Fundamentals: RDS + Aurora + ElastiCache

Notes by Christina Grys

Amazon RDS Overview

- RDS – Relational Database Service
    - It's a managed DB service for DB use SQL as a query language
    - Allows to create databases in cloud that are managed by AWS
        - Postgres SQL
        - MySQL
        - MariaDB
        - Oracle
        - Microsoft SQL Server
        - Aurora (AWS Proprietary database)

Advantage over using RDS vs. Deploying DB on EC2

- RDS is a managed service:
- Automated provisioning, OS patching
- Continuous backups and restore specific timestamp
- Monitoring dashboards
- Read replicas for improved read performance
- Multi AZ setup for DR (Disaster Recovery)
- Maintenance windows for upgrades
- Scaling capability (vertical and horizontal)
- Storage backed by EBS (gp2 or io I)

BUT you can't SSH into your instances

RDS backups

- Backups are automatically enabled in RDS
- Automated backups
- Daily full backup of the database (during the maintenance window)
- Transaction logs are backed-up by RDS every 5 minutes

  ➔ Ability to restore to any point in time (from oldest back up to 5 minutes ago)
- 7 days retention (can be increased to 35 days)

DB Snapshots:

- Manually triggered by the user

- Retention of backup for as long as you want

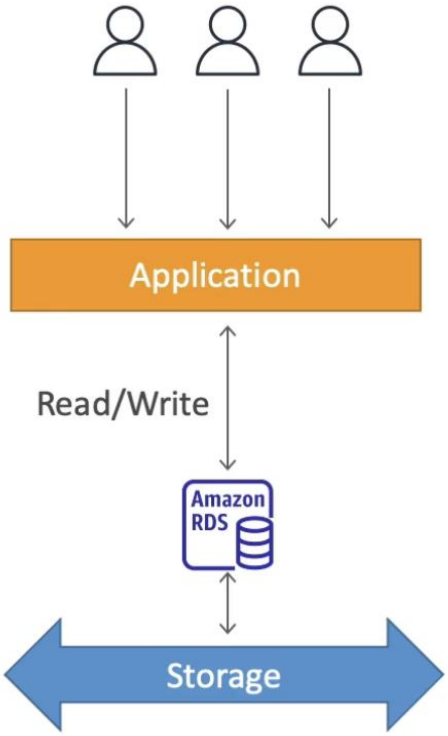
RDS – Storage Auto Scaling

- Helps you increase storage on your RDS DB instance dynamically
- When RDS detects you are running out of free database storage, it scales automatically
- Avoid manually scaling your database storage
- You have to set **Maximum Storage Threshold** (max. limit for DB storage)

Automatically modify storage if:

- Free storage is less than 10% of allocated storage
- Low-storage lasts at least 5 minutes
- 6 hours have passed since last modification

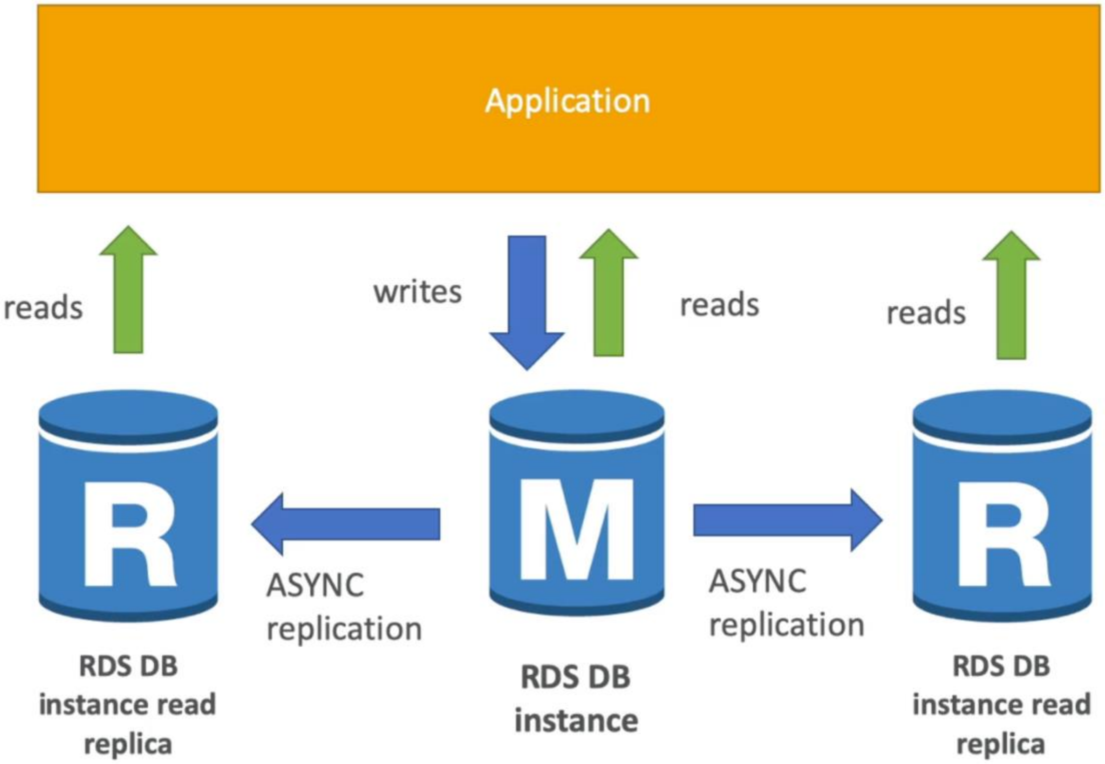
Useful for applications with unpredictable workloads
Supports all RDS database engines (MariaDB, MySQL, PostgreSQL, SQL, Server, Oracle)



RDS Read Replicas vs Multi AZ

RDS Read Replicas for read scalability

- Creates up to 5 read replicas
- Within AZ, Cross AZ, or Cross Region
- Replication is ASYNC, so reads are eventually consistent
- Replicas can be promoted to their own DB
- Applications must update the connection string to leverage read replicas



RDS Read Replicas – Use Cases

- You have a production database that takes on normal load
- You want to run a reporting application to run some analytics
- You create a Read Replica to run the new workload there
- The production application is unaffected
- Read replicas are used for SELECT (=read) only kind of statements (not INSERT, UPDATE, DELETE)

RDS Multi AZ (Disaster Recovery)

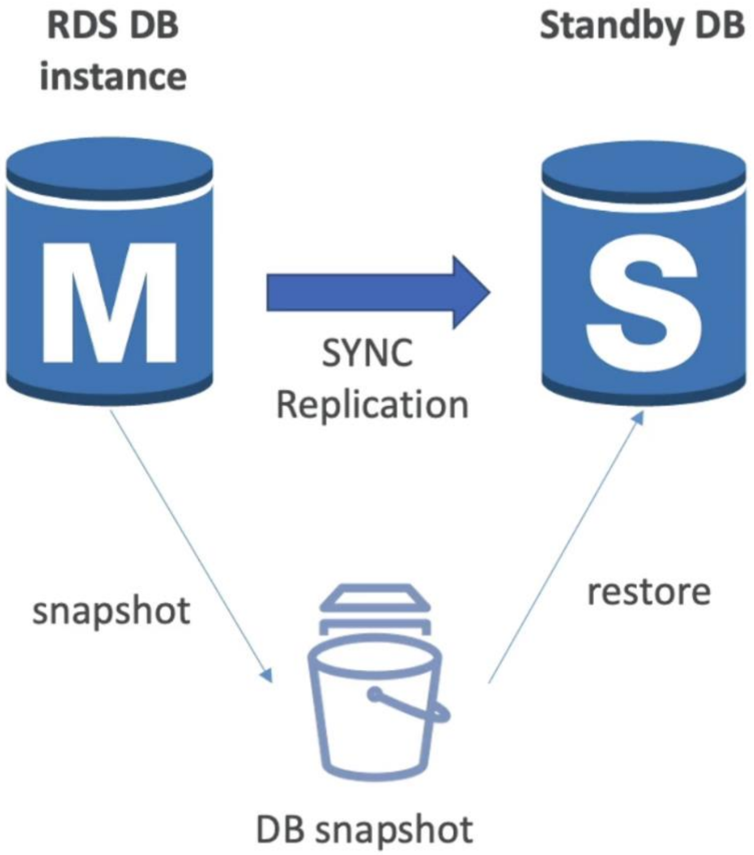
- SYNC replication
- One DNS name – automatic app failover to standby
- Increase availability

- Failover in case of loss of AZ, loss of network, instance or storage failure
- No manual intervention in apps
- Not used for scaling
- Note: The Read Replicas be setup as Multi AZ for Disaster Recovery (DR)

*** Yes, you CAN set up a RDS as a Multi AZ if you want it to – common exam question***

RDS – From Single-AZ to Multi-AZ

- Zero downtime operation (no need to stop the DB)
- Just click on "modify" for the database
- The following happens internally:
  - A snapshot is taken
  - A new DB is restored from the snapshot in a new AZ
  - Synchronization is established between the two databases



RDS Encryption + Security

RDS Security – Encryption

At rest encryption

- Possibility to encrypt the master & read replicas with AWS KMS – AES-256 encryption
- Encryption has to be defined at launch time
- If the master is not encrypted, the read replicas cannot be encrypted
- Transparent Data Encryption (TDE) available for Oracle and SQL Server

In-flight encryption

  - SSL certificates to encrypt data to RDS in flight
  - Provide SSL options with trust certificate when connecting to database
  - To enforce SSL:
  - PostgreSQL: rds.force_ssl=l in the AWS RDS Console (Parameter Groups)
  - MySQL: Within the DB:
  - GRANT USAGE ON ** TO 'mysqluser'@'%' REQUIRE SSL;

RDS Encryption Operations

Encrypting RDS backups

- Snapshots of un-encrypted RDS databases are un-encrypted
- Snapshots of encrypted RDS databases are encrypted
- Can copy a snapshot into an encrypted one

To encrypt an un-encrypted RDS database:

- Create a snapshot of the un-encrypted database
- Copy the snapshot and enable encryption for the snapshot
- Restore the database from the encrypted snapshot
- Migrate applications to the new database, and delete the old database

RDS Security – Network & IAM

Network Security

- RDS databases are usually deployed within a private subnet, not in a public one
- RDS security works by leveraging security groups (the same concept as for EC2 instances) – it controls which IP / security group can communicate with RDS

Access Management

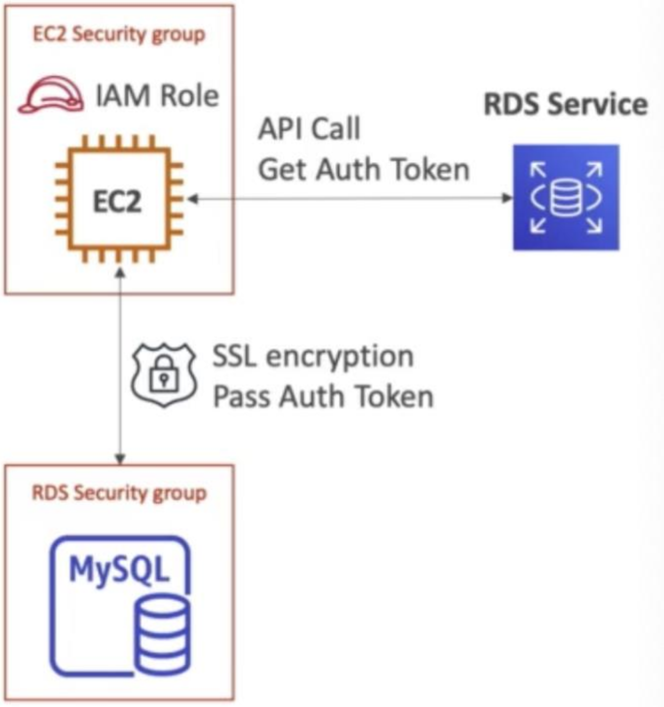
- IAM policies help control who can manage AWS RDS (through the RDS API)
- Traditional Username and Password can be used to login into the database
- IAM-based authentication can be used to login into RDS MySQL & PostgreSQL

RDS – IAM Authenitcation

- IAM database authentication works with MySQL and PostgreSQL
- You don't need a password, just an authentication token obtained through IAM & RDS API calls
- Auth token has a lifetime of 15 minutes

Benefits:

- Network in/out must be encrypted using SSL
- IAM to centrally manage users instead of DB
- Can leverage IAM Roles and EC2 Instance profiles for easy integration



RDS Security – SUMMARY

Encryption at rest:

- Is done only when you first create the DB instance

- Or: unencrypted DB => snapshot => copy snapshot as encrypted => create DB from snapshot

Your responsibility:

- Check the ports / IP / security group inbound rules in DB's SG
- In-database user creation and permissions or manage through IAM
- Creating a database with or without public access
- Ensure parameter groups or DB is configured to only allow SSL connections

AWS responsibility:

- No SSH access
- No manual DB patching
- No manual OS patching
- No way to audit the underlying instance

Aurora Overview

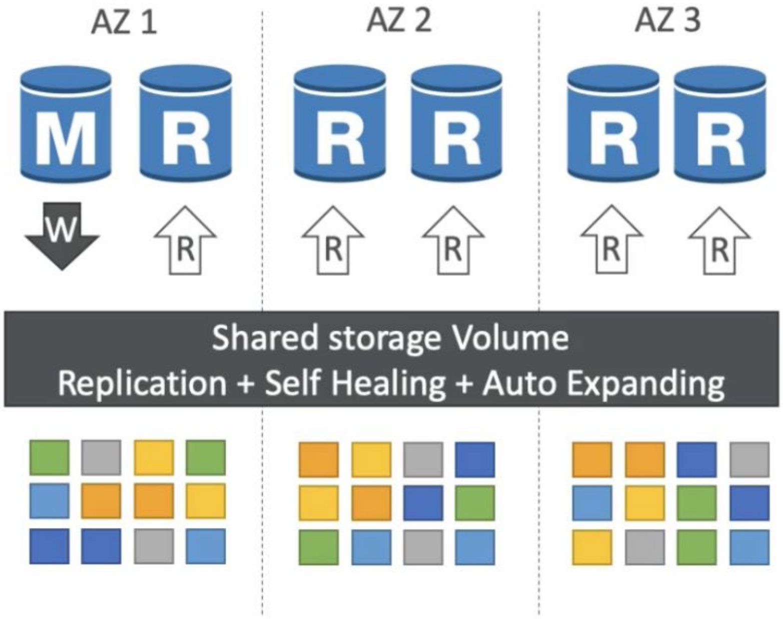
Amazon Aurora

- Aurora is a proprietary technology from AWS (not open sourced)
- Postgres and MySQL are both supported as Aurora DB (that means your drivers will work as if Aurora was a Postgres or MySQL database)
- Aurora is "AWS cloud optimized" and claims 5x performance improvement over MySQL on RDS, over 3x the performance of Postgres on RDS
- Aurora storage automatically grows in increments of 10 GB up to 64 TB
- Aurora can have 15 replicas while MySQL has 5, and the replication process is faster (sub 10 ms replica lag)
- Failover in Aurora is instantaneous. It's HA native.
- Aurora cost more than RDS (20% more) but it is more efficient

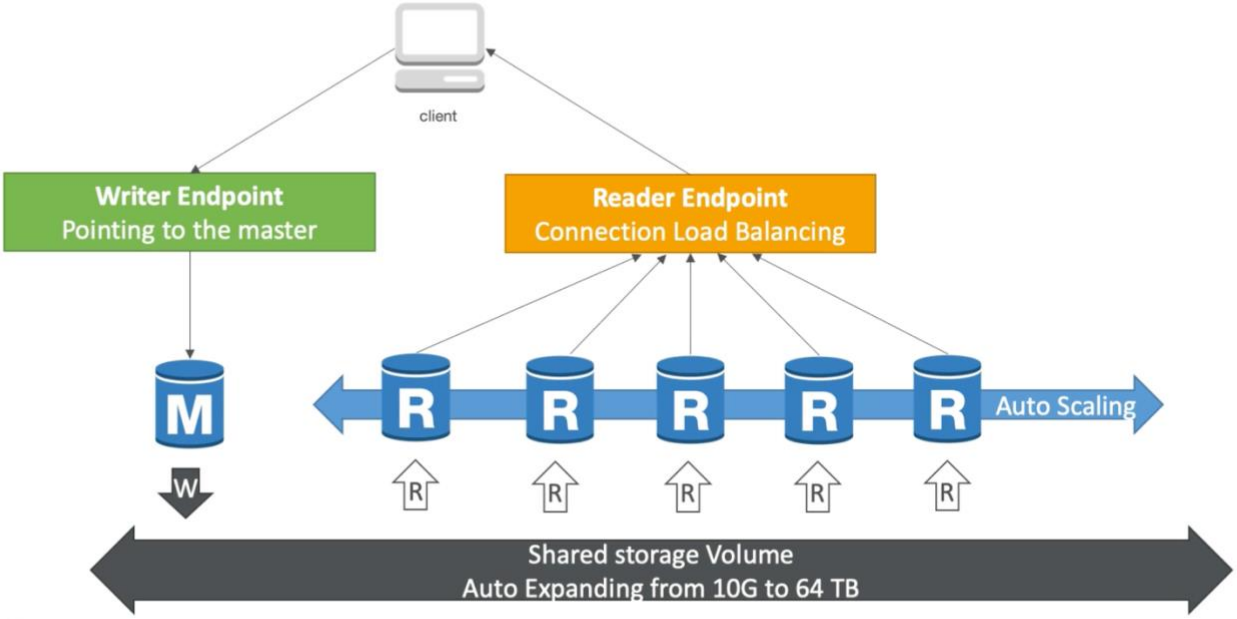
FEATURES: Automatic fail-over, backup and recovery, isolation and security, industry compliance, push-button scaling, automated patching with zero downtime, advanced monitoring, routine maintenance, backtrack: restore data at any point of time without using backups

Aurora High Availability and Read Scaling

- 6 copies of your data across 3 AZ:
  - 4 copies out of 6 needed for writes
  - 3 copies of 6 need for reads
  - Self-healing with peer-to-peer replication
  - Storage is striped across 100s of volumes
- One Aurora Instance takes writes (master)
- Automated failover for master in less than 30 seconds
- Master + up to 15 Aurora Read Replicas serve reads
- Support for Cross Region Replication



AURORA DB CLUSTER:



Aurora Security

- Similar to RDS because uses the same engines

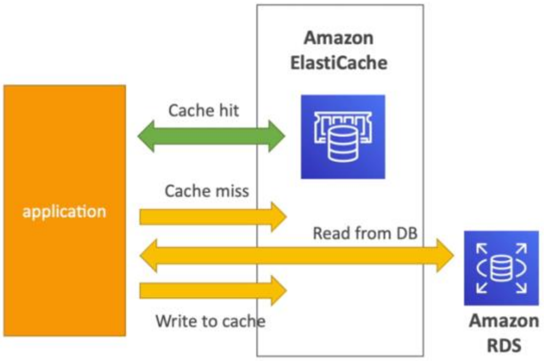
- Encryption at rest using KMS
- Automated backups, snapshots and replicas are also encrypted
- Encryption in flight using SSL (same process as MySQL or Postgres)
- Possibility to authenticate using IAM token (same method as RDS)
- You are responsible for protecting the instance with security groups
- You can't SSH

ElastiCache Overview

- Same way RDS is to get managed Relational Databases
- ElastiCache is to get managed Redis or Memcached
- Caches are in-memory databases with really high performance, low latency
- Helps reduce load off of databases for read intensive workloads
- Helps make your application stateless
- AWS take care of OS maintenance / pathing, optimizations, setup, configuration, monitoring, failure recovery and backups,
- Using ElastiCache involves heavy application code changes

ElastiCache Solution Architecture – DB Cache

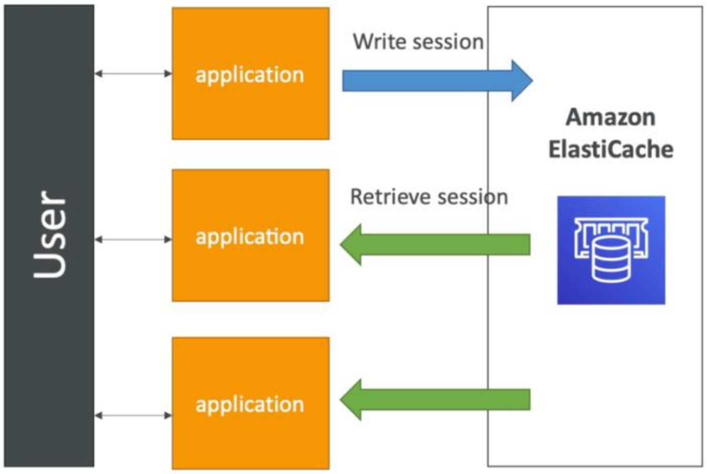
- Applications queries ElastiCache, if not available, get from RDS and store in ElastiCache.
- Helps relieve load in RDS
- Cache must have an invalidation strategy to make sure only the most current data is used in there.



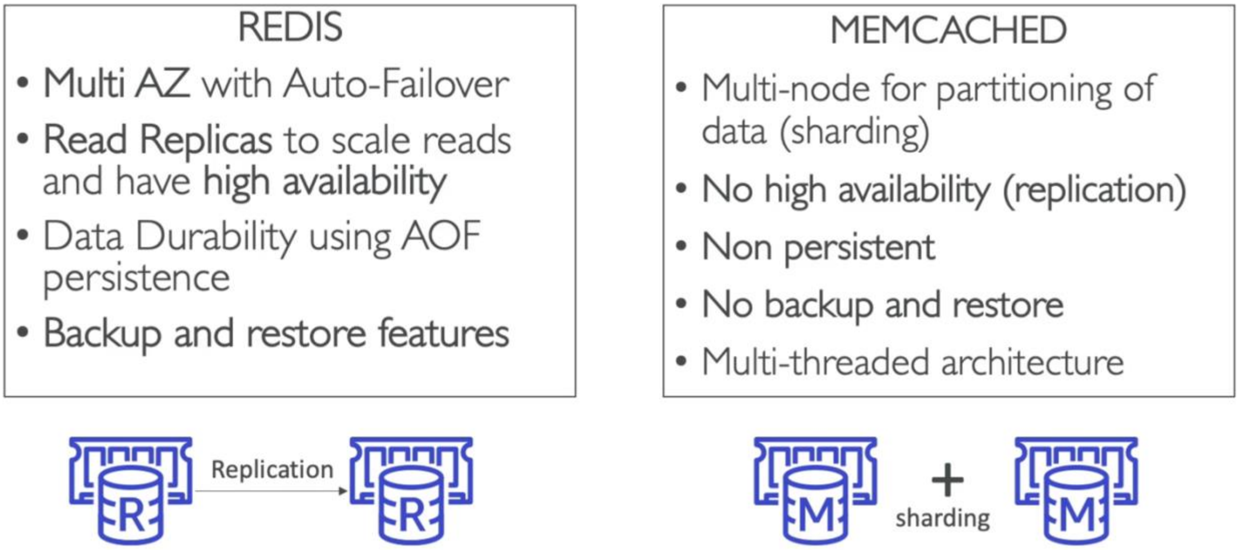
ElastiCache Solution Architecture – User Session Store

- User logs into any of the application

- The application writes the session data into ElastiCache
- The user hits another instance of our application
- The instance retrieves the data and the user is already logged in



ElastiCache – Redis vs. Memcached



REDIS

- Multi AZ with Auto-Failover
- Read Replicas to scale reads and have high availability
- Data Durability using AOF persistence
- Backup and restore features

MEMCACHED

- Multi-node for partitioning of data (sharding)
- No high availability (replication)
- Non persistent
- No backup and restore
- Multi-threaded architecture

R
Replication
R
M
+
sharding
M

ElastiCache Strategies

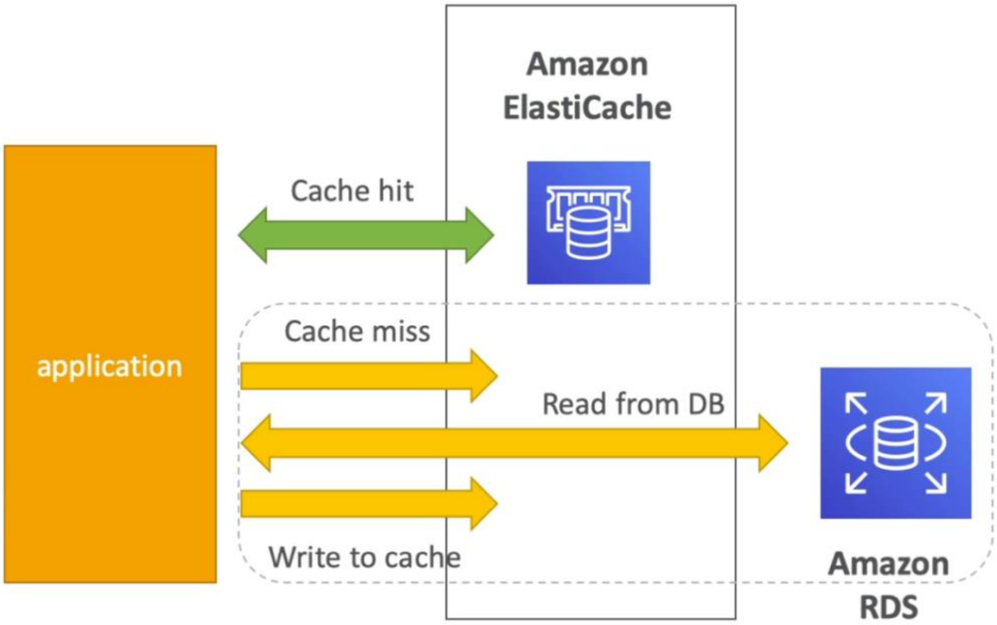
Caching Implementation Considerations
Read more at: https://aws.amazon.com/caching/implementation-considerations/

- Is it safe to cache data? Data may be out of date, eventually consistent
- Is caching effective for that data?
    - Pattern: data changing slowly, few keys are frequently needed
    - Anti-patterns: data changing rapidly, all large key space frequently needed
- Is data structure well caching?
    - Ex: key value caching, or caching of aggregations results

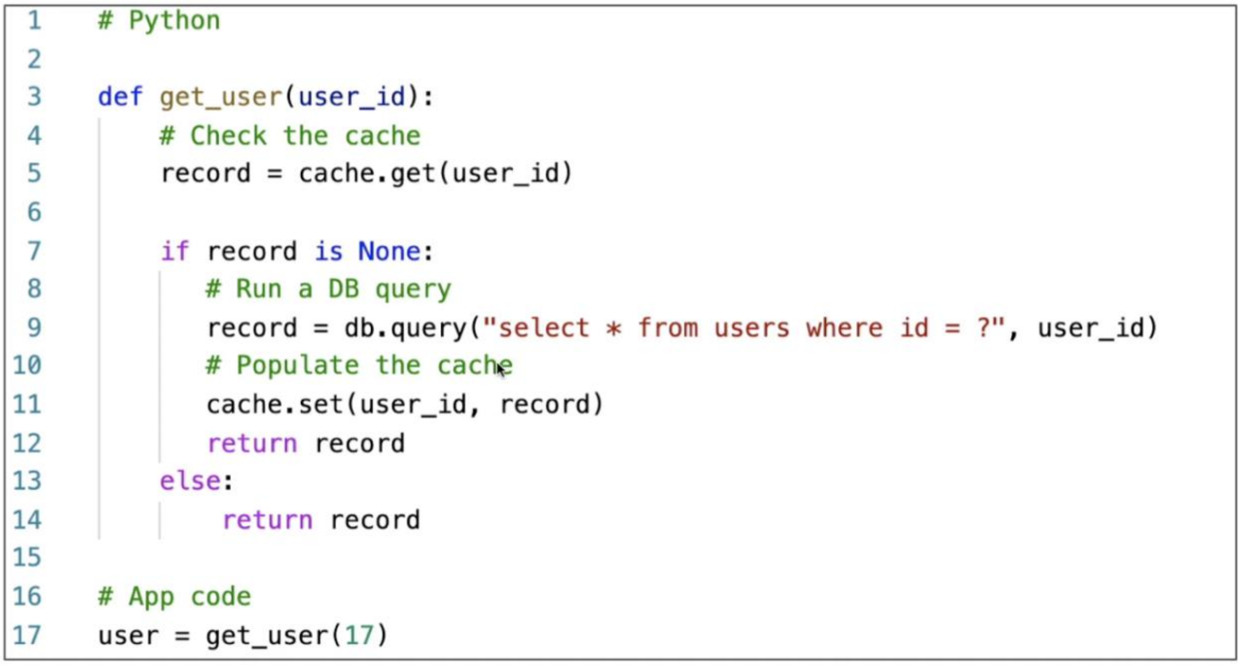
Which caching design pattern is the most appropriate?

Lazy Loading / Cache-Aside / Lazy Population

- Pros
    - Only requested data is cached (the cache isn't filled up with unused data)
    - Node failures are not fatal (just increased latency to warm the cache)
- Cons
    - Cache miss penalty that results in 3 round trips, noticeable delay for that request
    - Stale data: data can be updated in the database and outdated in the cache



Lazy Loading / Cache-Aside / Lazy Population Python Pseudocode



```python
# Python

def get_user(user_id):
    # Check the cache
    record = cache.get(user_id)

    if record is None:
        # Run a DB query
        record = db.query("select * from users where id = ?", user_id)
        # Populate the cache
        cache.set(user_id, record)
        return record
    else:
         return record

# App code
user = get_user(17)
```

Write Through – Add or Update cache when database is update

- Pros:
  - Data in cache is never stale, reads are quick
  - Write penalty vs. Read Penalty (each write requires 2 calls)
- Cons:
  - Missing Data until it is added/updated in the DB. Mitigation is to implement Lazy Loading strategy as well
  - Cache churn – a lot of data will never be read

Cache Evictions and Time-to-live (TTL)

- Cache eviction can occur in three ways:
  - You delete the item explicitly in the cache
  - Item is evicted because the memory is full and it's not recently used (LRU)
  - You set an item time-to-live (or TTL)

- TTL are helpful for any kind of data:
  - Leaderboards
  - Comments
  - Activity streams
- TTL can range from few seconds to hours or days
- If too many evictions happen due to memory, you should scale up or out

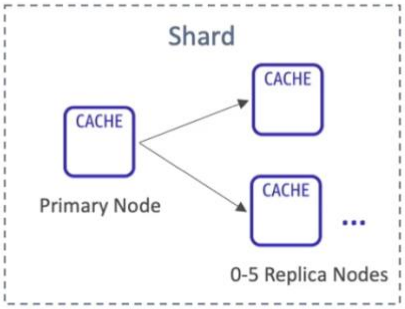
Final words of wisdom

- Lazy Loading / Cache aside is easy to implement and works for many situations as a foundation, especially on the read side
- Write-through is usually combined with Lazy Loading as targeted for the queries or workloads that benefit from this optimization
- Setting a TTL is usually not a bad idea, except when you're using Write-through. Set it to a sensible value for your application
- Only cache the data that makes sense (user profiles, blogs, etc.)
- *Quote: There are only two hard things in Computer Science: cache invalidation and naming things*

ElastiCache Replication: Cluster Mode Disabled

- One primary node, up to 5 replicas
- Asynchronous Replication
- The primary node is used for read/write
- The other nodes are read-only
- One shard, all nodes have all the data
- Guard against data loss if node failure
- Multi-AZ enabled by default for failover
- Helpful to scale read performance

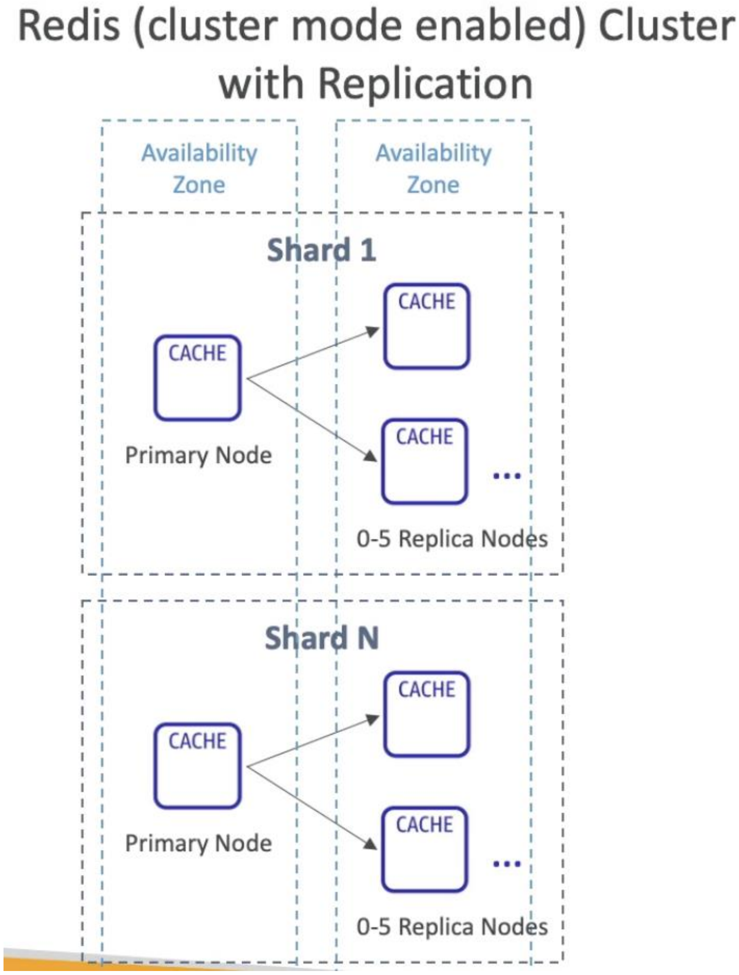
Redis (cluster mode disabled) Cluster
with Replication



The second mode: Cluster mode ENABLED

- Data is partitioned across shards (helpful to scale writes)
- Each shard has a primary and up to 5 replica nodes (same concept as before)
- Multi-AZ capability
- Up to 500 nodes per cluster:
- 500 shards with single master
- 250 shards with I master and I replica
- …

- 83 shards with one master and 5 replicas



Redis (cluster mode enabled) Cluster with Replication