ElastiCache

* + Fully managed implementations of two popular in-memory data stores
    - Redis
    - Memcached
  + EC is a web service that makes it easy to deploy and run Memcached or Redis protocol-compliant server nodes in the cloud
  + The in-memory caching provided by EC can be used to significantly improve latency and throughput for many read-heavy application workloads or compute-intensive workloads
  + **Best for scenarios where the DB load is based on Online Analytics Processing(OLAP) transactions**
  + Push-button scalability for memory, writes and reads
  + In-memory key/value store - not persistent in the traditional sense
  + Billed by node size and hours of use
  + EC EC2 nodes cannot be accessed from the Internet, nor can they be accessed by EC2 instances in other VPCs
  + Cached information may include the results of I/O-intensive database queries or the results of computationally-intensive calculations
  + Can be on-demand or reserved instances too(but not Spot instances)
  + EC can be used for storing session state
  + A node is fixed-sized chunck of secure, network-attached RAM and is the smalles building block
  + Each node runs an instance of Memcached or Redis protocol-compliant service and has its own DNS name and port
  + Failed nodes are automatically replaced
  + Access to Elasticache nodes is controlled by VPC SG's and subnet groups (when deployed in a VPC)
  + Subnet groups are a collection of subnets designed for your Amazon ElastiCache Cluster
  + You cannot move an existing Amazon ElastiCache Cluster form outside the VPC into a VPC
  + You need to configure subnet groups for EC for the VPC that hosts the EC2 instances and the EC cluster
  + When not using a VPC, Amazon EC allows you to control access to your clusters through Cache SG's(you need to link the corresponding EC2 SG's)
  + Elasticache nodes are deployed in clusters and can span more than one subnet of the same subnet group
  + A cluster is a collection of one or more nodes using the same caching engine
  + Applications connect to Elasticache clusters using endpoints
  + An endpoint is a node or cluster's unique address
  + Maintenance windows can be defined and allow software patching to occur
  + There are two types of EC engine
    - Memcached
      * simplest model, can run large nodes with multiple cores/threads
      * Can be scaled in and out
      * Can cache objects such as DBs
    - Redis
      * Complex model
      * Supports encryption
      * Master/slave replication
      * Cross AZ(HA)
      * Automatic failover
      * backup/restore

* + Use Cases
    - Table

      Description automatically generated

* + The table below describes requirements that would determine whether to use Memcached or Redis engine
    - Table

      Description automatically generated
  + Memcached
    - Not persistent
    - Cannot be used as a data store
    - Supports large nodes with multiple cores or threads
    - Scales out and in, by adding and removing nodes
    - Ideal front-end for data stores(RDS, DynamoDB etc)
    - Use Cases
      * Cache the contents of a DB
      * Cache data from dynamically generated web pages
      * Transient session data
      * High frequency counters for admission control in high volume web apps
    - Max 100 nodes per region, 1-20 nodes per cluster(soft limits)
    - Can integrate with SNS for node failure/recovery notification
    - Supports auto-discovery for nodes added/removed from the cluster
    - Scales out/in(horizontally) by adding/removing nodes
    - Scales up/down(vertically) by changing the node family/type
    - Does not support multi-AZ failover or replication
    - Does not support snapshots
    - You can place nodes in different AZs
    - With EC Memcached each node represents a partition of data and nodes in a cluster can span availability zones:
    - Graphical user interface, application, Word

      Description automatically generated
  + Redis
    - Data is persistent
    - Can be used as a datastore
    - Not multi-threaded
    - **Scales by adding shards, not nodes**
    - A Redis shard is a subset of the cluster's keyspace, that can include a primary node and zero or more read-replicas
    - Supports automatic and manual snapshots(S3)
    - Backups include cluster data and metadata
    - You can restore your data by creating a new Redis cluster and populating it from a backup
    - Supports master/slave replication
    - During backup you cannot perform CLI or API operations on the cluster
    - Automated backups are enabled by default(automatically deleted with Redis deletion)
    - You can only move snapshots between regions by exporting them from EC before moving between regions(can then populate a new cluster with data)
    - Multi-AZ is possible using read replicas in another AZ in the same region
    - Clustering mode disabled:
      * You can have only one shard
      * One shard can have one read/write primary node and 0-5 read only replicas
      * You can distribute the replicas over multiple AZs in the same region
      * Replication from the primary node is asynchronous
    - A Redis cluster with cluster mode disabled is represented in the diagram below:
    - Graphical user interface, application

      Description automatically generated
    - Multi-AZ failover
      * Failures are detected by EC
      * EC automatically promotes the replica that has the lowest replica lag
      * DNS records remain the same but point to the IP of the new primary
      * Other replicas start to sync with the new primary
      * You can have fully automated, fault toleratn Elasticache-Redis implementation by enabling both cluster mode and multi-AZ failover
      * The following table compares the Memcached and Redis engines:
      * Table

        Description automatically generated
    - Charges
      * Pricing is per Node-hour consumed for each Node Type
      * Partial Node-hours consumed are billed as full hours
      * There is no charge for data transfer between EC2 and EC within the same AZ
    - HA for EC
      * Memcached:
        + Because Memcached does not support replication, a node failure will result in data loss
        + Use multiple nodes to minimize data loss on node failure
        + Launch multiple nodes across available AZs to minimize data loss on AZ Failure
      * Redis
        + Use multiple nodes in each shard and distribute the nodes across multiple AZs
        + Enable Multi-AZ on the replication group to permit automatic failover if the primary node fails
        + Schedule regular backups of your Redis cluster