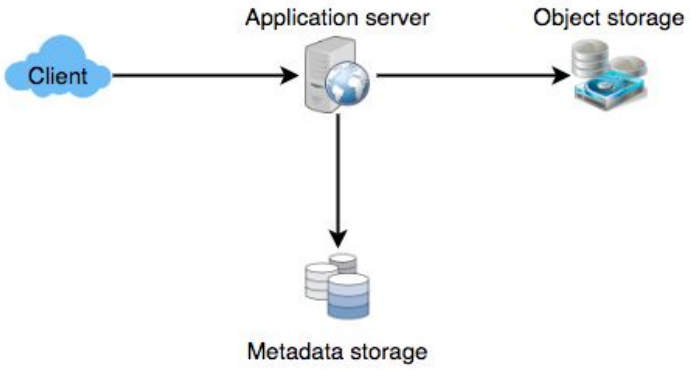
I. designing pastebin
    1. pastebin
        a. When data (text, image) is saved in the service, return as url-access to text stored in the url
        b. Share data easily and quickly using url
    2. requirements and goals of the system
        a. functional requirements
            i. Saving text returns url (reduced functionality)
            ii. Automatic expiration, expiration time selection
            iii. custom url support
        b. non-functional requirements
            i. High reliability-uploaded data cannot be lost
            ii. High availability- The text must be returned when requesting url
            iii. real-time access with minimum latency
            iv. The system-generated url should not be guessable
        c. extended requirements
            i. analystic
    3. design considerations
        a. Is it possible to limit the amount of text? Yes, 10mb is suitable
        b. Should I limit the number of characters in the custom url?
            i. Freely available
            ii. It is recommended that the number of characters is the same as the number of system-generated urls-maintain DB consistency
    4. capacity estimation and constraints

[assume]			
request per day	1,000,000	write/day	
read/write ratio	5	1	
1 paste size	10,000	byte	
[estimation]			
QPS	58	read/sec	
	12	write/sec	
storage	10,000,000,000	write/day	
	3,650,000,000,000	write/year	
bandwidth	115,741	byte/sec	incoming data
	578,704	byte/sec	outgoing data

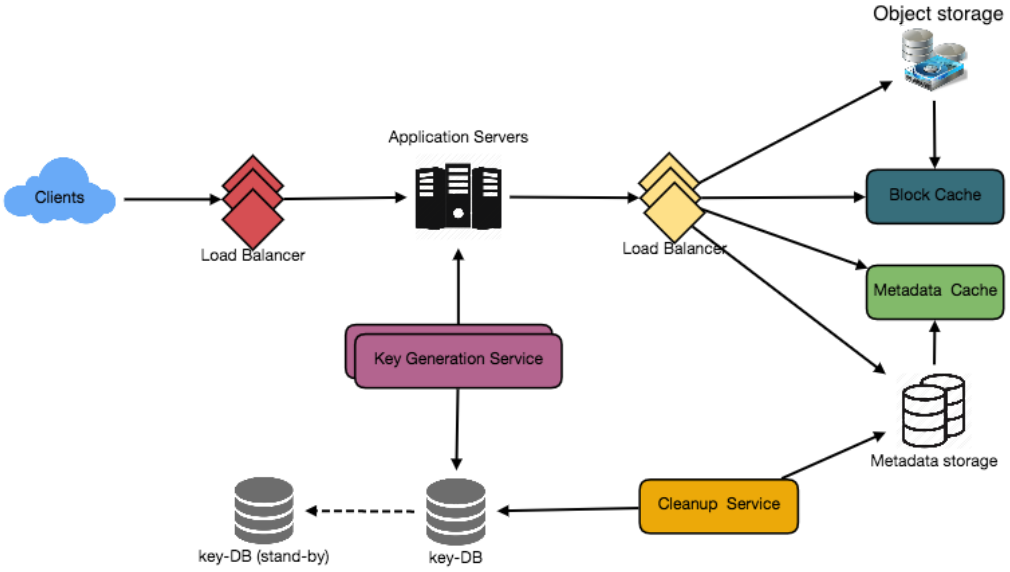
5. system apis - soap, rest api
    a. addPaste (api_dev_key, paste_data, custom_url=none, user_name=none, expire_date=none)
        i. return (string) : paste_url / error code
    b. getPaste (api_dev_key, paste_url)
        i. return (string) : paste_date / error code
    c. deletePaste (api_dev_key, paste_url)
        i. return (string) : success / error code
6. database design
    a. considerations
        i. Very many records
        ii. Small metadata size (less than 1kb)
        iii. contents (text) size average 10kb, maximum 10mb
        iv. High usage
    b. type
        i. Metadata Storage: Vertical expansion (high relationship, small capacity), RDBMS (ex) MySQL, MS-SQL, Oracle
        ii. Contents Storage: Horizontal expansion (relational, large capacity), Object Storage (ex) Amazon S3
    c. table
        i. Paste
            - URL: varchar(6), PK
            - ContentsPath: varchar(512)
            - UrlAlias : varchar(6)
            - ExpireDate : datetime,
            - UserID : int, FK
        ii. User
            - UserID : int, PK
            - Name : varchar(20)
            - Email : varchar(32)
            - DateOfBirth : datetime
            - LastLogin : datetime

7. high level design



a. application layer
    i. Handling read / write requests
    ii. Communicate with backend storage
b. storage layer
    i. metadata storage
    ii. contents storage

8. component design



c. application layer
    i. Receive write request-> Generate 6-digit key-> Save content and key to DB-> Return key or error code

- When a new key overlaps with an existing key: Regenerate the key and try until it succeeds-KGS can solve the shortcomings
- KGS (Key Generation Service)
  - ✓ Keys are generated in advance and put in a DB table-used whenever necessary
  - ✓ Shorter creation time, no overlap / conflict
- Return error code when CustomUrl is already in DB
- Can each application server cache part of Key_DB?
  - ✓ Possible, but lost unused key cached when the server is shut down

ii. Request to read-> Search key-> Return Uploaded Text or Error Code

d. datastore layer

i. metadata storage: vertical expansion (high relationship, small capacity), RDBMS Dynamo, Cassandra

ii. contents storage: horizontal expansion (relational, large capacity), Object Stroage (Amazon S3)

8. purge and DB cleanup

a. Delete expired content

i. (Real-time cleanup): Increase DB usage-Slow system speed

ii. (Delayed theorem)

- Customer request-> In case of expired record, return error and delete
- Separate cleanup service execution: Regular, service should be light and run when there is little traffic

iii. Default expiry time setting, expired URL keys can be reused

iv. Any URLs that haven't been used in 6 months?

- case by case
- Trends in making maintenance decisions: lower storage costs

9. data partitioning and replication
    a. data partitioning
        i. range based partitioning
            - Determine the partition to store using the first letter of the URL
            - Partitions become unbalanced (ex) Many start with E
        ii. hash based partitioning
            - hash function : key % number of server
            - Problem: Can't expand, partition becomes unbalanced
            - Solved: Consistent Hashing, repartitioning
    b. replication
        i. Traffic distribution: system response speed improvement
        ii. Data backup: If the original is lost, it can be restored as a copy
10. cache and load balancer
    a. cache
        i. Commercial solutions such as Memcache available
        ii. cache size : 20 % of total (based on 20:80 rule)
        iii. cache evitction policy : LRU (Least Recently Used)
        iv. (Efficiency) increases when the load is distributed among servers by replicating the cache server.
        v. cache type
            - write through cache
                ✓ Save both hdd and cache
                ✓ Pros: consistent sync
                ✓ Cons: write slow
            - write around cache

            - ✓ Save only hdd, copy from hdd to cache only when cache miss (the desired data is not in cache) and service
            - ✓ Pros: write medium
            - ✓ Cons: slow read only when cache miss occurs
        - write back cache
            - ✓ Cache only, regularly update cache update to hdd
            - ✓ Advantage: Fast write
            - ✓ Disadvantages: Data corruption when server is down (data in cache does not sync with hdd and evaporates)
    b. load balancer location
        i. Between client and web server
        ii. Between web server and application server
        iii. Between application server and db or cache
11. security and permissions
    a. Can a user create a personal URL or allow a specific set of users to access the URL?
        i. Set record (tuple) as public / private-> In the case of private, create a separate table and store the accessible UserID in the record-> Return error code if there is no user authority (ex) HTTP 401 Error