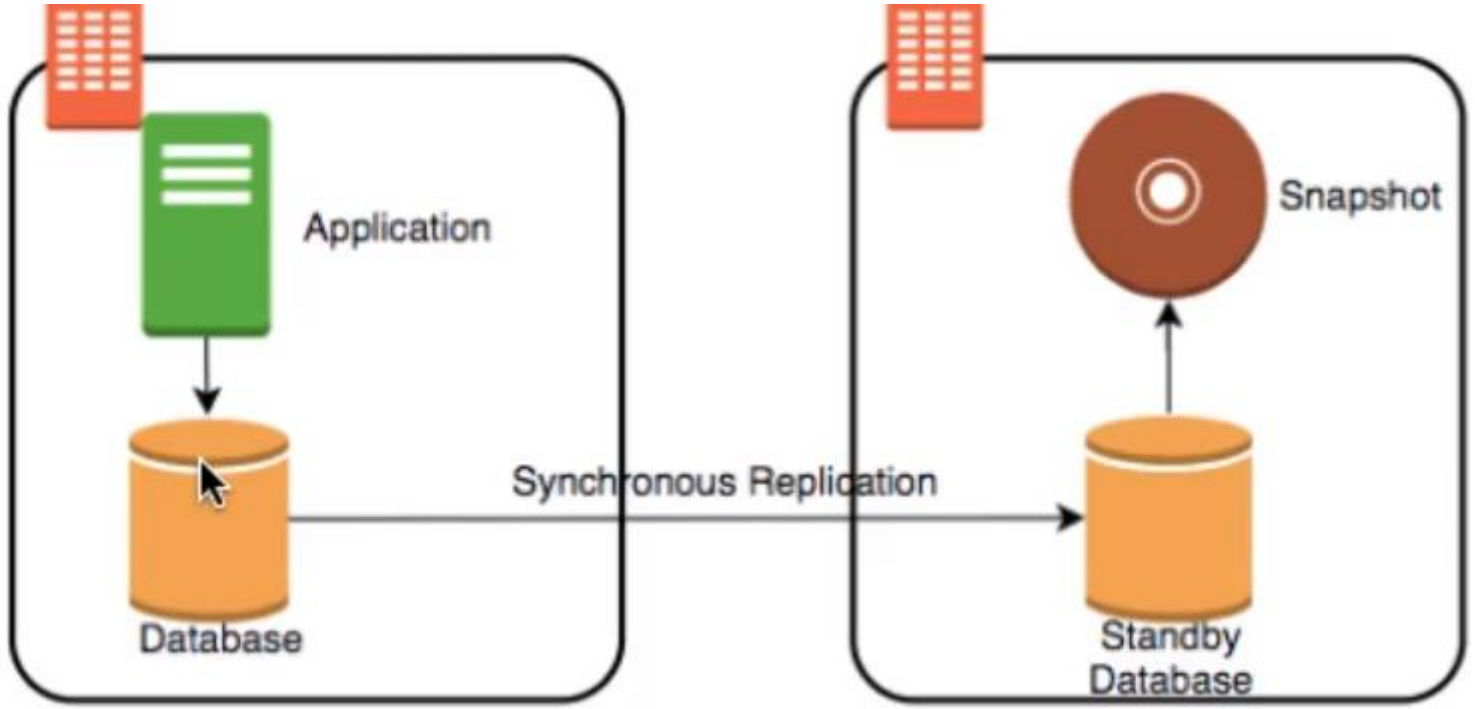
Database

- Databases provide organized and persistent storage for your data
- To choose between different database types, we would need to understand:
  - Availability
  - Durability
  - RPO (**Recovery Point Objective)** Maximum acceptable period of data loss
  - RTO (**Recovery Time Objective)** -Maximum acceptable downtime
  - Consistency

Database

- Challenges (How to solve the challenges?)
  1. Database will go down if datacenter crashes
  2. Data loss if datacenter crashes (Database snapshops will help to take backup)
  3. Database slow down during backup

Database



Datacenter -1 -2

Datacenter

- Challenges solved
  - Database Standby
  - Database Snapshot

Database

Availability

- Will I be able to access my data now and when I need it
- Percentage of time an application provides the operations expected of it
  - Increasing standby

Durability

- Will my data be available aʿer 10 or 100 or 1000 years
- 11 9's - 99.999999999 (If you **store one million files for ten million years**, you expect to **lose one file)**
  - **Increase Durability ,** Multiple copies of the data (Standby, transaction logs and replicas)

- Typically, an **availability of four 9's** is considered very good
- Typically, a **durability of eleven 9's** is considered very good

RTO and RPO

- You are running an EC2 instance storing its data on a EBS. You are taking EBS snapshots every 48 hours. If the EC2 instance crashes, you can manually bring it back up in 45 minutes from the EBS snapshot. What is your RTO and RPO?
- RTO - 45 minutes
- RPO - 48 hours

RTO and RPO

Scenario	Solution
Very small data loss (RPO - 1 minute) Very small downtime (RTO - 5 minutes) Hot standby	**Hot standby -** Automatically synchronize data Have a standby ready to pick up load Use automatic failover from master to standby
Very small data loss (RPO - 1 minute) BUT I can tolerate some downtimes (RTO - 15 minutes)	**Warm standby** - Automatically synchronize data Have a standby with minimum infrastructure Scale it up when a failure happens
Data is critical (RPO - 1 minute) but I can tolerate A downtime of a few hours (RTO - few hours)	Create regular data snapshots and transaction log Create database from snapshots and transactions logs when a failure happens
Data can be lost without a problem (for example: cached data)	Failover to a completely new server

Consistency

- How do you ensure that data in multiple database instances (standbys and replicas) is updated simultaneously?
- **Strong consistency** - Synchronous replication to all replicas Will be slow if you have multiple replicas or standbys
- **Eventual consistency** - Asynchronous replication. A little lag - few seconds - before the change is available in all replicas
  In the intermediate period, different replicas might return different values Used when scalability is more important than data integrity

  Examples : Social Media Posts - Facebook status messages, Twitter tweets, Linked in posts etc
- Read-after-Write consistency - Inserts are immediately available. Updates and deletes are eventually consistent
    - Amazon S3 provides read-after-write consistency

Database

Database Categories

There are **several categories** of databases:

- Relational (OLTP and OLAP),
  - Online Transaction processing database
  - Online Analytics processing databases
- Document database
- Key Value databse
- Graph

Database

Choosing type of database for your use case is not easy. A few factors:

- Do you want a **fixed schema**?
- Do you want flexibility in defining and changing your schema? (schemaless)
- What level of **transaction properties** do you need? (atomicity and consistency)
- What kind of **latency** do you want? (seconds, milliseconds or microseconds)
- **How many transactions** do you expect? (hundreds or thousands or millions of transactions per second)
- **How much data** will be stored? (MBs or GBs or TBs or PBs)

Database -Relational Databases

- Most popular type of database
- **Predefined schema** with tables and relationships
- Very **strong transactional** capabilities
- Used for
  - OLTP (Online Transaction Processing) use cases and
  - OLAP (Online Analytics Processing) use cases

Database -Relational Databases

Relational Database - OLTP (Online Transaction Processing

- Applications where **large number of users make large number of small transactions**
  - small data reads, updates and deletes

- Use cases:
  - Most traditional applications, ERP, CRM, e-commerce, banking applications

- **Popular databases**: MySQL, Oracle, SQL Server etc

- Recommended AWS Managed Service:
  - **Amazon RDS**
  - Supports Amazon Aurora, PostgreSQL, MySQL, MariaDB (Enhanced MySQL), Oracle Database, and SQL Server

Database -Relational Databases

Relational Database - OLAP (Online Analytics Processing)

- Applications allowing users to **analyze petabytes of data**
  - **Examples** : Reporting applications, Data warehouses, Business intelligence applications, Analytics systems
  - **Sample application** : Decide insurance premiums analyzing data from last hundred years
  - Data is consolidated from multiple (transactional) databases

- Recommended AWS Managed Service
  - **Amazon Redshift**
  - **Petabyte-scale** distributed data ware house based on PostgreSQL

Database -Document Database

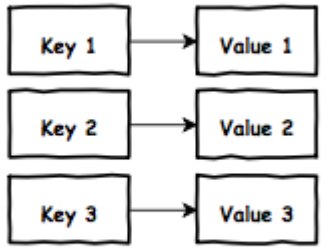
- Structure data the way your application needs it Create one table instead of dozens!
- Quickly evolving semi structured data (schema-less)
- Use cases : Content management, catalogs, user profiles
- Advantages : (Horizontally) Scalable to terabytes of data with millisecond responses upto millions of transactions per second

Recommended AWS Managed Service
Amazon DynamoDB

Database - Key-value Databases

Key-value

- Use a **simple key-value pair** to store data. Key is a unique identifier.
- Values can be objects, compound objects or simple data values
- **Advantages** : (Horizontally) Scalable to terabytes of data with millisecond responses upto millions of transactions per second
- Recommended AWS Managed Service - Amazon DynamoDB again
- Use cases : shopping carts, session stores, gaming applications and very high traffic



Key Value Database

Database

Graph

Store and navigate data with complex relationships
Use cases : Social Networking Data (Twitter, Facebook), Fraud Detection
Recommended AWS Managed Service - **Amazon Neptune**

Database
In-memory Databases

- **Retrieving data from memory is much faster from retrieving data from disk**
- In-memory databases like Redis deliver microsecond latency by storing **persistent data in memory**

- Recommended AWS Managed Service
  - **Amazon ElastiCache**
  - Supports Redis and Memcached
    - Redis is recommended for persistent data
    - Memcached is recommended for simple caches

- **Use cases** : Caching, session management, gaming leader boards, geospatial applications

Database - Databases - Summary

Database Type	AWS Service	Description
Relational OLTP databases	Amazon RDS	Row storage Transactional usecases needing predefined schema and very strong transactional capabilities
Relational OLAP databases	Amazon Redshift	Columnar storage Reporting, analytics & intelligence apps needing predefined schema
Document & Key Databases	Amazon DynamoDB	Apps needing quickly evolving semi structured data (schema-less) Scale to terabytes of data with millisecond responses upto millions of TPS Content management, catalogs, user profiles, shopping carts, session stores and gaming applications

Database - Databases - Summary

Database Type	AWS Service	Description
Graph Databases	Amazon Neptune	Store and navigate data with complex relationships Social Networking Data (Twitter, Facebook), Fraud Detection
In memory databases/caches	Amazon ElastiCache	Applications needing microsecond responses Redis - persistent data Memcached - simple caches

How to connect to RDS Instance from AWS EC2

sudo su
yum -y install mariadb-server
systemctl enable mariadb
systemctl start mariadb

Username : admin
Password :admin
RDS Endpoint : “from the rds instance “

Command to connect
- mysql –h “endpoint: -P 3306 -u admin –p admin

How to connect to RDS Instance from AWS EC2

- **Query to check the databases present**

  - SHOW Databases;