SECTION 4) AWS Fundamentals : RDS +Aurora+ Elastic Cache

**AWS RDS OVERVIEW**

Machine generated alternative text:
AVVS RDS Overview 
• RDS stands for Relational Database Service 
• It's a managed DB service for DB use SQL as a query language. 
• It allows you to create database 
cloud that are managed by AWS 
• postgres 
• MySQL 
• MariaDB 
• Oracle 
• Microsoft SQL Server 
• Aurora (AWS Proprietary database) 

Machine generated alternative text:
Advantage over using RDS versus deploying 
DB on EC2 
• RDS is a managed service: 
• Automated provisioning, OS patching 
• Continuous backups and restore tö•øee• c timestamp (Point in Time Restore)! 
• Monitoring dashboards 
• Read replicas for improved re p 
• Multi AZ setup for DR (Disaster cove 
• Maintenance windows for upgrades 
• Scaling capability (vertical and horizontal) 
• Storage backed by EBS (gp2 or iol) 
• BUT you can't SSH into your instances 

Machine generated alternative text:
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 backup 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 

Machine generated alternative text:
RDS Read Replicas for read scalability 
reads 
RDS DB 
instance read 
replica 
• 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 
reads 
replication 
RDS DB 
instance read 
replica 
Application 
reads 
replication 
RDS DB 
instance 

Machine generated alternative text:
RDS Read Replicas 
• You have a production database 
that is taking 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 
zread) only kind of statements 
not INSERT, UPDATE, DELETE) 
Use Cases 
Production 
Application 
reads 
ASYNC 
replication 
RDS DB 
Reporting 
Application 
reads 
RDS DB 
instance read 
repiica 

Machine generated alternative text:
RDS Read Replicas — Network Cost 
• In AWS there's a network cost when data goes from one AZ to another 
• To reduce the cost, you can have your Read Replicas in the same AZ 
Availability Zone 
us-east-la 
RDS DB 
instance 
: Availability Zone , 
us-east-lb 
RDS DB 
instance read 
replica 
Availability Zone 
us-east-la 
Replication 
Same AZ 
Free 
, Replication 
Cross AZ 
RDS DB 
instance 
RDS DB 
instance read 
replica 

Machine generated alternative text:
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) 
Application 
writes 
One DNS name — automatic failover 
SYNC 
replication 
RDS DB 
instance standby 
RDS Master D 
instance (AZ A 

AWS RDS hands on

Create first database in RDS

Enter endpoint

Machine generated alternative text:
MySQL Connections 
Local instance MySQL80 
Local 
Connection Name: aws 
Connection Remote Management System Profile 
Method to use to connect to the RDBMS 
Name or IP address of the sewer host - and 
Name of the user to connect with. 
The user's password. Will be requested later if its 
not set. 
The schema to use as default schema. Leave 
blank to select it later. 
Connection Met-rod : 
Parameters SSL 
username : 
Password: 
Default Schema: 
Standard (TCP/IP) 
Advanced 
admin 

Machine generated alternative text:
Amazon RDS 
Dashboard 
Databases 
Query Editor 
Performance Insights 
Snapshots 
Automated backups 
Reserved instances 
Subnet groups 
Parameter groups 
Option groups 
Events 
Event subscriptions 
Recommendations 
x 
Successfully created database my-first-sq!. 
Databases my-first-sql 
RDS 
Info 
@ Available 
Engine 
MySQL Community 
Configuration 
View credential details 
Modify 
Stop 
Reboot 
Actions A 
my-first-sql 
Summary 
DB identifier 
my-first-sql 
Role 
Instance 
Connectivity & security 
CPU 
Current activity 
Clas 
db.t 
Regi 
Create read replica 
Promote 
Take snapshot 
Restore to point in time 
Monitoring 
Logs & events 
Tags 
Maintenance & backups 

**RDS Encryption - Security**

Machine generated alternative text:
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 rnaster 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_sslzl in the AWS RDS Console (Parameter Groups) 
• MySQL: Within the DB: 
GRANT USAGE ON REQUIRE SSL; 

Machine generated alternative text:
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 

Machine generated alternative text:
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 
• Access Management 
• IAM policies help control who 
roup can communicate with RDS 
u 
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 

Machine generated alternative text:
RDS - IAM Authentication 
• 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 
EQ Security group 
IAM Role 
EC2 
RDS Service 
API Call 
Get Auth Token 
SSL encryption 
Pass Auth Token 
RDS Security group 
MySQL 

Machine generated alternative text:
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** is a **MySQL** and PostgreSQL-compatible relational database built for the cloud, that combines the performance and availability of traditional enterprise databases with the simplicity and cost-effectiveness of open source databases.

Machine generated alternative text:
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 I OGB, up to 64 T B. 
• 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 costs more than RDS (20% more) — but is more effcient 

Machine generated alternative text:
Aurora High Availability and Read Scaling 
• 6 copies of your data across 3 AZ: 
AZ 2 
• 4 copies out of 6 needed for writes 
• 3 copies out of 6 need for reads 
• Self healing with peer-to-peer replication 
• Storage is striped across I OOS of volumes 
• One Aurora Instance takes writes (master) 
Shared storage Volume 
• Automated failover for master in less than 
Replication + Self Healing + Auto Expanding 
30 seconds 
• Master + up to 15 Aurora Read Replicas 
serve reads 
• Support for Cross Region Replication 

Machine generated alternative text:
Aurora DB Cluster 
Auto Scalin 
Writer Endpoint 
Pointing to the master 
M 
w 
R 
Reader Endpoint 
Connection Load Balancing 
R 
R 
Shared storage Volume 
Auto Ex andin from IOG to 64 TB 

Machine generated alternative text:
Features of Aurora 
• 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 

**NOTE :**

**Aurora security is same as RDS security**

Machine generated alternative text:
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 as MySQL or Postgres) 
• Possibility to authenticate using«q.. ...fi05en (same method as RDS) 
• You are responsible for protectingthe Instance with security groups 
• You can't SSH 

**Think like autoscaling**

Machine generated alternative text:
Aurora Serverless 
• Automated database 
instantiation and auto- 
scaling based on actual 
usage 
• Good for infrequent, 
intermittent or 
unpredictable workloads 
• No capacity planning 
needed 
• Pay per second, can be 
more cost-effective 
Amazon 
Client 
Proxy Fleet 
(managed by Aurora) 
Amazon 
Shared storage Volume 
Amazon 
Aurora 

Machine generated alternative text:
Global Aurora 
• Aurora Cross Region Read Replicas: 
• Useful for disaster recovery 
• Simple to put in place 
• Aurora Global Database (recommended): 
• I Primary Region (read / write) 
• Up to 5 secondary (read-only) regions, replication lag is 
less than I second 
• Up to 1 6 Read Replicas per secondary region 
• Helps for decreasing latency 
• Promoting another region (for disaster recovery) has an 
RTO of K I minute 
us-east-I - 
PRIMARY region 
Amazon 
Aurora 
Applications 
Read / Write 
replication 
eu-west-l - SECONDARY region 
Amazon 
Aurora 
Applications 
Read Only 

**NOTE: aurora is not free**

**Elasticache Overview**

**Imp :- useful for cache redis and memcache**

Machine generated alternative text:
AWS ElastiCache Overview 
• The same way RDS is to get managed Relational Databases... 
• ElastiCache is to get managed Redis or Memcached 
• Caches are in-memory databases wit really high performance, low latency 
• Helps reduce load off of datab sf rea intensive workloads 
• Helps make your application 
ele 
• Write Scaling using sharding 
• Read Scaling using Read Replicas 
• Multi AZ with Failover Capability 
• AWS takes care of OS maintenance / patching, optimizations, setup, 
configuration, monitoring, failure recovery and backups 

Machine generated alternative text:
ElastiCache 
Solution Architecture - 
DB Cache 
Cache hit 
Cache miss 
• 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. 
Amazon 
ElastiCache 
Read from DB 
Amazon 
RDS 
application 
Write to cac 

**It relieve load from database and share session**

Machine generated alternative text:
ElastiCache 
Solution Architecture 
• 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 
User Session Store 
Write session 
application 
Retrieve sessi n 
application 
application 
Amazon 
ElastiCache 

Machine generated alternative text:
ElastiCache — Redis vs Memcached 
REDIS 
MEMCACHED 
• Multi AZ with Auto-Failover 
• Multi-node for partitioning of 
• Read Replicas to scale reads 
data (sharding) 
and have high availability 
• Non persistent 
• Data Durability using AOF 
• No backup and restore 
persistence 
• Multi-threaded architecture 
• Backup and restore features 
Replication 

**Elastic cache hands on**

**Search elasticache**

**Elasticache consideration**

Machine generated alternative text:
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 consister 
• 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 structured well for caching? 
• example: key value caching, or caching of aggregations results 
desi n attem is the most a ro riate? 
n a note here. 

**Case 1**

Machine generated alternative text:
Lazy Loading / Cache-Aside / Lazy Population 
Amazon 
ElastiCache 
Cache hit 
Cache miss 
application 
Read from DB 
Write to cach 
Amazon 
RDS 
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 

Machine generated alternative text:
Lazy Loading / Cache-Aside / Lazy Population 
Python Pseudocode 
P', user_id) 
1 
2 
3 
4 
5 
6 
7 
8 
9 
lø 
11 
12 
13 
14 
15 
16 
17 
# python 
def : 
# Check the cache 
record cache. get(user_id) 
if record is None: 
# Run a DB query 
record db. * from users where id 
# Populate the cache 
cache. record) 
return record 
else: 
return record 
# App code 
user — 

Machine generated alternative text:
Write Through — 
Add or Update cache when database is updated 
Cache hit 
application 
2) Write to che 
Amazon 
ElastiCache 
1) Write to DB 
• 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 the 
data will never be read 
Amazon 
RDS 

Machine generated alternative text:
Write-Through 
Python Pseudocode 
P' , user_id, 
values) 
1 
2 
3 
4 
5 
6 
7 
8 
9 
lø 
11 
12 
13 
14 
15 
16 
17 
# python 
def values) : 
# Save to DB 
record db. users where id 
# Push into cache 
cache. set (user_id, record) 
return record 
# App code 
save_user(17, ("name": "Nate Dogg")) 
user 

Machine generated alternative text:
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 (CRU) 
• You set an item time-to-live (or TTL) 
• ITI- 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 

Machine generated alternative text:
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 17 L 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... ) 