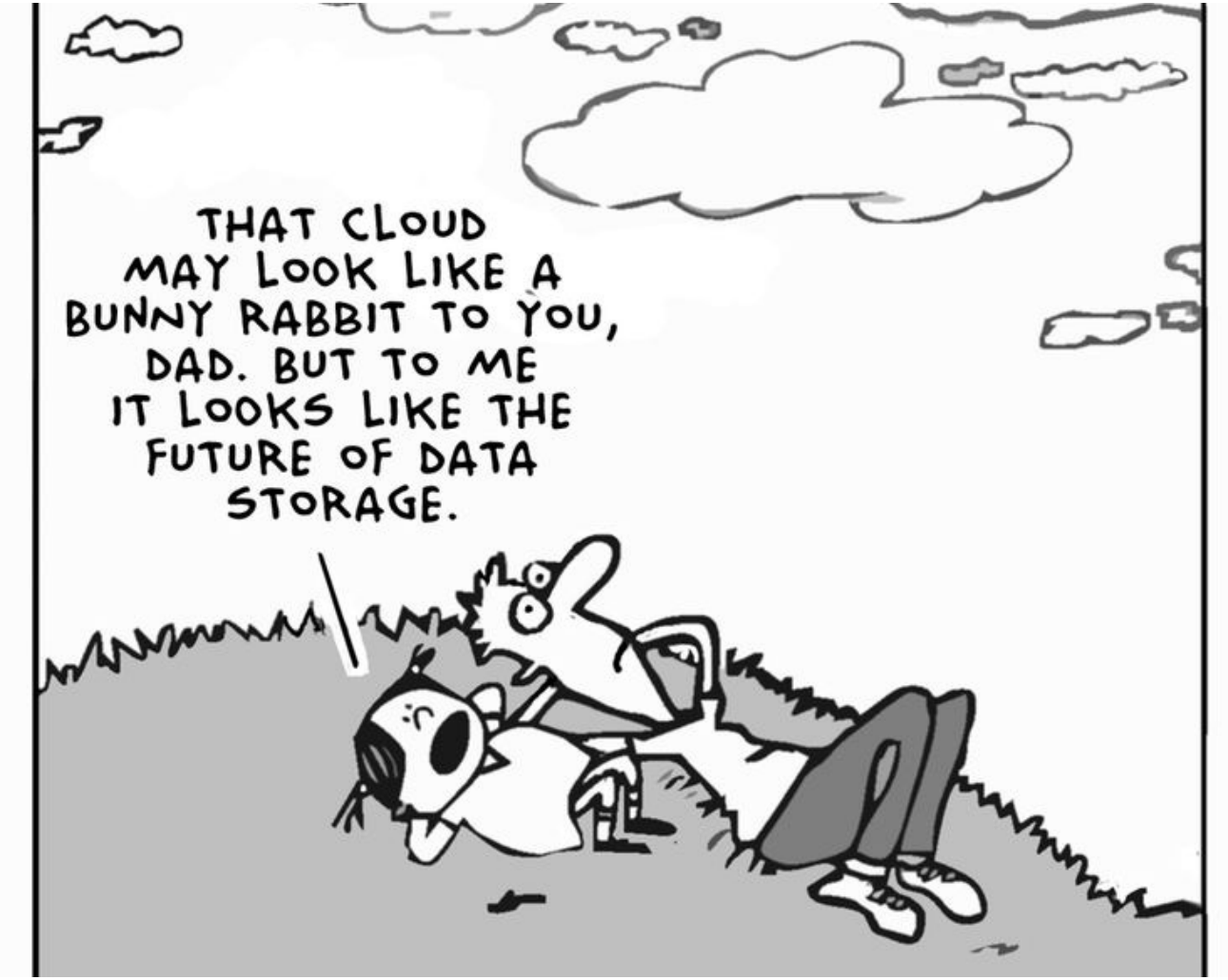
Cloud Storage Solutions

CSYE 6225

Tejas Parikh (t.parikh@northeastern.edu)

Northeastern University



Cloud Storage Options

- Block-level Storage
- Object Storage Service
- Relational Databases
- NoSQL Databases
- (and more)

Block-level Storage

Amazon EBS, Amazon EFS

What is Block-level Storage

- Block-level storage with a disk file system (FAT32, NTFS, ext3, ext4, XFS, and so on) can be used to store files as you do on a personal computer.
- A block is a sequence of bytes and the smallest addressable unit.
- The OS is the intermediary between the application that wants to access files and the underlying file system and block-level storage.
- The disk file system manages where (at what block address) your files are persisted on the underlying block-level storage.
- You can use block-level storage only in combination with an VM (EC2) instance where the OS runs.

Accessing Block-level Storage

- The OS provides access to block-level storage via open, write, and read system calls.
- The simplified flow of a read request goes like this:
    1. An application wants to read the file /path/to/file.txt and makes a read system call.
    2. The OS forwards the read request to the file system.
    3. The file system translates /path/to/file.txt to the block on the disk where the data is stored.

Why do we need Block-level Storage?

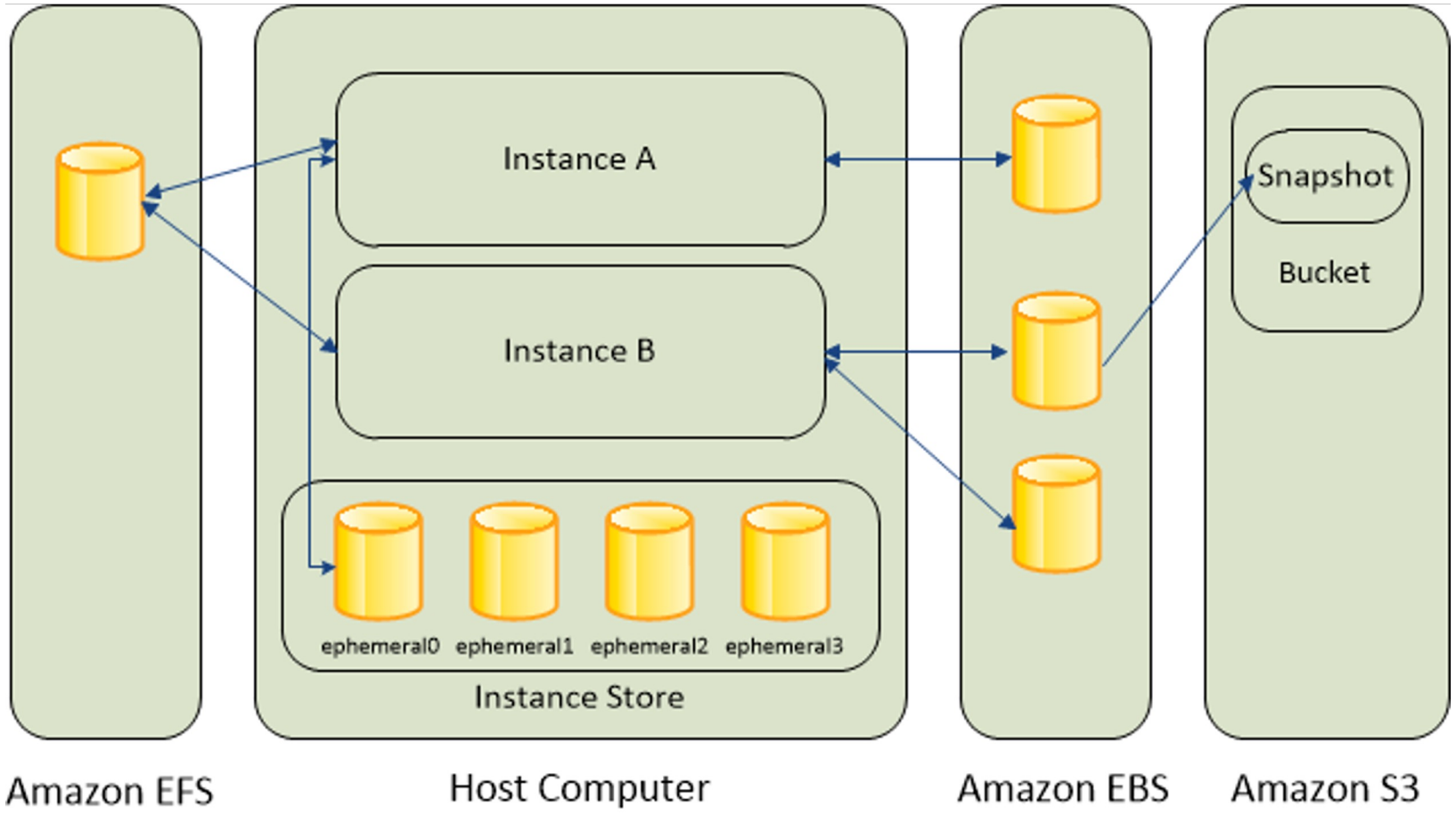
- Applications like databases that read or write files by using system calls must have access to block-level storage for persistence.
- You can’t tell a MySQL database to store its files in an object store because MySQL uses system calls to access files.

Block-level Storage on AWS

AWS provides two kinds of block-level storage:

1. Network Attached Storage (NAS) via AWS EBS service
2. Instance Storage

Amazon Elastic Block Storage



Amazon Elastic Block Store (Amazon EBS)

- Amazon Elastic Block Store (Amazon EBS) provides persistent block storage volumes for use with Amazon EC2 instances in AWS.
- You can think of EBS as disks that you attach to your virtual machine.

Amazon EBS Use Cases

Amazon EBS is an excellent choice for applications that need persistent block storage ideal for databases, data warehousing, big data applications and other low latency interactive applications that demand the highest IOPS or throughput and low latency with consistent, predictable performance.

EBS Lifetime

- EBS Volumes are NOT part of your EC2 instances.
- EBS Volumes are attached to your EC2 instances via a network connection.
- You can choose to terminate EC2 instance without terminating your EBS volumes.
- An EBS volume can be attached to NO EC2 instance or one EC2 instance at a time.
- EBS volumes can be used like normal hard disks.

Availability

- Each Amazon EBS volume is designed for 99.999% availability.
- Each Amazon EBS volume is automatically replicated within its Availability Zone to protect you from component failure, offering high availability and durability.

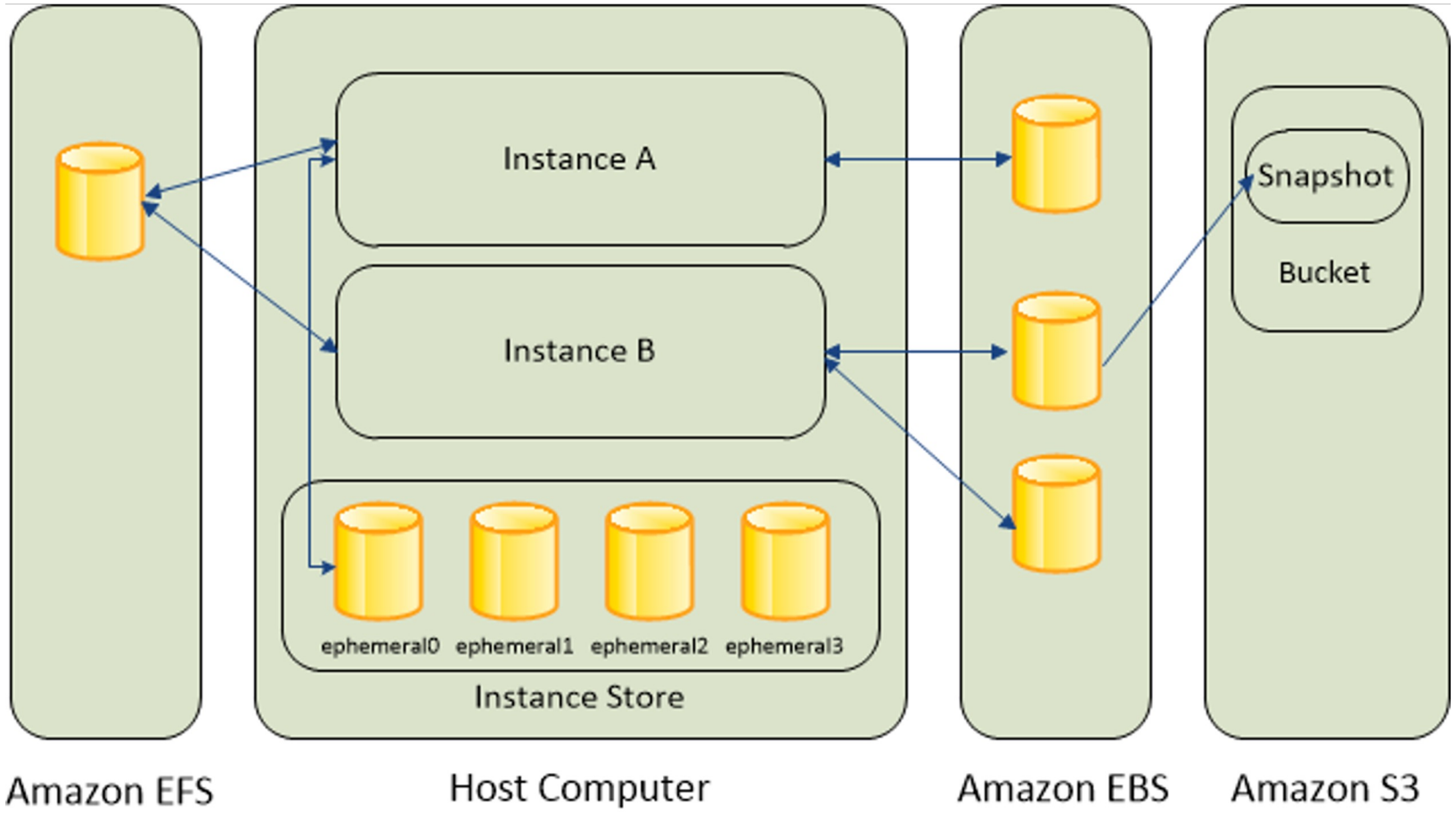
Snapshots

- Amazon EBS provides the ability to save point-in-time snapshots of your volumes to Amazon S3.
- Amazon EBS Snapshots are stored incrementally: only the blocks that have changed after your last snapshot are saved, and you are billed only for the changed blocks.
- Example: If you have a device with 100 GB of data but only 5 GB has changed after your last snapshot, a subsequent snapshot consumes only 5 additional GB and you are billed only for the additional 5 GB of snapshot storage, even though both the earlier and later snapshots appear complete.

Encryption

Amazon EBS encryption provides seamless support for data-at-rest and data-in-transit between EC2 instances and EBS volumes.

Instance Store



Amazon EC2 Instance Store

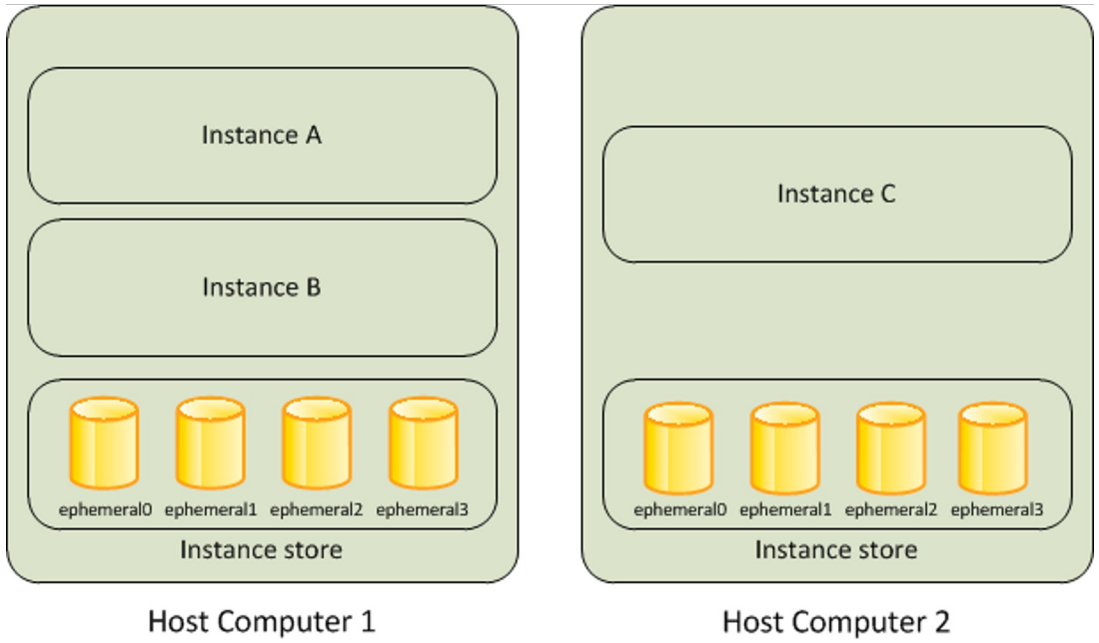
- An instance store provides temporary block-level storage for your instance.
- This storage is located on disks that are physically attached to the host computer.
- Instance store is ideal for temporary storage of information that changes frequently, such as buffers, caches, scratch data, and other temporary content, or for data that is replicated across a fleet of instances, such as a load-balanced pool of web servers.

Instance Store Use Cases

- Caching
- Temporary processing, or
- Applications that replicate data to several servers as some databases do

Instance Store Volumes

- An instance store consists of one or more instance store volumes exposed as block devices.
- The size of an Instance store varies by instance type.
- The virtual devices for instance store volumes are **ephemeral[0-23]**.
- Instance types that support one instance store volume have **ephemeral0**.
- Instance types that support two instance store volumes have **ephemeral0** and **ephemeral1**, and so on.



Instance Store Lifetime

- The data in an instance store persists only during the lifetime of its associated instance.
- If an instance reboots (intentionally or unintentionally), data in the instance store persists.
- However, data in the instance store is lost under the following circumstances:
  1. The underlying disk drive fails
  2. The instance stops
  3. The instance terminates

Instance Store Cost

No additional charge for instance store as instance store is only available with certain instances and instance store charges are included in the EC2 instance price.

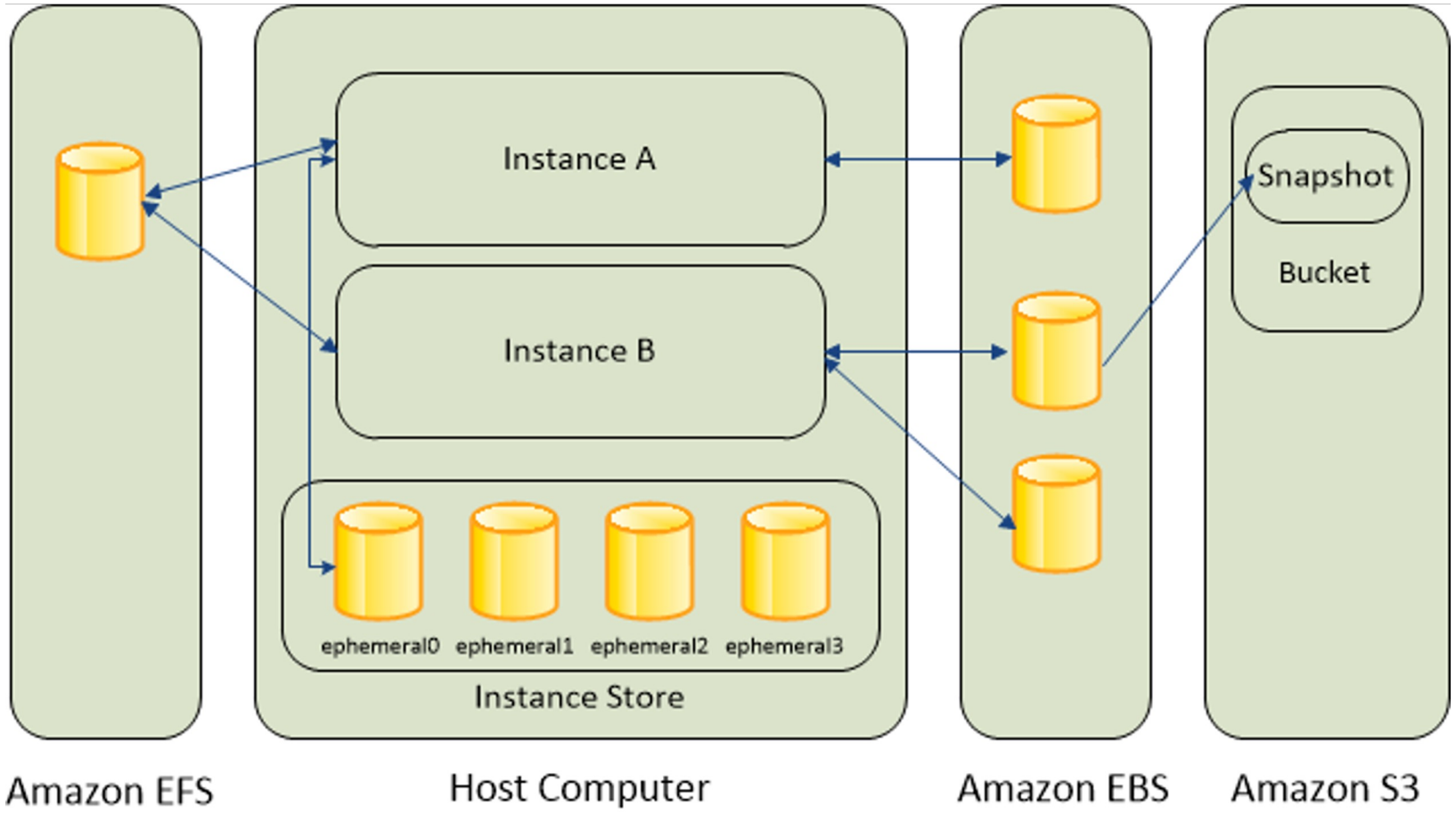
Encryption

- Instance store does not have out of box solution for encrypting data at rest.
- However there are ways to encrypt data using custom solution such as https://aws.amazon.com/blogs/security/how-to-protect-data-at-rest-with-amazon-ec2-instance-store-encryption/

Final Thoughts on Instance Store

- Instance store is included in the virtual server and cannot exist without the virtual server unlike EBS.
- Don’t use an instance store for data that must not be lost.
- Instance store helps avoid noisy neighbor problem as the EC2 servers do not have to go over network to access data.
- Instance store backed volume cannot be “stopped”. They can only be rebooted or terminated.

Amazon Elastic File System (Amazon EFS)



Amazon Elastic File System (Amazon EFS)

- Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with Amazon EC2 instances in the AWS Cloud.
- It supports Network File System versions 4 (NFSv4) and 4.1 (NFSv4.1), which makes it easy to migrate enterprise applications to AWS or build new ones.
- It is also highly available and highly durable because it stores data and metadata across multiple Availability Zones in a Region.

Accessing Amazon EFS

- Amazon EFS must be mounted to Amazon EC2 instance. Once mounted, Amazon EFS provides a standard file system interface and file system access semantics.
- Multiple Amazon EC2 instances can access an Amazon EFS at the same time allowing EFS to provide a common data source for workloads and applications running on more than one Amazon EC2 instance.
- Amazon EFS is designed to meet the needs of multi-threaded applications and applications that concurrently access data from multiple EC2 instances and that require substantial levels of aggregate throughput and input/output operations per second (IOPS).

Scalability and Elasticity

- Amazon EFS automatically scales your file system storage capacity up or down as you add or remove files without disrupting your applications, giving you just the storage you need, when you need it, and while eliminating time-consuming administration tasks associated with traditional storage management (such as planning, buying, provisioning, and monitoring).
- Your EFS file system can grow from an empty file system to multiple petabytes automatically, and there is no provisioning, allocating, or administration.

Final Thoughts on EFS

- EFS cannot be used as root volume for your EC2 instance.
- Data in EFS is replicated across multiple Availability Zones (AZs).

EFS Use Case

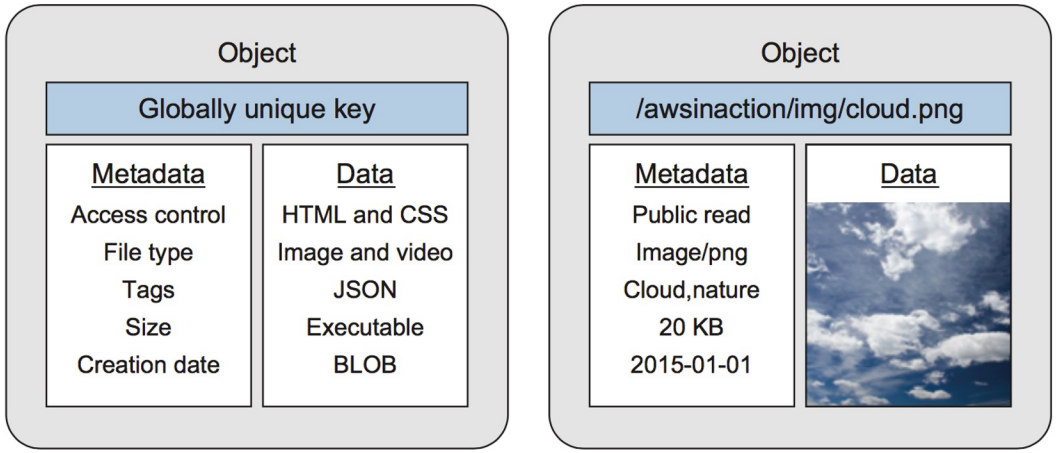
- Big Data and analytics
- Media processing workflows
- Content management
- Web serving
- Home directories

Object Storage Service

Amazon S3

What is Object & Object Store?

- In an object store, data is stored as objects.
- Each object consists of a globally unique identifier, some metadata, and the data itself.
- The separation of metadata and data allows clients to work only with the metadata for managing and querying data.
- You only have to load the data if you really need it.
- Metadata is also used to store access-control information and for other management tasks.



Amazon Simple Storage Service (S3)

- S3 is one of the oldest AWS service.
- S3 offers unlimited storage space.
- S3 is highly available and durable.
- An object in S3 can be as large as 5TB.
- S3 is accessed using RESTful APIs.

Amazon S3 Use Cases

Amazon S3 is an excellent choice for a large variety of use cases ranging from a simple storage repository for backup & recovery to primary storage for some of the most cutting-edge cloud-native applications in the market today and everything in between.

Data Consistency Model for S3

- All S3 GET, PUT, and LIST operations, as well as operations that change object tags, ACLs, or metadata, are **strongly consistent**.

S3 - Storage Tiers/Classes

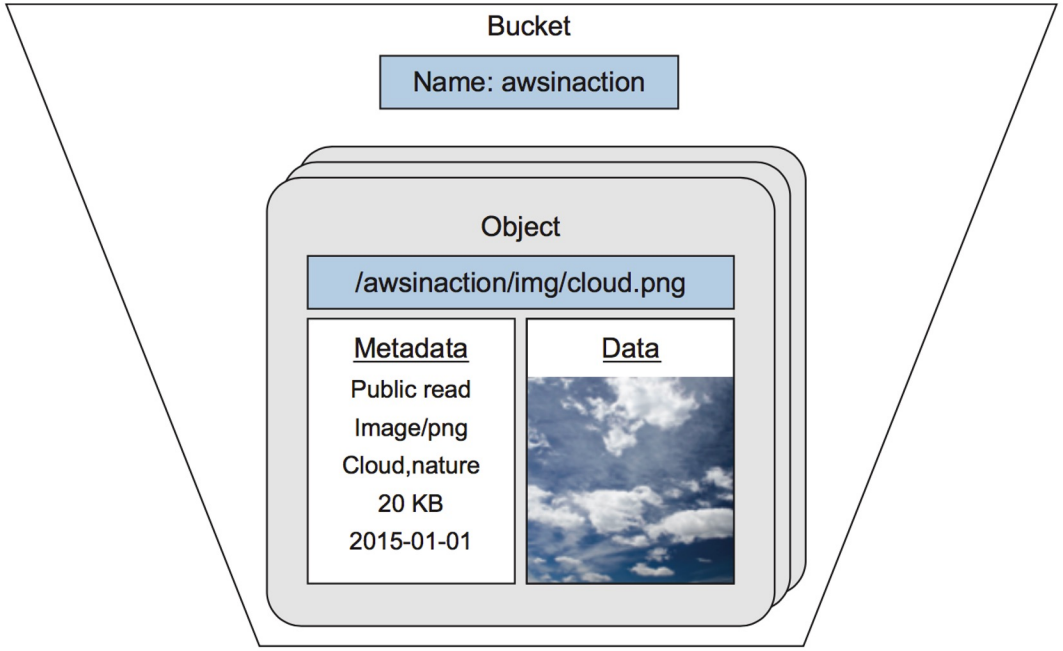
- **S3 (Standard)** - 99.99% availability, 99.999999999 durability, stored redundantly.
- **S3 - IA** (Infrequently Accessed) For data that is accessed less frequently but requires rapid access when needed. Costs less than S3 but you pay for retrieval.
- **Reduced Redundancy Stora**ge - 99.99% durability and 99.99% availability of objects over a given year.
- **Glacier** - Very cheap but used for archival only. It takes 3-5 hours to retrieve data from glacier.

S3 - Storage Tiers/Classes

	Standard	Standard - IA	Amazon Glacier
Designed for Durability	99.999999999%	99.999999999%	99.999999999%
Designed for Availability	99.99%	99.9%	N/A
Availability SLA	99.9%	99%	N/A
Minimum Object Size	N/A	128KB*	N/A
Minimum Storage Duration	N/A	30 days	90 days
Retrieval Fee	N/A	per GB retrieved	per GB retrieved**
First Byte Latency	milliseconds	milliseconds	select minutes or hours***
Storage Class	object level	object level	object level
Lifecycle Transitions	yes	yes	yes

S3 Buckets

- Every object you store in Amazon S3 resides in a bucket.
- Buckets may be used to group related objects in the same way that you use a directory to group files in a file system.
- Buckets have properties, such as access permissions and versioning status, and you can specify the region where you want them to reside.
- Buckets must have globally unique name.
- No other AWS customer in any other region can have same bucket name as yours.



Bucket Logging

- Logging provides a way to get detailed access logs delivered to a bucket you choose.
- An access log record contains details about the request, such as the request type, the resources specified in the request worked, and the time and date the request was processed.

Bucket Event Notifications

Amazon S3 buckets can be configured to send a notification message to a destination whenever one of the following event occur:

1. An object created event
2. An object removed event
3. A Reduced Redundancy Storage (RRS) object lost event

Bucket Versioning

- A versioning-enabled bucket can have multiple versions of objects in the bucket.
- By default, S3 versioning is disabled for every bucket.
- Suppose you use the following steps to upload two objects:
  - Add an object with key A and data 1.
  - Add an object with key A and data 2.
- If you download, also known as get, the object with key A, you'll download data 2. The old data 1 doesn't exist anymore.
- You can change this behavior by turning on versioning for a bucket.

Data Lifecycle Management

- Lifecycle configuration rules allow you to define actions you want Amazon S3 to take during an object's lifetime such as following:
  - Transition objects to another storage class
  - Archive objects
  - Delete objects after a specified period of time
- You can define a rule for all objects or a subset of objects in the bucket (by specifying the key name prefix). You can temporarily disable a rule.

Object Tagging

- S3 Object Tags are key-value pairs applied to S3 objects which can be created, updated or deleted at any time during the lifetime of the object.
- With these, you'll have the ability to create Identity and Access Management (IAM) policies, setup S3 Lifecycle policies, and customize storage metrics.
- These object-level tags can then manage transitions between storage classes and expire objects in the background.

Encryption

- SSE with Amazon S3 Key Management (SSE-S3)
  - With SSE-S3, Amazon S3 will encrypt your data at rest and manage the encryption keys for you.
- SSE with Customer-Provided Keys (SSE-C)
  - With SSE-C, Amazon S3 will encrypt your data at rest using the custom encryption keys that you provide.
- SSE with AWS KMS (SSE-KMS)
  - With SSE-KMS, Amazon S3 will encrypt your data at rest using keys that you manage in the AWS Key Management Service (KMS).

Cross-Region Replication

- Cross-region replication is the automatic, asynchronous copying of objects across buckets in different AWS regions.
- By activating cross-region replication, Amazon S3 will replicate newly created objects, object updates, and object deletions from a source bucket into a destination bucket in a different region.
- See http://docs.aws.amazon.com/AmazonS3/latest/dev/crr-what-is-isnot-replicated.html for limitations of Cross-Region Replication.

Querying S3

- Amazon S3 doesn’t offer query capabilities to retrieve specific objects.
- When you use Amazon S3 you need to know the exact bucket name and key for the files you want to retrieve from the service.
- Amazon S3 can’t be used as a database or search engine by itself.
- Instead, you can pair Amazon S3 with Amazon DynamoDB, Amazon CloudSearch, or Amazon Relational Database Service (Amazon RDS) to index and query metadata about Amazon S3 buckets and objects.

S3 is not suited for….

File System - Amazon S3 uses a flat namespace and isn’t meant to serve as a standalone, POSIX-compliant file system. Instead, consider using Amazon EFS as a file system.

Relational Databases

Amazon Relational Database Service (RDS)

What is a relational database?

- Relational databases are what most of us are all used to. They have been around since late 70s.
- The term "relational database" was invented by E. F. Codd at IBM in 1970.
- An RDBMS at minimum
  - Presents the data to the user as relations (a presentation in tabular form, i.e. as a collection of tables with each table consisting of a set of rows and columns);
  - Provides relational operators to manipulate the data in tabular form.

Popular RDBMS

- Oracle
- MySQL
- Microsoft SQL Server
- PostgreSQL
- IBM DB2
- Microsoft Access
- SQLite
- Amazon Aurora
- MariaDB

Transactions

- In order for a database management system (DBMS) to operate efficiently and accurately, it must support ACID transactions.
- *ACID* is short for *Atomicity*, *Consistency*, *Isolation*, *Durability*.

Atomicity

- Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, then the entire transaction fails, and the database state is left unchanged.
- An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes.
- To the outside world, a committed transaction appears (by its effects on the database) to be indivisible ("atomic"), and an aborted transaction does not happen.

Consistency

The consistency property ensures that any transaction will bring the database from one valid state to another.

Isolation

The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially, i.e., one after the other.

Durability

- The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.
- In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter).

Automated Backups of AWS RDS

There are two different types of backups for AWS.

1. Automated Backups
2. Database Snapshots

Advantages of AWS RDS

- Easy to Administer
- Highly Scalable
- Available and Durable
- Fast
- Secure

Automated Backups

- Recover database to any point in time within the retention period.
- Automated backups take a full daily snapshot and will also store transaction logs throughout the day.
- Automated Backups are enabled by default.
- Backup is stored in S3 but it is free.
- You do not pay for S3.
- During backup window, you might experience higher latency as storage I/O may be suspended.
- Automated backups are deleted when you terminate your RDS instance.

Database Snapshots

- Snapshots are done manually.
- They are stored even after you delete the original RDS instance.

Restoring Backups

Whenever you restore either an **Automatic Backup** or a manual **Database Snapshot** backup, the restored version of the database will be a new RDS instance with a new end point.

Encryption at Rest and in Transit

- Amazon RDS allows you to encrypt your databases using keys you manage through AWS Key Management Service (KMS).
- On a database instance running with Amazon RDS encryption, data stored at rest in the underlying storage is encrypted, as are its automated backups, read replicas, and snapshots.
- Amazon RDS supports the use of SSL to secure data in transit.

Amazon RDS Multi-AZ Deployments

- Amazon RDS Multi-AZ deployments provide enhanced availability and durability for Database (DB) Instances.
- When you provision a Multi-AZ DB Instance, Amazon RDS automatically creates a primary DB Instance and synchronously replicates the data to a standby instance in a different Availability Zone (AZ).
- Each AZ runs on its own physically distinct, independent infrastructure, and is engineered to be highly reliable.
- In case of an infrastructure failure Amazon RDS performs an automatic failover to the standby, so that you can resume database operations as soon as the failover is complete.
- Since the endpoint for your DB Instance remains the same after a failover, application can resume database operation without the need for manual administrative intervention.

Amazon RDS Read Replicas

- Amazon RDS Read Replicas provide enhanced performance and durability for database (DB) instances.
- This replication feature makes it easy to elastically scale out beyond the capacity constraints of a single DB Instance for read-heavy database workloads.
- You can create one or more replicas of a given source DB Instance and serve high-volume application read traffic from multiple copies of your data, thereby increasing aggregate read throughput.
- Read replicas can also be promoted when needed to become standalone DB instances.

NoSQL Databases

Value of Relation Database

- Store large amounts of persistent data
- Concurrency
- Standard Model (Relational Algebra & SQL)
- ACID

Not so Good side of RDBMS

- Impedance mismatch - The difference between relational model and in-memory data structure.
- Mediocre horizontal scaling support

What is NoSQL?

- The term NoSQL may refer to "non SQL", "non relational" or "not only SQL".
- A NoSQL database provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases.

Why is it called "NoSQL"?

- Johan Oskarsson, then a developer at Last.fm, wanted a name for meetup in 2009.
- Something that would make a good hashtag: short, memorable, and without too many Google hits so that search of the term would quickly find the meetup.
- The name attempted to label the emergence of an increasing number of non-relational, distributed data stores.

Characteristics of NoSQL Database

- Schema-less
- Does not use relational model
- Runs well on cluster

NoSQL is more than rows

NoSQL systems store and retrieve data from many formats such as

- Key-Value (DynamoDB)
- Graph
- Column-family (Cassandra)
- Document (MongoDB)

NoSQL is free of Joins

NoSQL systems allow you to extract your data using simple interfaces. It does not support “joins”.

NoSQL databases are Schema Free

- A NoSQL system can handle data in various formats.
- You do not have to tell NoSQL system in advance about the format of data.

Scaling NoSQL

- NoSQL systems scale linearly.
- More processors you add, more performant the system gets.

CAP Theorem

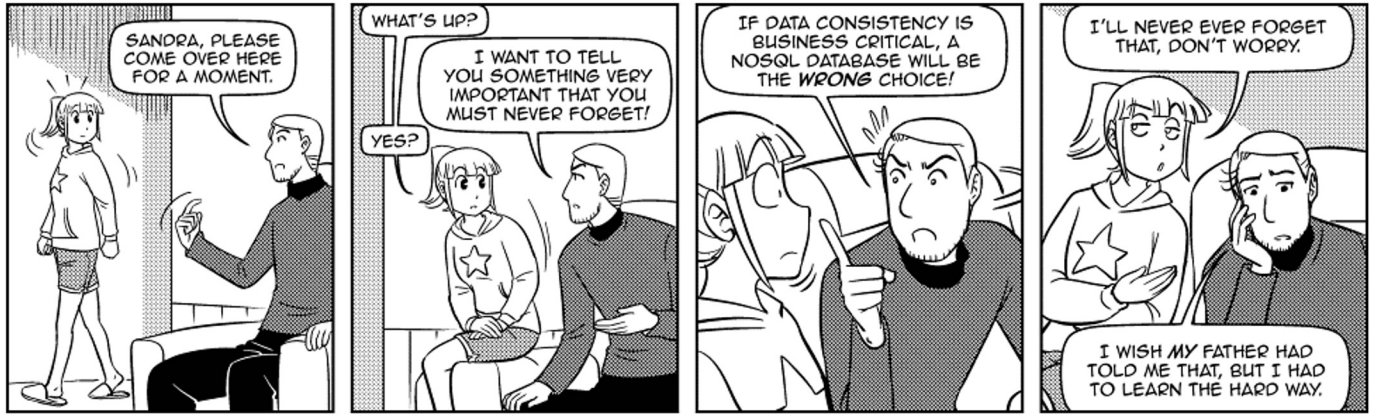
- The CAP theorem states that it is impossible for a distributed computer system to simultaneously provide more than two out of three of the following guarantees:
  - Consistency - Every read receives the most recent write or an error
  - Availability - Every request receives a (non-error) response – without guarantee that it contains the most recent write
  - Partition tolerance - The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
- In other words, the CAP Theorem states that in the presence of a network partition, one has to choose between consistency and availability.
- Note that consistency as defined in the CAP Theorem is quite different from the consistency guaranteed in ACID database transactions.

Transactions in NoSQL

Most NoSQL stores lack true ACID transactions.

NoSQL Databases & Consistency

NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes "eventually" (typically within milliseconds) so queries for data might not return updated data immediately or might result in reading data that is not accurate, a problem known as stale reads.

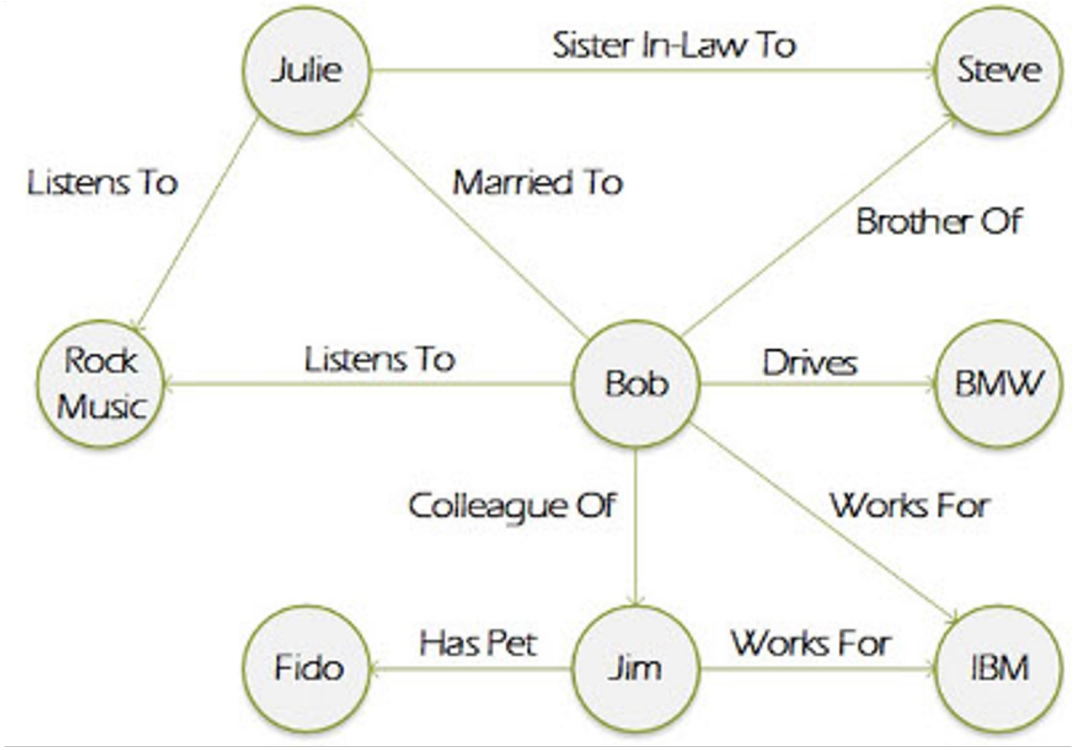


What is key-value store?

- A key-value store is a simple database that when presented with a simple string (the key) returns an arbitrary large BLOB of data (the value).
- Key-value stores usually have no query language.
- In a key-value store, key is of String data type and value can be of any type.
- One does not have to specify the type for value.
- Example: DynamoDB & Redis

Graph Databases

- This kind of database is designed for data whose relations are well represented as a graph consisting of elements interconnected with a finite number of relations between them.
- The type of data could be social relations, public transport links, road maps or network topologies.



Column Family (Bigtable) Stores

- Bigtable maps two arbitrary string values (row key and column key) and timestamp (hence three-dimensional mapping) into an associated arbitrary byte array.
- Think of spreadsheets.
- You can use combination of row number and column letter as an address to “look up” the value of any cell.
- Examples: Google Bigtable & Apache Cassandra

Document Database

- Document Databases differ from other NoSQL databases we have looked at in the sense that they Key may be simple id which is never used or seen but you can get almost any item out of document store by querying any value or content within the document.
- The central concept of a document-oriented database is the notion of a document.
- While each document-oriented database implementation differs on the details of this definition, in general, they all assume documents encapsulate and encode data (or information) in some standard formats or encodings such as JSON, BSON, XML, etc.

Normalization & Denormalization

- **Database normalization** is a technique for designing relational database schemas that ensures that the data is optimal for ad-hoc querying and that modifications such as deletion or insertion of data does not lead to data inconsistency.
- **Database denormalization** is the process of optimizing your database for reads by creating redundant data. A consequence of denormalization is that insertions or deletions could cause data inconsistency if not uniformly applied to all redundant copies of the data within the database.

Polyglot Persistence

- Polyglot Persistence, like polyglot programming, is all about choosing the right persistence option for the task at hand.
- NoSQL has result in widespread use of Polyglot Persistence.

Concurrency Control Mechanisms

- **Optimistic** - Delay the checking of whether a transaction meets the isolation and other integrity rules (e.g., serializability and recoverability) until its end, without blocking any of its (read, write) operations ("...and be optimistic about the rules being met..."), and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit. An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead (versus executing it to the end only once). If not too many transactions are aborted, then being optimistic is usually a good strategy.
- **Pessimistic** - Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is typically involved with performance reduction.

Write-Write Conflict

- Two users updating same data item with different values.
- Optimistic Approaches:
  - Conditional Update: See if value has changed since last read before update
  - Process both updates and record that there is a conflict

Read-Write Conflict

- Alice and Bob are using a website to book tickets for a specific show. Only one ticket is left for the specific show. Alice signs on first to see that only one ticket is left, and finds it expensive. Alice takes time to decide. Bob signs on and also finds one ticket left, and orders it instantly. Bob purchases and logs off. Alice decides to buy a ticket, to find there are no tickets. This is a typical read-write conflict situation.

Version Stamps

- A field that changes every time the underlying data in the record changes.
- When you read the data you keep note of the version stamp, so that when you write data you can check to see if the version has changed.

Single Server (Distribution Model)

- One machine handles all reads and writes to the database.
- Simplest distribution model as there is no cluster.
- Suitable for development setup.

Sharding (Distribution Model)

- Store different parts of data on different servers.
- Spread the load across the cluster as different users ideally will be accessing data from different nodes.
- Sharding can improve both read and write performance.
- Note: If the data is sharded using wrong key, you can end up with highly unbalanced cluster and all traffic will end up going to same server nodes.

Primary/Secondary Nodes Replication (Distribution Model)

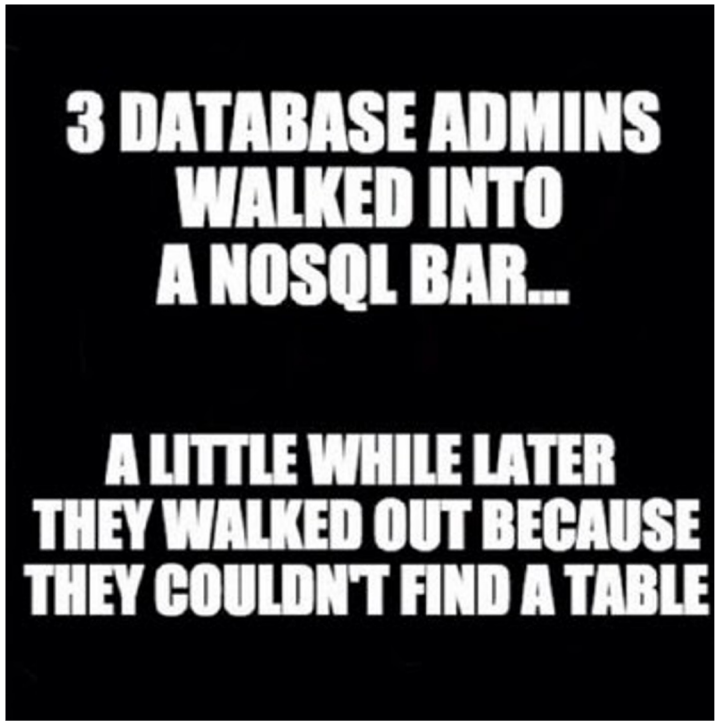
- In this distribution model you replicate data across multiple nodes.
- One node is designated as “primary” and others are designated as “secondary.
- This “primary” node is authoritative source of data and is usually responsible for processing any updates to that data.
- primary node can be a bottleneck as all writes must go to it.
- If primary node dies, secondary node can still answer read queries while another node is elected primary either automatically or manually.
- primary node can be single point of failure.

Peer-to-Peer Replication (Distribution Model)

- There is no master node in peer-to-peer replication.
- All replicas have equal weight, they can all accept writes, and loss of any of them doesn’t prevent access to data store.

Combining Sharding & Replication (Distribution Model)

Peer-to-peer replication combined with sharding is common strategy used by column-family databases.


3 DATABASE ADMINS
WALKED INTO
A NOSQL BAR...
A LITTLE WHILE LATER
THEY WALKED OUT BECAUSE
THEY COULDN'T FIND A TABLE

Additional Resources

See Lecture Page