1. What is Amazon Athena ? What is the usage ?

Amazon Athena is an interactive query service that makes it easy to analyze data directly in Amazon Simple Storage Service (Amazon S3) using standard [SQL](https://docs.aws.amazon.com/athena/latest/ug/ddl-sql-reference.html). With a few actions in the AWS Management Console, you can point Athena at your data stored in Amazon S3 and begin using standard SQL to run ad-hoc queries and get results in seconds.

Athena is serverless, so there is no infrastructure to set up or manage, and you pay only for the queries you run. Athena scales automatically—running queries in parallel—so results are fast, even with large datasets and complex queries.

Athena is Query service(and not processing engine(emr) or DW(redshift/Bigquey).

What Athena compares to other clouds?

Athena is only for selects , We might not able to update/delet the records .

1. When to Use Athena ?

For quick adhoc query analysis connecting to S3/Hive metastore. Athena is serverless and connect to s3 directly and doesn't need to transfer data from s3 to Athena.

<https://sonra.io/snowflake/comparing-snowflake-cloud-data-warehouse-to-aws-athena-query-service/>

1. Which is mostly better EMR or EC2+(Hbase,Spark) . When you will use EMR and when you will go to EC2 and install software scnorios?
2. How Athena is Different from EC2+Presto(over s3)?
3. What is Presto or PrestoDB?

Presto (or PrestoDB) is an open source, distributed SQL query engine, designed from the ground up for fast analytic queries against data of any size. It supports both non-relational sources, such as the Hadoop Distributed File System (HDFS), [Amazon S3](https://aws.amazon.com/s3/), Cassandra, MongoDB, and [HBase](https://aws.amazon.com/emr/details/hbase/), and relational data sources such as MySQL, PostgreSQL, [Amazon Redshift](https://aws.amazon.com/redshift/), Microsoft SQL Server, and Teradata.

Presto can query data where it is stored, without needing to move data into a separate analytics system. Query execution runs in parallel over a pure memory-based architecture, with most results returning in seconds. You’ll find it used by many well-known companies like [Facebook](https://code.facebook.com/projects/552007124892407/presto/), [Airbnb](https://medium.com/airbnb-engineering/airpal-a-web-based-query-execution-tool-for-data-analysis-33c43265ed1f), [Netflix](https://medium.com/netflix-techblog/using-presto-in-our-big-data-platform-on-aws-938035909fd4), [Atlassian](https://www.youtube.com/watch?v=0vdW1ORLWyk&feature=youtu.be&t=20m58s), and [Nasdaq](https://www.youtube.com/watch?v=LuHxnOQarXU&feature=youtu.be&t=25m13s).

A single **Presto query can** combine data from **multiple sources**.**Presto** offers connectors to data **sources** including files in HDFS, AWS S3, Azure Blob/ADLS, Google Cloud Storage, MySQL, PostgresSQL, SQLServer, Oracle, AWS Redshift, Snowflake, BigQuery, Cassandra, MongoDB, Redis

Amazon Athena as well connects to different DB within same query.

1. Difference between S3 and Cloud SQL? Advantages and disadvantages ?

Table

Description automatically generated

**AWS S3 pricing:**

* S3 costs $0.023/GB;
* For replicating data across multiple regions costs $0.046/GB, plus a $0.01/GB transfer fee.
* AWS’ cool storage service, S3 Infrequent Access (IA) $0.0125/GB

**Cloud storage pricing:**

No doubt Google pricing is cheaper than AWS is many aspects.

* Single-region storage costs $0.02/GB
* Multi-region cost is $0.026/GB, with free transfer of data
* Cool storage platform Nearline costs $0.01/GB
* Cold/archival product Coldline costs $0.007/GB

Moreover, different case studies prove that for the same solution Google Cloud Storage costs 35% less than AWS S3.

Hence, concerning pricing, no doubt Cloud storage has a competitive edge over Amazon S3.

|  |  |  |
| --- | --- | --- |
|  | S3 | CLOUD STORAGE |
| PERFORMANCE |  | bit faster for uploading huge files |
| service Model |  | the indexing pattern of Amazon S3 and Cloud storage is quite different. Due to this, users may face the hotspotting issue for high throughput use cases in Amazon S3, which is not a case for Cloud storage. |

1. When to use S3 or EBS or EFS?

Fixing the comparison:

* S3 is a storage facility accessible any where
* EBS is a device you can mount onto EC2
* EFS is a file system you can mount onto **several EC2 instances at the same time**

At this point it's a little premature to compare EFS and EBS- the performance of EFS isn't known, nor is its reliability known.

Why would you use S3?

* You don't have a need for the files to be 'local' to one or more EC2 instances.
* (effectively) infinite capacity
* built-in web serving, authentication

**Usecases:**

* Whenever we have thousands of instances who needs to process file simultaneously EFS is recommended over S3.
* Also note that S3 is object based storage while EFS is file based it implies that whenever we have requirement that files are updated continuously (refreshed) we should use EFS.
* S3 is eventually consistent while EFS is strong consistent. In case you can't afford eventual consistency, you should use EFS

1. When to use Amazon Glacier ?

For Backup and Archives . Storage Cost is less

Retrieval/Read from Glacier is really Slow

DESIGN QQ-

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  | Message queues example -   1. Amazon SQS 2. Kafka 3. RabbitMQ |
|  | <https://www.educative.io/module/lesson/web-application-architecture-101/3j5kxZQxXRr>    What is tier ?  Think of a *tier* as a logical separation of components in an application or a service. And when I say separation.      Diagram  Description automatically generated |  |
|  |  |  |

Different Architecture -

Event based Architecture -

Diagram

Description automatically generated

Diagram

Description automatically generated

<https://www.educative.io/module/lesson/web-application-architecture-101/g70G6wjyn7r>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  | Data Mesh - Decentralized  Data Lake - Centralized      **DATA MESH** is a relatively new concept that became one of the fastest-growing trends during 2020. It extends the paradigm shift that was introduced by the microservices architectures and applies it to data architectures, enabling agile and scalable analytics and machine learning or artificial intelligence.  **DATA MESH** provides an alternative to the “centralized” organizational and architectural pattern of the data lake with a distributed and decentralized architecture designed to help enterprises to:   * Enable agility and business scalability * Reduce the time-to-market of the business initiatives * Lower maintenance costs * Allow a fair and transparent internal cost allocation. | |  |  |
|  |  |  | Difference between Data Lake and DWH    Data Lake - ELT (original format, structured, unsctructured, No schema)  DWH - ETL. (schema,structured , Data modelled) .         * + Data Lake stores all data irrespective of the source and its structure whereas Data Warehouse stores data in quantitative metrics with their attributes.   + Data Lake is a storage repository that stores huge structured, semi-structured and unstructured data while Data Warehouse is blending of technologies and component which allows the strategic use of data.   + Data Lake defines the schema after data is stored whereas Data Warehouse defines the schema before data is stored.   + Data Lake uses the ELT(Extract Load Transform) process while the Data Warehouse uses ETL(Extract Transform Load) process.   + Comparing Data lake vs Warehouse, Data Lake is ideal for those who want in-depth analysis whereas Data Warehouse is ideal for operational users. |
|  |  |  |  |  |
|  |  | |  |  |
|  |  |  |  |  |

System Design -

<https://www.educative.io/courses/grokking-the-system-design-interview/B8nMkqBWONo>

1. - requirement gatherings?

1. Back-of-the-envelope estimation

What scale is expected from the system?

How much storage will we need?

What network bandwidth usage are we expecting?

1. System interface definition ?

Define what API's are expected from the system?

1. Defining data Model ?

Defining the data model in the early part of the interview will clarify how data will flow between different system components

1. High Level Design -

1. Detailed Design?
2. Identifying and resolving bottlenecks?

* Is there any single point of failure in our system? What are we doing to mitigate it?
* Do we have enough replicas of the data so that we can still serve our users if we lose a few servers?

DOCKER:

<https://www.quora.com/What-is-the-difference-between-containerization-Docker-and-virtualization-VMWare-VirtualBox-Xen>

<https://stephen-odaibo.medium.com/docker-containers-python-virtual-environments-virtual-machines-d00aa9b8475>

<https://www.quora.com/Can-you-run-containers-for-different-OSes-on-a-host-with-a-different-OS>

<https://www.educative.io/collection/page/10370001/5534036859551744/5822825091825664>

Containers:

Containers are as the name implies. They are small sealed units, generally defined in software, and normally running inside of a cloud server instance.

Imagine A street of houses, each house having it’s own garden, and around each garden is a massively high fence, so high that the occupants of each house have no knowledge of who or what is in the house either side of them.

Now imagine that each house is a container, and the street is your operating system.

The only person that’s allowed into the street is the postman, so the only way that information can be exchanged between the outside world, is via this postman.

If one house want’s to talk to another, the message has to be given to the postman, who then processes it and makes sure it gets delivered to the correct house.

Each house however, is connected to the same water supply, and the same energy supply, both of which are supplied by the street supply.

So each “Container” shares resources supplied by the “Operating System” which is just a normal operating system, running on a normal definition of a computer, a computer which might be virtual, or might be physical, it might be on some one else’s infrastructure in a cloud provider, or it might be plugged in, in the room right next door to you.

The benefit of this segregation, is the shared resource. Instead of having 10 machines, one for each application, you have one machine running 10 containers.

Since most applications don’t run at 100% usage, 24 hours a day, most modern hardware can easily accommodate several of these containerized boxes, that each run one instance of a given application.

Making best use of one bit of hardware, means your costs are lower, but your return on investment is higher.

[**an containers only be installed on a matching host operating system?**](https://www.quora.com/Can-containers-only-be-installed-on-a-matching-host-operating-system)

The container and host OS need to have the same *kernel*. **However**, there are workarounds to this.

Let’s understand what is really going on behind the scenes.

* + When you run a container on any host OS, the container uses the *kernel* that the host OS provides.
  + For example, if you have Ubuntu as your host OS, you have the linux *kernel.* So your container image can be anything that *uses the linux kernel*.
  + You can have Arch, CentOS - any linux distro for that matter.
  + Containers just provide the file system representation - they do not have any kernel specific patches.

Coming to your question, yes, it is possible to run Linux containers on Windows (you can extrapolate this logic to macOS too).

* + If your host OS is Windows, Docker will spin up a Linux VM for you and run containers in this VM.
  + But instead of using Virtual Box, the Linux VM is run using Hyper-V - a Windows-native hypervisor. You can open Hyper-V Manager and see that a Linux VM is running.
  + There is a docker daemon running in this VM which provides the required features to run Linux-based containers.
  + Also, if you install Docker for Windows and run docker version you'll see that the Docker Linux daemon is running on "Moby Linux".

Similarly, you can even run Windows-based containers on Windows virtual machine which is running on a host macOS or Linux-based OS.

Hope this clears it for you!

DATA MODEL –

**What is Snowflake ? When you will use it .**

1) When the Dimension attributes are normalized to a separate table it its called snowflake.

2) Snowflake helps in the space constraints in Datawarehouse.

3)

**Why to have a different Fact table ?**

When the grain of the Fact table is different . We can't have monthly/Weeky/Daily Level snapshot in the same table.

**Why we need the Bridge table ?**

In general Case, the Dimension and FACT is 1: Many relation (one dimension can have multiple facts).

The Bridge table is needed when Dimension and FACTS are in Many to Many relations . Example . Same order facts is done by 2 salesperson(customer). To Avoid duplicate counting , we need bridge table.

**Disadvantage of Snowflake ?**

* Normalized design . Too much joins .
* Advantage - Space reduction.

**What is FACTLESS table and uses ?**

When a fact table doesn’t have any facts and only dimension keys. This is useful when you want to count the events(transactions) at defined level. Mostly useful for Counts(aggressions).

**What is Degenerate Dimensions and Uses?**

When a dimension (order\_line\_id, Customer\_purchase\_id) is placed inside the fact table is called degenerate dimensions.

Advantages ?

Avoid Joins

Disadvanatges?

increase the size of fact table

What is Junk Dimensions?

Junk dimensions are used to reduce the number of dimensions in the dimensional model and reduce the number of columns in the fact table.  A junk dimension combines two or more related low cardinality flags into a single dimension. An example of this may be car color (red, black, blue, etc.) and body style (sedan, van, SUV, etc.) As you can see these are limited in number and, if created as single dimensions, the dimensions would be limited to a single attribute. In order to eliminate these small dimensions, we create a single “junk” dimension which cross joins all possible attributes into a single dimension which will be used in the fact table.

Advatage - Less Join and decrease storage . Easy to main.

Disadvanatge - only can be done on low cardinality dimensions.

What is a ESL(enterprise semantic layer) and uses ?

ESL - > Sematic layer where the business uses have direct access and not to the underlaying datasets.

Example , If you want to Join 2 fact tables(from different teams sales, marketing) and provide a needed columns to the users . You do a ETL for the sematic layer and expose to the needed columns to the views.

Exposing columns to the views will be different for each teams.

**How ESL is different from providing views?**

Again ESL will be maintained in typ2(similar to other warehouse). Views maintainence will be more and might not accommodate for longer run.

What is the advantage of Aggregates ?

**Advantage -**

* Pre compute the processing(one time) , Users fetch will be faster. Imaging the time taken to performe the Aggregates each time the Analyst need to run.
* BL tools love to have aggregates done at Database level.
* Aggregates can be done at daily/weekly/Monthly level.0

**Disadvantage -**

Maintenance and extra space. Should be decided based

DATA ARCHITECTURE –

Things to consider for Big data architecture ?

<https://www.shreyapal.com/post/designing-bigdata-solution-part-1> - Good.

<https://www.shreyapal.com/post/designing-bigdata-architecture-part-2>

Common Use cases:

* + **Data archival/ Data Offload** – Despite the cumbersome process and long SLAs for retrieval of data from tapes, it’s the most commonly used method of backup as the cost prohibits the amount of active data maintained in the current systems. Alternatively Hadoop facilitates storing huge amounts of data spanning across years (active data) at a very low cost
  + **Process Offload** – Offload jobs that consume expensive MIPS cycles or consume extensive CPU cycles on the current systems.
  + **Data Lake Implementation**– Data lakes help in storing and processing massive amounts of data
  + **Unstructured data processing** – BigData technologies provide capabilities to store and process any amount of unstructured data natively. RDBMS’s can also store unstructured data as BLOB or CLOB but wouldn’t provide processing capabilities natively.
  + **Datawarehouse Modernization** – Integrate the capabilities of BigData and Datawarehouse to increase operational efficiency.

* + **Vendor selection -**

This is important , most vendors offers same functionality with slightly different cost structure . Personal bias.

* + **Deployment strategy -**

On premise -

Advantage - Secured

Disadvantages - High Cost and maintenance.

Cloud -

Low cost

Deployment strategy solely depends on the company . Hybrid mix where to retain the PII data on premise and other in Cloud (this would be difficult to maintain and might increase the network latency).

* 1. **Capacity Volume -**

Daily Data load

retention period

replication factor

* 1. **Data Source Identification -**

type of data source – Database, File, web service, streams etc.

Determine the type of data – structured, semi structured or unstructured

* 1. Data ingestion -

Is there any data validation or transformation required before ingestion (Pre-processing)

Batch Data ingestion -

Real time Data ingestion -

**Data Storage -**

Hdfs

No sql store

Synchronous Storage -> Batch Analytics(hdfs storage)

Asynchronous Storage -> Realtime Analutics (no sql storage)

Compression requirement -

**Conclusion**

The two biggest challenges in designing BigData Architecture are:

* + **Dynamics of use case**: There a number of scenarios as illustrated in the article which need to be considered while designing the architecture – form and frequency of data, Type of data, Type of processing and analytics required.
  + **Myriad of technologies:** Proliferation of tools in the market has led to a lot of confusion around what to use and when, there are multiple technologies offering similar features and claiming to be better than the others.

To be nimble, flexible and fluid in our approach to a solution we need to use these technologies in a way that will adapt to the changing business and technology dynamics.

Graphical user interface, text, application

Description automatically generated

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | How to Scale the writes DB?  Sharding    Sharding Techniques -  Hashing and sending to the right note.      Disadvantage in distributed writes(sharding)  Data Skewness | |  |
|  | Diagram  Description automatically generated | | |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  | Diagram  Description automatically generated | |
|  |  |  |  |  |  |  |  |
|  |  | Why we need read replica ? |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Diagram

Description automatically generated  
A picture containing text

Description automatically generated  
Graphical user interface, text, application

Description automatically generated  
Diagram

Description automatically generated  
Graphical user interface, text, application, email

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  | Database Challenges  <https://aws.amazon.com/caching/database-caching/>      When building distributed applications that require low latency and scalability, there are a number of challenges that disk-based databases can pose to your applications. A few common ones include:   * Slow processing queries: While there are a number of query optimization techniques and schema designs that can help boost query performance, the data retrieval speed from disk plus the added query processing times generally will put your query response times in double-digit millisecond speeds, at best. This assumes you have a steady load and your database is performing optimally. * Cost to scale: Whether the data is distributed in a disk-based NoSQL database or vertically scaled up in a relational database, scaling for extremely high reads can be costly and may require a number of database read-replicas to match what a single in-memory cache node can deliver in terms of requests per second. * The need to simplify data access: While relational databases provide excellent means to data model relationships, they aren’t optimal for data access. There are instances where your applications may want to access the data in a particular structure or view to simplify data retrieval and increase application performance.   Before implementing database caching, many architects and engineers spend great effort in squeezing as much performance as they can out of their database. And while doing so with reasonable expectations is great, **it’s counterproductive to try and solve a problem with the wrong tools.** For example, say you are trying to lower the latency of your database query, doing this with reasonable expectations is a best practice, but trying to defy the laws of physics associated with retrieving data from disk is a waste of time. |
|  | CDN uses - Static pages.    <https://lethain.com/introduction-to-architecting-systems-for-scale/>        Diagram  Description automatically generated |  |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  | How Caching Helps  A database cache supplements your primary database by removing unnecessary pressure on it, typically in the form of frequently accessed read data. The cache itself can live in a number of areas including your database, application or as a standalone layer.  The three most common types of database caches are the following:   * **Database Integrated Caches:** Some databases such as Amazon Aurora offer an integrated cache that is managed within the database engine and has built-in write-through capabilities. When the underlying data changes on the database table, the database updates its cache automatically, which is great. There is nothing within the application tier required to leverage this cache. Where integrated caches fall short is in their size and capabilities. Integrated caches are typically limited to the available memory allocated to the cache by the database instance and cannot be leveraged for other purposes, such as sharing data with other instances. * **Local Caches:** A local cache stores your frequently used data within your application. This not only speeds up your data retrieval but also removes network traffic associated with retrieving data, making data retrieval faster than other caching architectures. A major disadvantage is that among your applications, each node has its own resident cache working in a disconnected manner. The information stored within an individual cache node, whether its database cached data, web sessions or user shopping carts cannot be shared with other local caches. This creates challenges in a distributed environment where information sharing is critical to support scalable dynamic environments. And since most applications utilize multiple app servers, if each server has its own cache, coordinating the values across these becomes a major challenge.  In addition, when outages occur, the data in the local cache is lost and will need to be rehydrated effectively negating the cache. The majority of these cons are mitigated with remote caches. A remote cache (or “side cache”) is a separate instance (or multiple instances) dedicated for storing the cached data in-memory.  When network latency is of concern, a two-tier caching strategy can be applied that leverages a local and remote cache together. We won’t discuss this strategy in detail, but it is used typically used only when absolutely needed as it adds complexity. For most applications, the added network overhead associated with a remote cache is of little concern given that a request to it is generally fulfilled in sub-millisecond performance. * **Remote caches:** Remote caches are stored on dedicated servers and typically built upon key/value NoSQL stores such as [Redis](https://aws.amazon.com/redis/) and [Memcached](https://aws.amazon.com/memcached/). They provide hundreds of thousands to up-to a million requests per second per cache node. Many solutions such as [Amazon ElastiCache for Redis](https://aws.amazon.com/elasticache/redis/)also provide the high availability needed for critical workloads.  Also, the average latency of a request to a remote cache is fulfilled in sub-millisecond latency, orders of magnitude faster than a disk-based database. At these speeds, local caches are seldom necessary. And since the remote cache works as a connected cluster that can be leveraged by all your disparate systems, they are ideal for distributed environments.   With remote caches, the orchestration between caching the data and managing the validity of the data is managed by your applications and/or processes leveraging it. The cache itself is not directly connected to the database but used adjacently to it. We’ll focus our attention on leveraging remote caches and specifically [Amazon ElastiCache for Redis](https://aws.amazon.com/elasticache/redis/) for caching relational database data.  To learn more about Caching Patterns, please visit the [Implementation Considerations Page.](https://aws.amazon.com/caching/implementation-considerations/) |
|  | What is the relation between CPU,RAM and DISKS.    Diagram  Description automatically generated         * + ***CPU*percentage**: to show how much percentage of **processor is**being used.   + ***Memory*percentage**: to show how much percentage of inside memory (**RAM**) is being used.   + ***Disk*percentage**: to show how much percentage of **storage**(HDD, SSD) is being used. |  |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Graphical user interface, application  Description automatically generated    Graphical user interface, application  Description automatically generated  **Shared Everything –** A single box that has CPU, local memory and local disk.  **Shared memory architecture –** One alternative in distributed environment is shared memory architecture. There are multiple CPUs on different machines but memory is shared across. There is a unified memory view. Each CPU has global view of all in memory data structures. CPU can access memory address space via a fast interconnect.  Shared memory architecture is expensive and Increase in memory, CPU, disk is not directly proportional to the load that can be handled. It may also give limited fault tolerance as it’s on a single machine and same geography.  Ex: Mostly seen in HPC world.  **Shared Disk Architecture –** CPU can access a single logical disk directly via interconnect but each can have their own private memory. Execution layer and storage layer can scale independent of each other. This may suffer from contention and locking issues  Ex: Distributed file systems  **Shared Nothing Architecture** – There are a bunch of machines acting as a single machine which have their Independent CPU RAM and disk. All communication between nodes is done using software. This is also called as horizontal scaling.  Shared nothing architecture can be of two types:    **Homogenous Architecture:**  · Every node in the cluster can perform same set of tasks (on different partitions)  · Provisioning and failover is easy  · Ex: Cassandra  **Heterogeneous Architecture**  · Nodes are assigned specific tasks  · A physical machine may have multiple virtual nodes for dedicated task  · Provisioning and failover is not easy  · Ex: MongoDB    Partition type -  Range partition - Range of keys goes to specific partitions (nodes)  Hash Partition - key value is hashed and sent to specific nodes.    Partition **Challenges**:   * + Partitioning can create skews if the partition strategy is not proper   + Hot spot – when one or more partitions get most of the load   + Repartition can lead to data shuffles and high network consumption   + Partition Rebalancing **-**When a node goes down and comes back up or a new node is added, rebalancing is needed. Rebalancing is moving data from one node to another in the cluster. Rebalancing is costly as it consumes network resources. (It can be manual or automatic). |  |  |
|  |  | **What is the Difference between Partitioning and Sharding Database** ?    <https://www.quora.com/Whats-the-difference-between-sharding-DB-tables-and-partitioning-them>    General concept -  **Partitioning ->** Data is subset and partition in the same Db instance (example oracle table partition where the partitioned data uses same memory/same CPU/same resource in the same machine).parttion is kind of shared disk architecture.  Partitioning is a general term used to describe **the act of breaking up your logical data elements into multiple entities** for the purpose of performance, availability, or maintainability.    **Sharding -> Data is partitioned(divided) to different machine (db instance) using hashing techinques (cassandra/teradata).sharding is shared nothing architecture.**  Sharding is a kind of horizontal parttiing .      I take sharding to mean the partitioning of a table **over multiple machines** (over multiple database instances in a distributed database system), whereas partitioning may just refer to the splitting up of a table **on the same machine**.  So a table that is sharded has been partitioned, but a table that has been partitioned has not necessarily been sharded. |
|  |  |  |

What is the needed for System Design Components ?

Chat server example -

Diagram

Description automatically generated

**Components** -------------------

1. No of active users - 500 million users per day
2. Data capacity - 2TB perday(high) 200 GB per day
3. Network Bandwidth per seconds - 25 mb per second - 200gb per day /82600 seconds

**System -**

No of Servers needed - Assume 500 million users and 50K concurrent users are allowed . Then we need 10K servers (500k/50k)

Load Balancer - Load balancer will be in front of the servers, will decide which server they load will go . Load balancer can be round robin/Based on user Location/Based on server loads.

**Data Partitioning-**

Data partitioing (kind of sharding) can be done based on the Hash value (of userid).

Can be done based on the Range partitioning (certain ranges goes to one share ).

Diagram

Description automatically generated

Generally ->

One Physical Db server -> Multiple Db instance (similar server/virtual instance).

What is the advantages of Denormalizations ?

<https://levelup.gitconnected.com/how-to-design-a-system-to-scale-to-your-first-100-million-users-4450a2f9703d>

AWS –

What is Cloud ->

 Early cloud was all about i**nfrastructure-as-a-service ( IaaS)**. That is, the spinning up of storage, compute and networking resources to support startups, dev/test, SaaS and eventually moving more business workloads into the cloud.

If Amazon S3 attracted initial customers to AWS and EC2 instances made them stay, RDS enabled AWS to scale its operations by making customers more sticky to their cloud services.

EC2 – Compute instance

EMR – EC2 + Spark/Hadoop.

EMR is costly the EC2. We ca do set up Hadoop processing with EC2 instance but bit tedious.

One EC2 can have only one EBS(elastic block storage) and if needed multiple then goes to EFS(elastic file system).

Questions -

What is the difference between On demand and Spot instance ?

Spot Instances are spare unused EC2 instances which one can bid for. Once the bid exceeds the existing spot price (which changes in real-time based on demand and supply) the spot instance will be launched. If the spot price becomes more than the bid price then the instance can go away anytime and terminated within 2 minutes of notice. The best way to decide on the optimal bid price for a spot instance is to check the price history of last 90 days that is available on AWS console. The advantage of spot instances is that they are cost-effective and the drawback is that they can be terminated anytime. Spot instances are ideal to use when –

* There are optional nice to have tasks.
* You have flexible workloads which can be run when there is enough compute capacity.
* Tasks that require extra computing capacity to improve performance.

On-demand instances are made available whenever you require them and you need to pay for the time you use them on an hourly basis. These instances can be released when they are no longer required and do not require any upfront commitment. The availability fo these instances is guaranteed by AWS unlike spot instances.

Difference between instance store volumes and EBS volumns ?

**Issue**

I’m not sure whether to store the data associated with my Amazon EC2 instance in instance store or in an attached Amazon Elastic Block Store (Amazon EBS) volume. Which option is best for me?

**Resolution**

[Some Amazon Elastic Compute Cloud (Amazon EC2) instance types](https://aws.amazon.com/ec2/instance-types/) come with a form of directly attached, block-device storage known as the [instance store](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html). The instance store is ideal for temporary storage, because the data stored in instance store volumes is not persistent through instance stops, terminations, or hardware failures.

For data you want to retain longer, or if you want to encrypt the data, use [Amazon Elastic Block Store (Amazon EBS) volumes](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html) instead. EBS volumes preserve their data through instance stops and terminations, can be easily backed up with EBS snapshots, can be removed from one instance and reattached to another, and support full-volume encryption.

* 1. What is AWS Lambda .

Eavery code needs a server to run/execute. Here the context serverless means the application doesn't wan to maintain the EC2 instances Aws Lambda takes care of this.

Lambda is a compute services where you can upload your code and create a Lamdba function AWS Lambda takes care of provisioning and managing the servers that you use to run the code. You do not have to worry about operating systems, patching, scaling etc.

It is essentially described as an event-driven compute service where AWS Lambda runs your code in response to events. These events could be changes to the data in an Amazon S3 bucket or an Amazon Dynamo DB table.

Lambda events can trigger other Lamdba events or call other AWS services like SQS or SNS.

Lambda is also called as 'function as service'.

When Lambda can be used -

To trigger events - event based orchestration.

Advantages of Lambda -

Low cost - calculated based on the milliseconds.

Disadvantages of Lambda -