Storage on the Cloud
Types of Cloud Storage
Object storage

● Store arbitrary data objects

● Many general uses:
  ○ Media / user-generated content
  ○ Log files
  ○ Intermediate pipeline outputs

● Each storage container is called a bucket
Properties of Object Storage

- Unlike a filesystem, objects aren’t stored hierarchically (no directories)
  - Instead: lookup an object using bucket name and key

- Designed for high scale and high reliability
  - No limits on the size of the objects that can be stored
  - AWS S3 features 99.999999999% (11 9s) data durability due to redundant architecture

- Harder to do better than S3 for general data storage!
Demo: S3 bucket
Data Archival

- Cheaply retain infrequently accessed data for a long time
  - Why? Backups, regulatory/compliance requirements, support some niche customer needs

- Tradeoff between cost, data access latency, and availability
  - S3 has multiple intermediary archive storage tiers

- Example solutions:
  - AWS S3 Glacier
  - Archive tier for Google Cloud Storage
  - Azure Archive Storage
Axes of Object Storage Costs

● Data volume: $0.001-$0.025/GB/mo, depending on access frequency
  ○ To store 1TB of data: $1-$25/mo

● Requests:
  ○ Writes: $0.005-$0.05 per 1000 requests (more for archive tiers)
  ○ Reads: $0.0004-$0.01 per 1000 requests (more for archive tiers)

● Data egress to the internet: $0.05-$0.09/GB
  ○ $90 to serve 1TB of content
## S3 Cost Table

<table>
<thead>
<tr>
<th>Tier</th>
<th>Cost per GB</th>
<th>Cost per 1000 Reads</th>
<th>Cost per 1000 Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3 Standard</td>
<td>$0.023</td>
<td>$0.0004</td>
<td>$0.005</td>
</tr>
<tr>
<td>S3 Standard (Infrequent)</td>
<td>$0.0125</td>
<td>$0.001</td>
<td>$0.01</td>
</tr>
<tr>
<td>S3 Glacier (Instant)</td>
<td>$0.004</td>
<td>$0.01</td>
<td>$0.02</td>
</tr>
<tr>
<td>S3 Glacier (Flexible)</td>
<td>$0.0036</td>
<td>$0.0004</td>
<td>$0.03</td>
</tr>
<tr>
<td>S3 Glacier (Deep Archive)</td>
<td>$0.00099</td>
<td>$0.0004</td>
<td>$0.05</td>
</tr>
</tbody>
</table>
Reducing Storage Costs

- **Easy optimization: S3 Intelligent Tiering**
  - Automatically chooses the lowest-cost storage tier dynamically based on stored data volume and request frequency

- **Reducing data egress fees requires more care**
  - Consider how much data actually needs to be served to the internet (data transfer within AWS – especially within a region – is cheaper)
  - For content that needs to be served: cache it with a CDN (e.g. Cloudfront)
Block and file storage

- Minimal abstraction: a storage device or a file system
- Used to back compute instances and for file shares
- Often used as a migratory step from on-premise infra
Databases

● Quick insertion / retrieval of specific pieces of structured data using queries
  ○ Allows easy access to only the data momentarily relevant for an application
  ○ e.g. retrieve the most recent comments for a given user

● Many types of databases are available
  ○ Traditional SQL databases, e.g. MySQL, Postgres via AWS RDS, Azure SQL DB, GCP SQL
  ○ NoSQL databases
    ■ Key-value store: AWS DynamoDB, Google BigTable, Azure Cosmos
    ■ Graph database: AWS Neptune

● More detail in Wednesday’s guest lecture (Benjamin Bercovitz)
Relational Databases

- Data for an application is stored in a database running on a database server/cluster in tables with defined schemas
  - Relationships between tables through foreign keys: unique keys that map to unique keys of other tables

- Constraints on structure and data types help with efficient access and data integrity
  - Caution: also makes changing the data model hard! Consider data model carefully when starting to build
Relational Databases

Query via SQL
e.g. “find the first and last name of customers who bought product 1”

```
SELECT firstName, lastName
FROM customers
JOIN orders
ON customers.customerID = orders.customerID
WHERE productID = 1;
```
Object-Relational Mappers

- Interface with relational databases at application language level of abstraction
  - e.g. SQLAlchemy (Python), Active Record (Ruby on Rails), Sequelize (Typescript)

```python
Session.query(Customers, Orders)
  .filter(Customers.CustomerID == Orders.CustomerID)
  .filter(Orders.ProductID == 1)
```

- Advantages: composability, maintainability, more defense against SQL injection attacks
Demo: PostgreSQL database
Key-Value Stores

- Single primary key (like a hash table), but value schema isn’t enforced
  - Making sure value schema stays consistent can be challenging

- Offers very fast lookup of values from keys, but hard to do aggregate operations across rows

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paul</td>
<td>(091) 9786453778</td>
<td>10.94.214.172</td>
<td>3c:22:fb:86:c1:b1</td>
</tr>
<tr>
<td>Greg</td>
<td>(091) 9686154559</td>
<td>10.94.214.173</td>
<td>00:0a:95:9d:68:16</td>
</tr>
<tr>
<td>Marco</td>
<td>(091) 9868564334</td>
<td>10.94.214.174</td>
<td>3c:1b:fb:45:c4:b1</td>
</tr>
</tbody>
</table>
Caches & In-Memory Databases

- For data that doesn’t need to persist indefinitely, but needs to be quickly accessed
  - Store frequently accessed or computed data in memory: reduces application latency
  - Common use cases: session management, user feeds (if infrequently changed)

- Typically uses key-value architecture

- Common open-source software: Memcached and Redis
  - Offered as managed services by cloud providers, e.g. AWS ElastiCache
Document-Model Databases

- Store documents (JSON objects) in collections at a specified path
  - Schema not enforced and can change easily over time – blessing and curse

- Well-known options: MongoDB, AWS DocumentDB, Google Firestore
Hosted Databases as “Backend-as-a-Service”

- Motivation: Avoid writing a full backend by exposing database functionality as public APIs
  - Avoids having to write new REST API endpoints for each bit of functionality
  - Makes creating some sorts of apps easier: just focus on the frontend logic
Pitfalls of BaaS

- Any client (incl. malicious ones) can access database
  - Database must gate access to sensitive data
  - Yet, less granularity in access control vs. writing regular backend code
  - Poor validation (e.g. misconfigured rules) $\rightarrow$ unauthorized data access

- Application logic becomes irrevocably tied to specific BaaS platform
  - Cost and performance at the whim of the BaaS provider; migration becomes difficult

Our recommendation: use hosted technologies with well-defined open protocols e.g. AWS Aurora can be swapped out to any other hosted Postgres offering
Storage in the Real World
Common Web Service Architecture

Instagram’s initial architecture (2011) – Assignment 2 (Yoctogram) is a modernized version
Data Warehouse

- Aggregate data from many diverse sources into a central database
  - Eager processing: Extract-Transform-Load (ETL) integrates incoming data into a consistent format
- Enables efficient analytics queries for business and product insights
Data Lake

- Aggregate data from many diverse sources, but without a consistent schema at storage time
  - Storage using S3, Hadoop Filesystem (HDFS) – self-managed – or dedicated providers: Azure Data Lake, Google BigLake
- Allows flexibility to figure out what business questions need to be asked of the data later
Criticisms of Data Lakes

- Lack of structure makes it easy to lose track of what data is stored
  - “Data Swamp”, “Data Graveyard”

- Also creates inefficiencies in querying for relevant data
A hybrid approach: Data Lakehouse
Assignment 2 Out Soon (Due 2/13)
AWS Credits to be Distributed on Friday
Next Lecture: Database Design and Tradeoffs (1/24)
GUEST LECTURE by Benjamin Bercovitz (Verkada)
**Mandatory** (graded) attendance