capability).
* FGAC is integrated with IAM, which manages the security credentials and the associated permissions.

## DynamoDB Advanced Topics

Refer to [DynamoDB Advanced](http://jayendrapatil.com/aws-dynamodb-advanced/) post, which covers DynamoDB Streams, Triggers, Cross Region Replication, DAX, VPC Endpoints etc.

## DynamoDB Encryption

* Data in Transit Encryption
  + can be done by encrypting sensitive data on the client side or using encrypted connections (TLS)
* DynamoDB supports Encryption at rest
  + Encryption at rest enables encryption for the data persisted (data at rest) in the DynamoDB tables.
  + Encryption at rest includes the base tables, secondary indexes
  + Encryption at rest automatically integrates with AWS KMS for managing the keys used for encrypting the tables.
  + Encryption at rest can be enabled only for a new table and not for an existing table
  + Encryption once enabled for a table, cannot be disabled
  + DynamoDB Streams do not support encryption
  + On-Demand Backups of encrypted DynamoDB tables are encrypted using S3’s Server-Side Encryption
  + Encryption at rest encrypts your data using 256-bit AES encryption.

## DynamoDB Costs

* Index Storage
  + DynamoDB is an indexed data store
    - Billable Data = Raw byte data size + 100 byte per-item storage indexing overhead
* Provisioned throughput
  + Pay flat, hourly rate based on the capacity reserved as the throughput provisioned for the table
  + one Write Capacity Unit provides one write per second for items < 1KB in size.
  + one Read Capacity Unit provides one strongly consistent read (or two eventually consistent reads) per second for items < 4KB in size.
  + Provisioned throughput charges for every 10 units of Write Capacity and every 50 units of Read Capacity.
* Reserved capacity
  + Significant savings over the normal price
  + Pay a one-time upfront fee

## DynamoDB Best Practices

* Keep item size small
* Store metadata in DynamoDB and large BLOBs in Amazon S3
* Use table per day, week, month etc for storing time series data
* Use conditional or Optimistic Concurrency Control (OCC) updates
  + Optimistic Concurrency Control is like Optimistic locking in the RDMS
  + OCC is generally used in environments with low data contention, conflicts are rare and transactions can be completed without the expense of managing locks and transactions
  + OCC assumes that multiple transactions can frequently be completed without interfering with each other.
  + Transactions are executed using data resources without acquiring locks on those resources and waiting for other transaction locks to be cleared
  + Before a transaction is committed, it is verified if the data was modified by any other transaction. If so, it would be rollbacked and needs to be restarted with the updated data
  + OCC leads to higher throughput as compared to other concurrency control methods like pessimistic locking, as locking can drastically limit effective concurrency even when deadlocks are avoided
* Avoid hot keys and hot partitions

# AWS Redshift

* Amazon Redshift is a fully managed, fast and powerful, petabyte scale data warehouse service
* Redshift automatically helps
  + set up, operate, and scale a data warehouse, from provisioning the infrastructure capacity
  + patches and backs up the data warehouse, storing the backups for a user-defined retention period
  + monitors the nodes and drives to help recovery from failures
  + significantly lowers the cost of a data warehouse, but also makes it easy to analyze large amounts of data very quickly
  + provide fast querying capabilities over structured data using familiar SQL-based clients and business intelligence (BI) tools using standard ODBC and JDBC connections.
  + uses replication and continuous backups to enhance availability and improve data durability and can automatically recover from node and component failures.
  + scale up or down with a few clicks in the AWS Management Console or with a single API call
  + distribute & parallelize queries across multiple physical resources
  + supports VPC, SSL, AES-256 encryption and Hardware Security Modules (HSMs) to protect the data in transit and at rest.
* **Redshift only supports Single-AZ deployments** and the nodes are available within the same AZ, if the AZ supports Redshift clusters
* Redshift provides monitoring using CloudWatch and metrics for compute utilization, storage utilization, and read/write traffic to the cluster are available with the ability to add user-defined custom metrics
* Redshift provides Audit logging and AWS CloudTrail integration
* Redshift can be easily enabled to a second region for disaster recovery.

## Redshift Architecture

## Redshift Performance

* **Massively Parallel Processing (MPP)**
  + automatically distributes data and query load across all nodes.
  + makes it easy to add nodes to the data warehouse and enables fast query performance as the data warehouse grows.
* **Columnar Data Storage**
  + organizes the data by column, as column-based systems are ideal for data warehousing and analytics, where queries often involve aggregates performed over large data sets
  + columnar data is stored sequentially on the storage media, and require far fewer I/Os, greatly improving query performance
* **Advance Compression**
  + Columnar data stores can be compressed much more than row-based data stores because similar data is stored sequentially on disk.
  + employs multiple compression techniques and can often achieve significant compression relative to traditional relational data stores.
  + doesn’t require indexes or materialized views and so uses less space than traditional relational database systems.
  + automatically samples the data and selects the most appropriate compression scheme, when the data is loaded into an empty table

## Redshift Single vs Multi-Node Cluster

* Single Node
  + single node configuration enables getting started quickly and cost-effectively & scale up to a multi-node configuration as the needs grow
* Multi-Node
  + Multi-node configuration requires a leader node that manages client connections and receives queries, and two or more compute nodes that store data and perform queries and computations.
  + Leader node
    - provisioned automatically and not charged for
    - receives queries from client applications, parses the queries and develops execution plans, which are an ordered set of steps to process these queries.
    - coordinates the parallel execution of these plans with the compute nodes, aggregates the intermediate results from these nodes and finally returns the results back to the client applications.
  + Compute node
    - can contain from 1-128 compute nodes, depending on the node type
    - executes the steps specified in the execution plans and transmit data among themselves to serve these queries.
    - intermediate results are sent back to the leader node for aggregation before being sent back to the client applications.
    - supports Dense Storage or Dense Compute nodes (DC) instance type
      * Dense Storage (DS) allow creation of very large data warehouses using hard disk drives (HDDs) for a very low price point
      * Dense Compute (DC) allow creation of very high performance data warehouses using fast CPUs, large amounts of RAM and solid-state disks (SSDs)
    - direct access to compute nodes is not allowed

## Redshift Availability & Durability

* Redshift replicates the data within the data warehouse cluster and continuously backs up the data to S3 (11 9’s durability)
* Redshift mirrors each drive’s data to other nodes within the cluster.
* Redshift will automatically detect and replace a failed drive or node
* If a drive fails, Redshift
  + cluster will remain available in the event of a drive failure
  + the queries will continue with a slight latency increase while Redshift rebuilds the drive from replica of the data on that drive which is stored on other drives within that node
  + single node clusters do not support data replication and the cluster needs to be restored from snapshot on S3
* In case of node failure(s), Redshift
  + automatically provisions new node(s) and begins restoring data from other drives within the cluster or from S3
  + prioritizes restoring the most frequently queried data so the most frequently executed queries will become performant quickly
  + cluster will be unavailable for queries and updates until a replacement node is provisioned and added to the cluster
* In case of Redshift cluster AZ goes down, Redshift
  + cluster is unavailable until power and network access to the AZ are restored
  + cluster’s data is preserved and can be used once AZ becomes available
  + cluster can be restored from any existing snapshots to a new AZ within the same region

## Redshift Backup & Restore

* Redshift replicates all the data within the data warehouse cluster when it is loaded and also continuously backs up the data to S3
* Redshift always attempts to maintain at least three copies of the data
* Redshift enables automated backups of the data warehouse cluster with a 1-day retention period, by default, which can be extended to max 35 days
* Automated backups can be turned off by setting the retention period as 0
* Redshift can also asynchronously replicate the snapshots to S3 in another region for disaster recovery

## Redshift Scalability

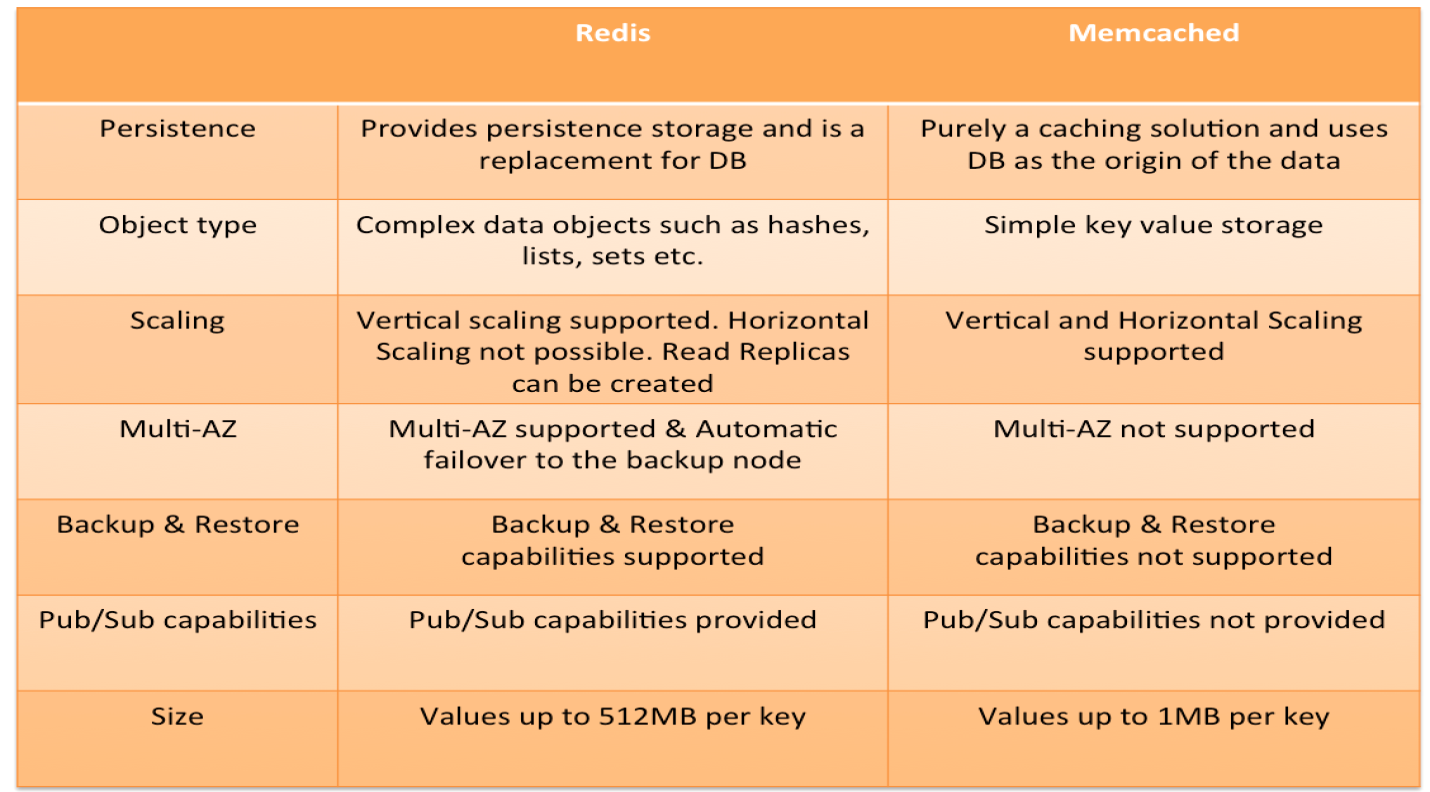
* Redshift allows scaling of the cluster either by
  + increasing the node instance type (Vertical scaling)
  + increasing the number of nodes (Horizontal scaling)
* Redshift scaling changes are usually applied during the maintenance window or can be applied immediately
* Redshift scaling process
  + existing cluster remains available for read operations only while a new data warehouse cluster gets created during scaling operations
  + data from the compute nodes in the existing data warehouse cluster is moved in parallel to the compute nodes in the new cluster
  + when the new data warehouse cluster is ready, the existing cluster will be temporarily unavailable while the canonical name record of the existing cluster is flipped to point to the new data warehouse cluster

## Redshift vs EMR vs RDS

* RDS is ideal for
  + structured data and running traditional relational databases while offloading database administration
  + for online-transaction processing (OLTP) and for reporting and analysis
* Redshift is ideal for
  + large volumes of structured data that needs to be persisted and queried using standard SQL and existing BI tools
  + analytic and reporting workloads against very large data sets by harnessing the scale and resources of multiple nodes and using a variety of optimizations to provide improvements over RDS
  + preventing reporting and analytic processing from interfering with the performance of the OLTP workload
* EMR is ideal for
  + processing and transforming unstructured or semi-structured data to bring in to Amazon Redshift and
  + for data sets that are relatively transitory, not stored for long-term use

# AWS ElastiCache

* AWS ElastiCache is a managed web service that helps deploy and run Memcached or Redis protocol-compliant cache clusters in the cloud easily
* ElastiCache is available in two flavours: **Memcached and Redis**
* ElastiCache helps
  + simplify and offload the management, monitoring, and operation of in-memory cache environments, enabling the engineering resources to focus on developing applications
  + automate common administrative tasks required to operate a distributed cache environment.
  + improves the performance of web applications by allowing retrieval of information from a fast, managed, in-memory caching system, instead of relying entirely on slower disk-based databases.
  + helps improve load & response times to user actions and queries, but also reduce the cost associated with scaling web applications
  + helps automatically detect and replace failed cache nodes, providing a resilient system that mitigates the risk of overloaded databases, which can slow website and application load times
  + provides enhanced visibility into key performance metrics associated with the cache nodes through integration with CloudWatch
  + code, applications, and popular tools already using Memcached or Redis environments work seamlessly, with being protocol- compliant with Memcached and Redis environments
* ElastiCache provides in-memory caching which can
  + significantly lower latency and improve throughput for many
    - read-heavy application workloads *for e.g. social networking, gaming, media sharing and Q&A portals* or
    - compute-intensive workloads such as a recommendation engine
  + improve application performance by storing critical pieces of data in memory for low-latency access.
  + be used to cache results of I/O-intensive database queries or the results of computationally-intensive calculations.
* ElastiCache currently allows access only from the EC2 network and cannot be accessed from outside networks like on-premises servers



## Redis

* [Redis](http://redis.io/) is an open source, BSD licensed, advanced key-value cache & store
* ElastiCache enables the management, monitoring and operation of a Redis node; creation, deletion and modification of the node
* ElastiCache for Redis can be used as a primary in-memory key-value data store, providing fast, sub millisecond data performance, high availability and scalability up to 16 nodes plus up to 5 read replicas, each of up to 3.55 TiB of in-memory data
* ElastiCache for Redis supports (similar to RDS features)
  + Redis Master/Slave replication.
  + Multi-AZ operation by creating read replicas in another AZ
  + Backup and Restore feature for persistence by snapshotting
* ElastiCache for Redis can be vertically scaled upwards by selecting a larger node type, however it cannot be scaled down
* Parameter group can be specified for Redis during installation, which acts as a “container” for Redis configuration values that can be applied to one or more Redis primary clusters
* Append Only File (AOF)
  + provides persistence and can be enabled for recovery scenarios
  + if a node restarts or service crash, Redis will replay the updates from an AOF file, thereby recovering the data lost due to the restart or crash
  + cannot protect against all failure scenarios, cause if the underlying hardware fails, a new server would be provisioned and the AOF file will no longer be available to recover the data
  + Enabling Redis Multi-AZ as a Better Approach to Fault Tolerance, as failing over to a read replica is much faster than rebuilding the primary from an AOF file

### Redis Read Replica

* Read Replicas help provide Read scaling and handling failures
* Read Replicas are kept in sync with the Primary node using Redis’s asynchronous replication technology
* Redis Read Replicas can help
  + Horizontal scaling beyond the compute or I/O capacity of a single primary node for read-heavy workloads.
  + Serving read traffic while the primary is unavailable either being down due to failure or maintenance
  + Data protection scenarios to promote a Read Replica as primary node, in case the primary node or the AZ of the primary node fails
* ElastiCache supports initiated or forced failover where it flips the DNS record for the primary node to point at the read replica, which is in turn promoted to become the new primary
* Read replica cannot span across regions and may only be provisioned in the same or different AZ of the same Region as the cache node primary

### Redis Multi-AZ

* ElastiCache for Redis shard consists of a primary and up to 5 read replicas
* Redis asynchronously replicates the data from the primary node to the read replicas
* ElastiCache for Redis Multi-AZ mode
  + provides enhanced availability and smaller need for administration as the node failover is automatic
  + impact on the ability to read/write to the primary is limited to the time it takes for automatic failover to complete
  + no longer needs monitoring of Redis nodes and manually initiating a recovery in the event of a primary node disruption
* During certain types of planned maintenance, or in the unlikely event of ElastiCache node failure or AZ failure,
  + it automatically detects the failure,
  + selects a replica, depending upon the read replica with the smallest asynchronous replication lag to the primary, and promotes it to become the new primary node
  + it will also propagate the DNS changes so that the the primary endpoint remains the same
* If Multi-AZ is not enabled,
  + ElastiCache monitors the primary node
  + in case the node becomes unavailable or unresponsive, it will repair the node by acquiring new service resources
  + it propagates the DNS endpoint changes to redirect the node’s existing DNS name to point to the new service resources.
  + If the primary node cannot be healed and you will have the choice to promote one of the read replicas to be the new primary

### Redis Backup & Restore

* Backup and Restore allows users to create snapshots of the Redis clusters
* Snapshots can be used for recovery, restoration, archiving purpose or warm start an ElastiCache for Redis cluster with preloaded data
* Snapshots can created on a cluster basis and uses Redis’ native mechanism to create and store an RDB file as the snapshot
* Increased latencies for a brief period at the node might be encountered while taking a snapshot, and is recommended to be taken from a Read Replica minimizing performance impact
* Snapshots can be created either automatically (if configured) or manually
* ElastiCache for Redis cluster when deleted removes the automatic snapshots. However, manual snapshots are retained

## Memcached

* [Memcached](http://memcached.org/) is an in-memory key-value store for small chunks or arbitrary data
* ElastiCache for Memcached can be used to cache a variety of objects
  + from the content in persistent data stores such as RDS, DynamoDB, or self-managed databases hosted on EC2) to
  + dynamically generated web pages *for e.g. with Nginx* or
  + transient session data that may not require a persistent backing store
* ElastiCache for Memcached
  + can be scaled Vertically by increasing the node type size
  + can be scaled Horizontally by adding and removing nodes
  + does not support persistence of data
* ElastiCache for Memcached cluster can have
  + nodes can span across multiple AZs within the same region
  + maximum of 20 nodes per cluster with a maximum of 100 nodes per region (soft limit and can be extended)
* ElasticCache for Memcached supports auto discovery, which enables automatic discovery of cache nodes by clients when they are added to or removed from an ElastiCache cluster

## ElastiCache Mitigating Failures

* ElastiCache should be designed to plan so that failures have a minimal impact upon your application and data
* Mitigating Failures when Running Memcached
  + Mitigating Node Failures
    - spread the cached data over more nodes
    - as Memcached does not support replication, a node failure will always result in some data loss from the cluster
    - having more nodes will reduce the proportion of cache data lost
  + Mitigating Availability Zone Failures
    - locate the nodes in as many availability zones as possible, only the data cached in that AZ is lost, not the data cached in the other AZs
* Mitigating Failures when Running Redis
  + Mitigating Cluster Failures
    - Redis Append Only Files (AOF)
      * enable AOF so whenever data is written to the Redis cluster, a corresponding transaction record is written to a Redis AOF
      * when Redis process restarts, ElastiCache creates a replacement cluster and provisions it and repopulating it with data from AOF
      * It is time consuming
      * AOF can get big
      * Using AOF cannot protect you from all failure scenarios
    - Redis Replication Groups
      * A Redis replication group is comprised of a single primary cluster which your application can both read from and write to, and from 1 to 5 read-only replica clusters.
      * Data written to the primary cluster is also asynchronously updated on the read replica clusters
      * When a Read Replica fails, ElastiCache detects the failure, replaces the instance in the same AZ and synchronizes with the Primary Cluster
      * Redis Multi-AZ with Automatic Failover, ElastiCache detects Primary cluster failure, promotes a read replica with least replication lag to primary
      * Multi-AZ with Auto Failover is disabled, ElastiCache detects Primary cluster failure, creates a new one and syncs the new Primary with one of the existing replicas
  + Mitigating Availability Zone Failures
    - locate the clusters in as many availability zones as possible