# Amazon Relational Database Service (Amazon RDS)

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the AWS Cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.

## **Overview of Amazon RDS**

* Managed relational database service
* It’s a managed DB service for DB use SQL as a query language.
* It allows you to create databases in the cloud that are managed by AWS
  + Postgres
  + MySQL
  + MariaDB
  + Oracle
  + Microsoft SQL Server
  + Aurora (AWS Proprietary database)

**Advantage over using RDS versus deploying DB on EC2**

* RDS is a managed service:
* Automated provisioning, OS patching
* Continuous backups and restore to specific timestamp (Point in Time Restore)!
* 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 io1)
* 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 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

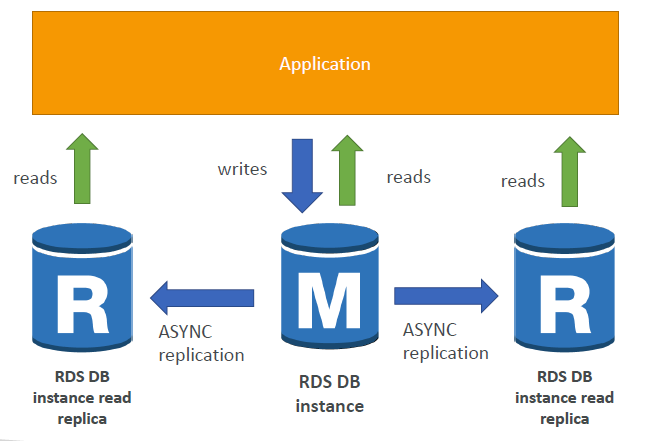
**RDS Read Replicas for read scalability**

Amazon RDS uses the MariaDB, MySQL, Oracle, PostgreSQL, and Microsoft SQL Server DB engines' built-in replication functionality to create a special type of DB instance called a read replica from a source DB instance.

Updates made to the source DB instance are asynchronously copied to the read replica.

you can reduce the load on your source DB instance by routing read queries from your applications to the read replica.

Using read replicas, you can elastically scale out beyond the capacity constraints of a single DB instance for read-heavy database workloads.

****

* 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 is taking on normal load.

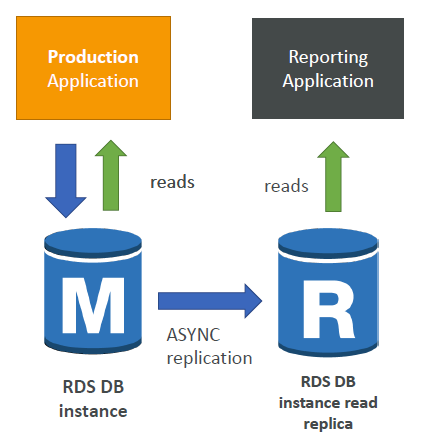
• But if You want to run a reporting application to run some analytics to the same DB then its going to be overloaded & slows down the applications

Instead

• You can create a Read Replica to run the new workload there

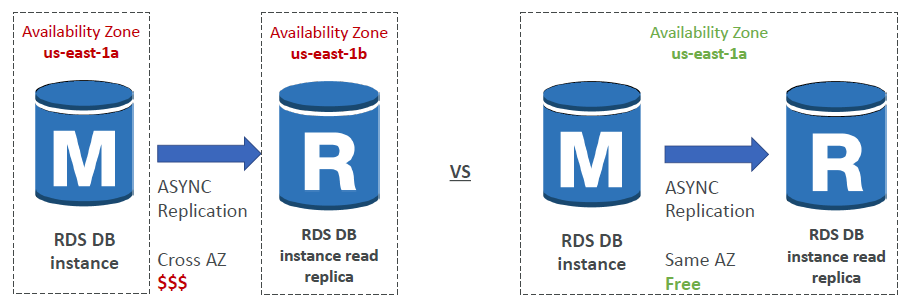
• The production application is unaffected

• Read replicas are used for SELECT (=read) only kind of statements (not INSERT, UPDATE, DELETE)



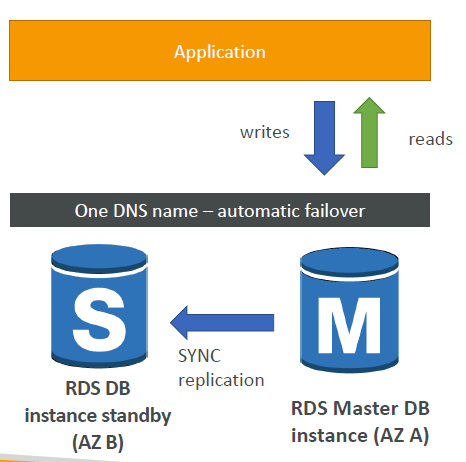
**RDS Read Replicas – Network Cost**

* 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



**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)



**RDS HANDS ON**

Create SQL or Ppostgress data base

Install SQL client in your machine [SQLTRON] & connect the database

**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=1 in the AWS RDS Console (Parameter Groups) [all the client must use ssl cert]
* 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

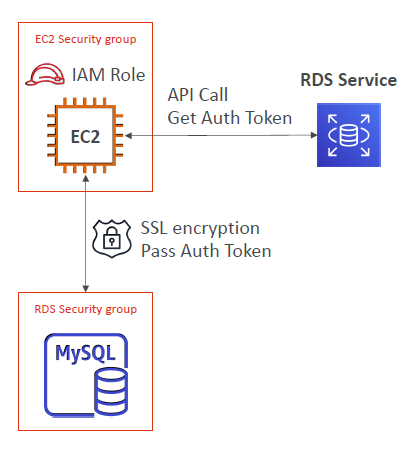
* IAM policies help control who can manage AWS RDS (through the RDS API)
* Traditional Username and Password can be used to login into the database
* IAM-based authentication can be used to login into RDS MySQL & PostgreSQL

**RDS - IAM 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



© Stephane Maarek

**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:

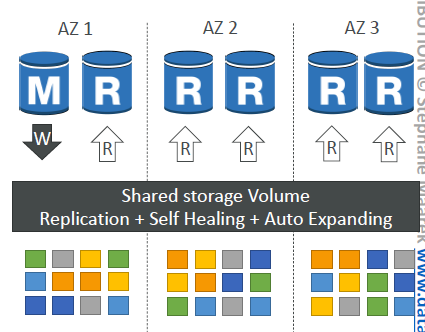
* No SSH access
* No manual DB patching
* No manual OS patching
* No way to audit the underlying instance

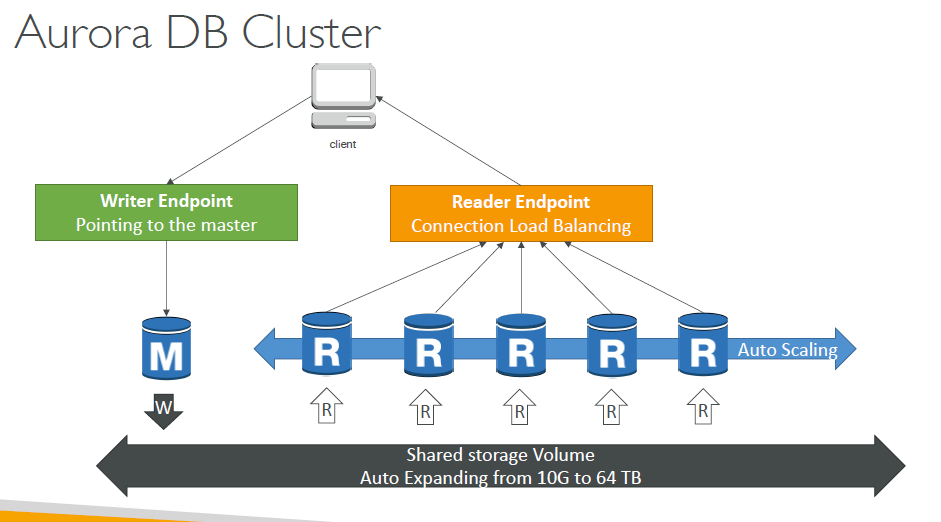
**Amazon Aurora**

* 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 10GB, 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 (High Availability) native.
* Aurora costs more than RDS (20% more) – but is more efficient

**Aurora High Availability and Read Scaling**

* It stores 6 copies of your data across 3 AZ:
* 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 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



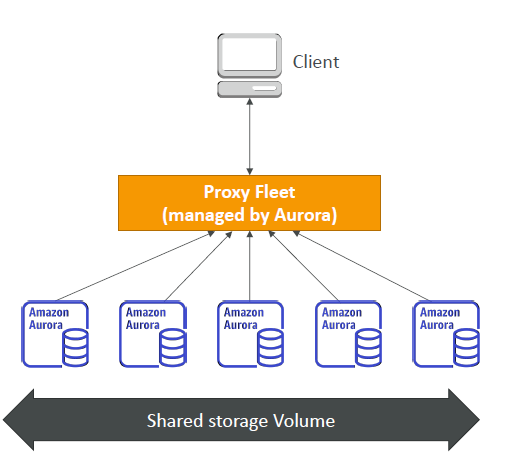


**Features of Aurora**

* 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

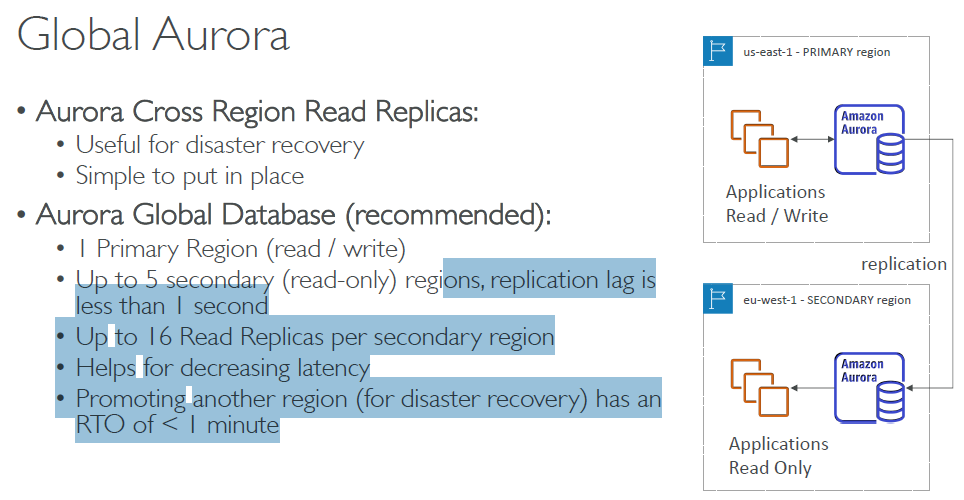
**Aurora Security**

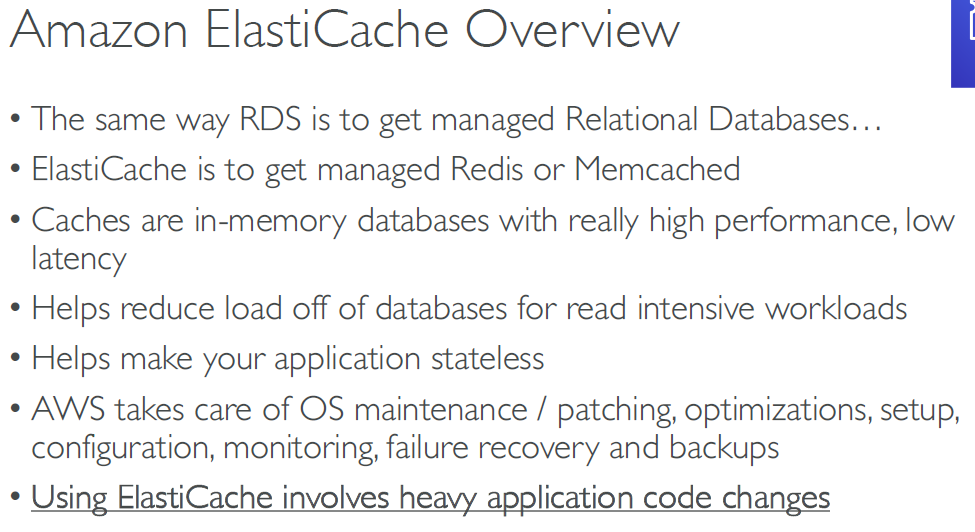
* Similar to RDS because uses the same engines
* Encryption at rest using KMS
* Automated backups, snapshots and replicas are also encrypted
* Encryption in flight using SSL (same process as MySQL or Postgres)
* Possibility to authenticate using IAM token (same method as RDS)
* You are responsible for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

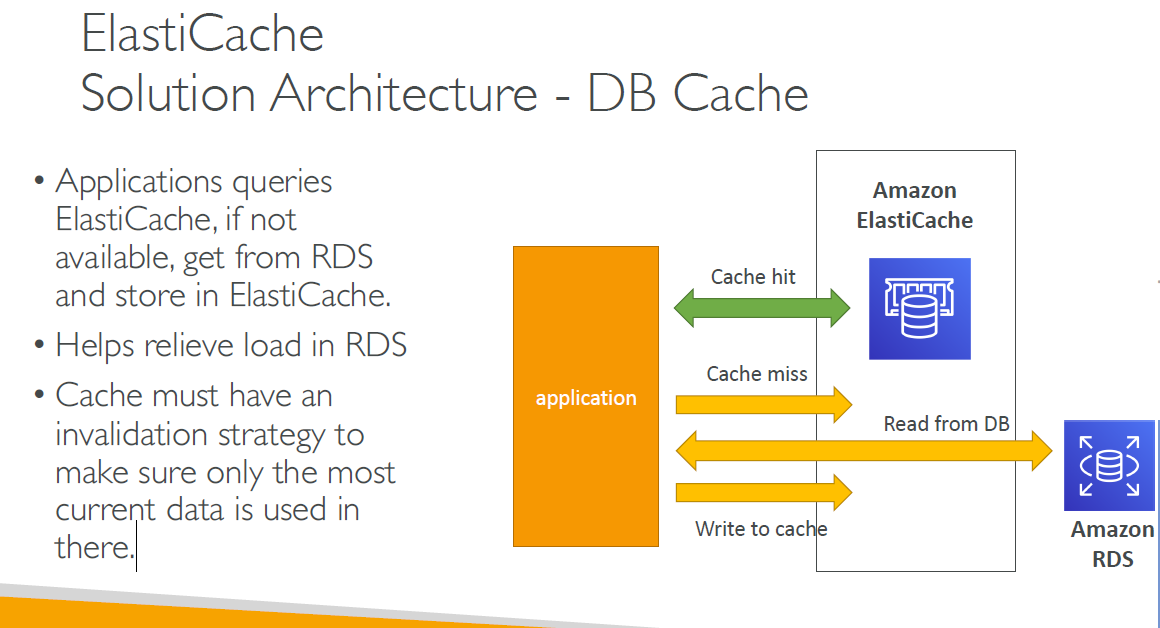


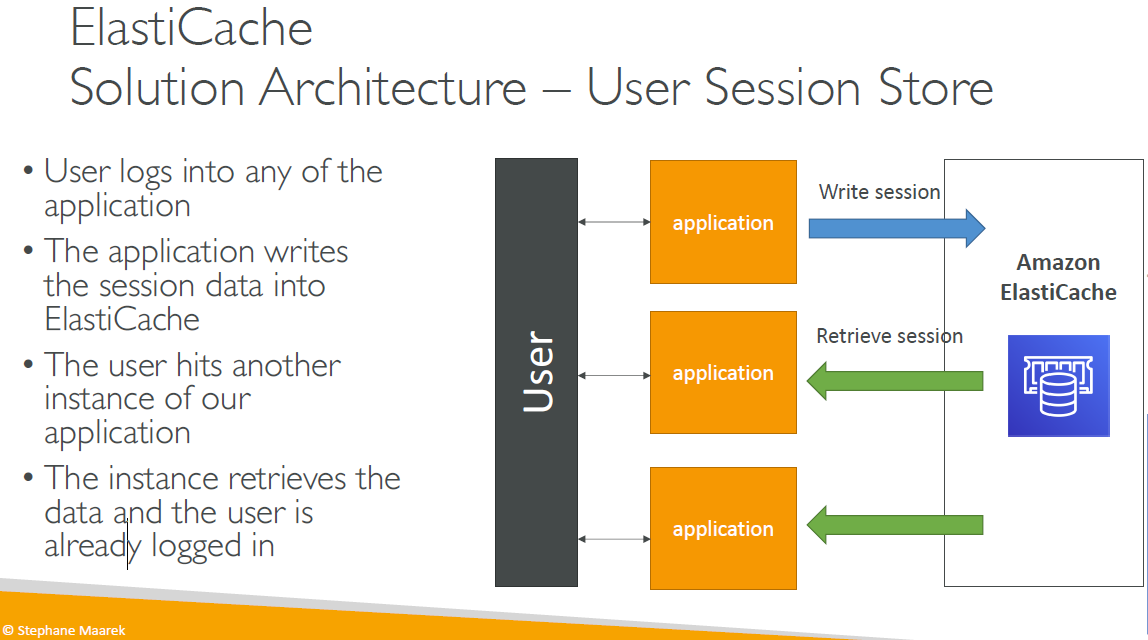
**Aurora Serverless Shared storage Volume**

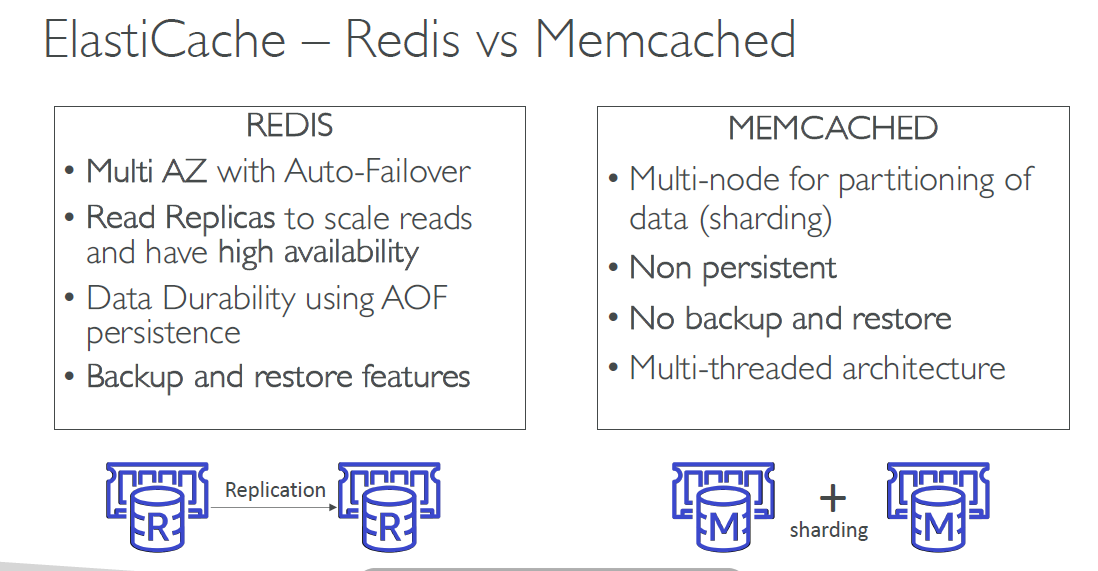
* Automated database Client instantiation and autoscaling based on actua usage.
* Good for infrequent, intermittent or unpredictable workloads
* No capacity planning needed
* Pay per second, can be more cost-effective



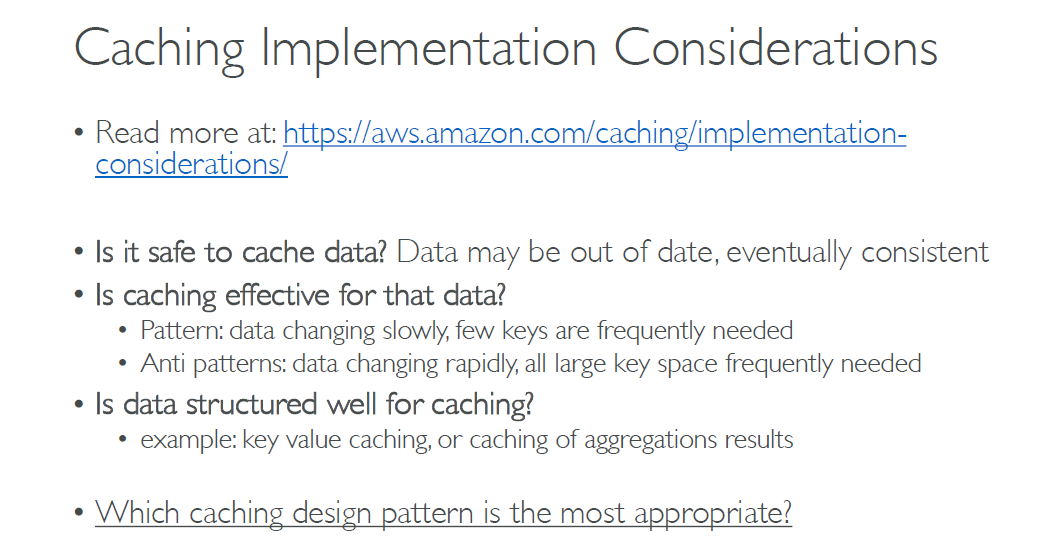








Redis Port: 6349



Cache is not for every type of data.

Pattern yes

Anto pattern no

