7️⃣

Lecture 7

Missed - EBS

EBS:

- Storage options like this can not be attached to multiple instance at the same time
- Its either one instance or no instace

Availability:

- EBS discs will have replicas in the backed - managed by the cloud providers any reads and writes will be replicated on the replicas. incase of main disk failure they will switch over to standby - we dont pay for this its backed in to the pricing of the EBS

Snapshot:

- this is a instance specific feature. All the cloud vendors usually take 1 snapshot a day. The automatic snapshots provider takes will be deleted when we delete the instance. the one we manually take will not be deleted
- How does snapshot work. for a disk of 10 gb. for 1st snapshot the size will be 10 gb. from second snapshot onwards they just save the delta.

Encryption:

- history: everyone was under the impression that all the internal network are secure. So, they did not encrypt the internal traffic.
- So, now all the internal traffic data is encrypted
- Example: Block storage

Instance store (AWS instance store):

- When EBS is attached to a VM. There is a physical distance between two instances - this is usually the default option
- What if we need a local storage at the physical location of the VM itself
- This instance stores are temp. storage in case of VM failure you lose the storage as well
- we will use them situations of caching
- one drawback of this is that we dont get all the type of instance that comes this way with the storage

  Normal storage with VM:

  - we are actually buying two things (build and located separately connected by network)
  - The VM
  - The Storage
  - when we stop the compute we dont pay for the VM, we only pay for the storage
- In instance store we are buying it together - Validate it
- This disk is not persistence. terminate the instance lose the data
- we have to work with the assumption that we are going to lose the data at any given point
- There is no Encryption, no failover
- **We should we able to compare all storage option for mid-term, there will be one question**

Unique offering from AWS:

- Google does not have it - they have something called “Filestore” which is white labeled from Netapp
- Elatic File System - why did Amazon decide to build it and google din’t. Amazon is cloud leader - they would have to support legacy apps for big orgs.
- we can have a network share - attached to storage and create a symlink and attach to multiple server
- WE needed some storage which could connect to multiple instance
- in the early - startup - building new stuff
- but legacy companies - they have proceees in place and critical stuff which can not be messed around
- EFS is still a virtual storage. EFS is elastic when we create a storage dont need to specify the storage. it will grow as we add data and shrink as we remove data
- Google requires a minimum - because in File-store - reselling third party service - they have to makeup for the cost. 1TB can be expensive
- EFS is regional, unlike other storage which are zonal
- Limitation - we cannot run OS from this storage, this is always a secondary storge system
- This can only be attached from onside the VM and not from the commandline

  EFS use case:

    - Movies - video editing - media processing
- Google Filestore is really expensive than AWS - There is lot of complaints about EFS being really slow than EBS (he just read about it)

Object storage service (Amazon S3) | Google cloud storage | blob storage

  - when we want to store data. in traditional way - we will VM and then storage attached to it. which means this App machine will have to take in request and respond to request in lare scale this is going to get really expensive - here comes Amazon S3 - its all managed - cloud made object storage popular
  - The operational expertise was not present in the past - hence it was not popular - it also gets expensive if we had to hire and manage it
  - with cloud we are outsourcing a lot of things - hence things get cheaper initially - as we scale we will be paying a premium
  - BigQuery - data warehousing solution - this will spin up as many servers need to process the query
    - i.e the cost of the process was 14k - with SELECT *; query has run
  - Some might complain - cloud is expensive - but, they provide reliability, they manage the operational overhead
- if we are working on a web application with any kind of files we use S3 instead of EFS reason the requests are handled by their server and not by us
- Network charge, no of request charge, bandwidth usage charge - they make money by charging for bandwidth
- Google equaling is cloud storage - every object has some metadata and globally unique identifier
- The bandwidth is not charged for metadata, we only get charged for the files
- why call it object? - can be anything .pdf, .bin etc...
- Highly available / highly durable
- Reliability - less confident on Azure then other two platforms
- The difference is between other storage systems and this that we are not working with an operational system. we dont need a VM to work with it. we will handle everything through - http
- WE CANNOT MOUNT S# or Google cloud storage - some third party will make it seem like mounting it - but its http is the background

Data consistency Model for S3

- eventual consistency vs consistency
- Objects are immutable - meaning when ever we updating a file we are just replacing it - although it might seem like we are updating
- This is a very important property they are talked about in the docs
- Amazon in the past had eventual consistency - but they have moved to total consistency
- But google came with strong consistency from day 1

  How to turn eventual consistency to consistency:

  - instead of updating we just add a new version and update it in the database this is the latest version
  - The first obejct will will strongly consistent
- Doesn’t the strongly consistent model cost more?
  - The overhead is that we have to manage versions
  - now we have to pay for each versions we have to store
  - we have to track these things in our DB if this did not exist
- Amazon makes us stay for Standard tier for 30 days (just a fact)
- Use case:
  - Tiktok - new videos tend to be accessed more - So , we need it select is tier that suits our need (standard)
    - but when the videos get older (we can move it to less superior tier)
- Refer - slite (34) for types of S3 storage classes
- How do they actually save money
  - because - they make the disk inactive - So that the wear and tear on this disks are less, so they are just passign the savings they made on infrastructure

S3 Buckets:

Bucket Logging:

- it will take a while to even figure out the attack is happened
- logging will help us get a trail of what actually happened and derive insights - Logging is not free - we need to store data

Bucket Event Notification:

- we can set up events to happen when a data is uploaded to it
- When we upload a photo in cloud they will make derivatives out of it - smaller image and all
- this notification will help them generate this

Bucket versioning

- we pay for storage of each version, by default versioning is not enables with google or amazon, if we switch it on. from then on only they are versioned

Data lifecycle management:

- Example: look at logging solutions - lot of the persistent storage systems are reliaing on object storage system as backup. now we have gotten to a point where we can scale object stores.
- They cloud storage lets us define lifecycle policies - like after x days move it to y storage and after z days delete it.
- we can move petabytes of data just by writing a policy

Object Tagging:

- we can add tags to objects

Cross-region-replication:

- we can activate this on amazon this will Asynchronously replicated to other regions. we pay for bandwidth, two times the storage

Querying S3:

- we can search with a name and it will not give results. we have to track where our object is in our db.

S3 is not suited as a file system - explained above

Relational Databases:

- Based on mathematical concepts of relational algebra. That why we feel that, Mongo seem so easier
- Mongo does have a schema which is managed by it
- but in RDBMS we have to manage it
- There is a lot of options for relational databases
- cloud providers: Some providers have their own feature set
- All the RDBMS have in common is Transactions - ACID
- if something does not support ACID transaction then its not a relational DB
- Automicity:
    - its either the whole transaction is complete else, it should not change it at all. This is called Automicity
    - When a transactoin is commited it should stay commited (powerfailute, VM crash anything)
- Consistency:
    - The DB must remain consistent
- Isolation:

    - even if there are 10 transactions running in parallel. it shoud be like they are running sequentially
    - Deadlock
    - This character dont not only vary by which tech (mysql, psotgres) but also the kind of storage we use.
    - the granularity in which we lock says how many transactions we can do in a unit time
- Durability:
    - Whats committed stays committed
    - How is this ensured?
        - Write head logs - as we commit a transactions we log the transaction - the DB cannot wait to write in disk - we create a write ahead log - as soon as we restart the db. it check the write ahead log. it will replay the transactions to make it Durabale
        - when we back up backup storage and write ahead log
        - in disaster recovery - the write ahead logs are also used in replicas
- Automated backups of AWS RDS
    - one of the feature this has is that. we have point in time recovery. lets say we took a snapshot at 6:00AM morning - we have a problem at 6:00PM we are not okay with 12 hr data loss
    - it is going to const in money. there is a lot of value in this
    - we can als otrigger a backup ourself - they bring up the VM. they launch the instance and copy the instance back it up and put it in a object store. run it once a day
        - Automated Backups
        - Database Snapshots

Advantages of AWS RDS:

Restoring RDS from backup:

it will always be a new instance - no matter the platform - we wil end up with a new host

Amazon RDS Multi-AZ Deployment:

- we can write to single one and read from other ones
- The only one form scaling in relational db is vertical scaling
- we can have one instance that is active and one will be on standby
- The standby is setup to sync with the active instance
- as long as the standby is in sync with the active one. (sometimes standby will not be able to keep up) - as long as its not too far from the active - we will perform DB things on standby
- Upgradeing stuff on active - make the standby active, make the active standby then make the update on the new standby
- In old days - we had this 80:20 rule 80% reads , 20% writes
- now: 95% reads 5% writes

Amazon RDS read replicas:

- we will many standby with read replicas in sync with your active. we can use it to read data from there
- What about lag in sync. this does not happen when we insert date we fetch data from actual instance - the DB has a hint of whats lacking
- There can be instance where the replicas can go out of sync - they will taken out of comission
- There are other scaling technics - we will shard the DBs they will have clusters, the stored data is also split in between them
- Facebook runs on mysql (on their own customized verison)

Value of Relational DB:

- we can remove the requirements of RDS, the throughput increases

Not so good side of RDBMS:

- after some point it wil lnot care about the index. in order to normalize data, they might split in to differ table
- mismatch on how data is stored and used

We had NoSQL db's forever, its just now we have a terminalogy

Whats a key value store?

- we need to base64 encode it and store it why - > we can get a string representation of the DATA

Graph datasets?

- emereged from to see how people are connected
- Whats 6 degree of seperation?

Document database?

NoSQL database are schema free?

they are, but, now we manage the schema in the appication code

CAP Theorem:

- Choosing Availability than consistency:
  - Social medai - likes - can be late

    - online buying - Amazon - if we could track what people are looking at their activity - all this data is useful for insigts like where to place the item and button. This ddata has no replationship. if we lost 50 data points it dones ot matter. this is why nosql dbs become popular
- Choosing Consistency over Availability:

Eventual consistency:

Amazon when we create the RDS instance, and delete it you can do it back t back

Google when we create a RNDS instance and delete it. google will ask you to wait 7 days - becuse of eventual consistency

example for comcast record show

Normalization and denormailization:

- Normalization become not so important because storage become really cheap
- in case of nosql dedenormailization does not make any sense because, there are not joins data is just there in one part

A storage system can be in two categories:

- Source of truth:
    - self explanatory
- derived data system
    - maybe a derived DB to support search function
    - can put the data in elastic search and implement searh here. if something goes south. we can rebuild it from source of south
    - The pressure on derived is so, less because its replaceable

    - i.e Amazon when a listing is posted it will take 15 min to get updated - eventual consistency

Polyglot Persistence:

- monlith | micro services
- The advantage of micro srevices is the language can be different only the medium of communication should be the same - same goes for DBs
- we can have NoSQL and SQL dbs working in a system

Concurrency control mechanisms:

- optimistic
    - dont lock, just go for it
    - wil give you more throughput
    - before we commit a transaction we check if any change has happed, if yes then we roll back.
- pessimistic
    - Lock it then make the change

Write-write conflict (NoSQL):

- it will just indicate it we nned to handle it

Read-write conflict:

- Ticket booking - seats all gone - very common conflict

Version stamps:

MVCC (POSTGRES)

- create a new version and then clear out the old version (Async)

- if the vaccum process is slow - we run out of transaction ids (we just have to give postgres time to clean up postgres)

, in place replace (MYSQL)

Single Server (Distribution model):

- there is alimt to how much we can store in the Db while still maintaing performance.
- Transactions are limited to clusters
- Spread loads accross cluster
- we have to find the right sharding key to actually shard data equlally

Primary/Secondary Nodes Replication:

Peer to Peer Replication:

- every node is treated equally
- if one dissappres others will be in charge
- depends on how much duplication we have. if have more we can deal with multui node failure

What is sharding key?

Assignment 5:

- we will use cloud sql instance
- create one host name, and all pass it on to the application, server using terrafrom
- Then make the application use the fail

- remove the local DB installation

terraform:

cloud sql instance

then use the calues and create the env file as a part of launching the applicaiton (start up script) - look up for cloud init

in google this startup script is run every time the system reboots but Amazon only does it once

make sure the script accounts for the fact and idempotent way

you start with no file existing , does this file exist no? create it else dont create it

for now this can be available in plain text

is my script excecuted?

check var/log/

tail secure

tail messages

startup=yes

create cloud sql instace - enterprise - multizone deployment - select privete IP and enable IP, when we setup VPC enable private_access_enable

when we have out network servies setup

then set up cloudsql instance

disable delete protection

storage - 10gb

no need automatic storage

random-name for db instance

same randoem-usrename

random -password

to debug, we will ssh into app server and then talk with the Db server

we can create an empty file

remove the DB which is aprt of the packer