**Design for Short URL service**

**Problem Statement**

The requirement is to create a URL shortener service that converts a provided long URL - <https://some.domain.com/some-path-to/somewhere> and return a shorter URL like <https://w.xyz/abcdef> for easier accessibility.

**Requirements**

* Aggregated metrics on the created shortened URL.
* Users can create custom URL with maximum character limit of 16, if not provided a custom length. The URL must be as small as possible.
* 100 million requests created per month.
* Once created, the URL stays in the system for a period of 2 years. If there are no hits in the last 2 years. The URL is removed.
* Creating a short URL for a long URL is expected to be O(1) at the user experience API.
* The read: write ratio of the URLs is 100:1.
* Application should support zonal deployments, along with alerting system for the developers.

**Estimation**

We have 100M new URL shortenings per month, with 100:1 read/write ratio, we can expect 10B redirections during the same period:

New URLs shortenings per second:

100 million / (30 days \* 24 hours \* 3600 seconds) = ~38 new URLs/s

URLs redirections per second will be:

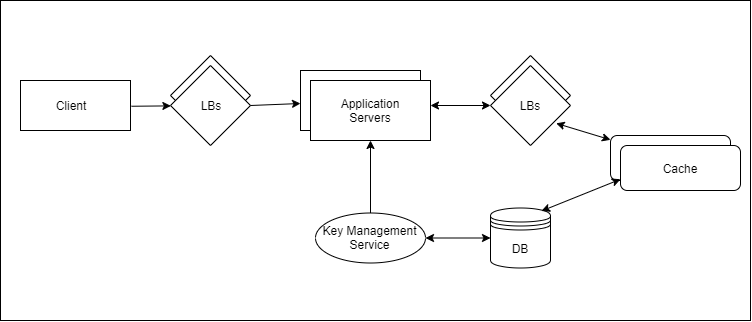
100 \* 38 URLs/s = 3800 redirections/s

Storage:

In our MVP it was around 200Bytes with ID, short URL with 6 key length, a long URL with 105 char length and created date with only date.

Since we store every new URL for 2 years with 100M URLs every month. We expect 2.4Billion records. Assuming 500Bytes for each record in DB, we can expect to store 1.2 TB.

**Design**



**Key Management**

We will have a separate service (Key Management Service) to create the keys and store in the DB. We will reuse this service to purge the short URLs which are older than 2 years.

Since this service is going to be running in background and creating unique keys, we can use several techniques to create the keys to independently create the keys.

We can create a new guid every time and pick the first set number of characters as key for a short URL generation. Once the key is used, the service will update the table accordingly.

With GUID method used in the MVP, if all the keys length are 6 characters, then we will only be able create ~2.1B unique keys which is not enough for our requirements.

For a scalable solution we will use Base62 encoding which allows us to use characters and numbers totalling 62. So for a 6 character unique key, we can serve 62^6 = ~56B unique keys.

**APIs**

Create short URL

POST https://<host>/Shorten

Accept : application/json

{

“originalUrl” : ”<your long url>”,

“keyLength” : 10

}

Get list of short URLs for a user

GET https://<host>/shortUrls/user/<UserID>

Response

List of short URLs.

Get Metrics for short URL

GET https://<host>/metrics/shortUrl/<shortUrlID>

Get Metrics for a particular user

GET https://<host>/metrics/user/<userID>

**Database**

We will use the RDBMS like PorstgreSQL, MsSQL etc., It can be scaled using sharding. With sharding we need to select a key for the shards. For a given short URL key we can create an integer hash and assign it as shard key for the db shards to evenly distribute the records.

User Table

* User ID: A unique user id or API key to make user globally distinguishable
* Name: The name of the user
* Email: The email id of the user
* Creation Date: The date on which the user was registered

ShortUrl Table

* Short URL ID: ID to used to track metrics for the URL.
* Short Url: Unique short URL.
* Original Url: The original long URL.
* UserId: The unique user id or API key of the user who created the short URL.
* Created Date: short URL created date.

Unique Short Key

* Short Key ID: primary key for the short key table.
* Short Key: a unique short key to be used for a short URL.
* IsKeyUsed: Indicates if the key is already in use for a short URL.

**Metrics**

For collecting and presenting various metrics to the users for this service while keeping the load our database to the minimum, we will make use of Kafka clusters.

The application servers will act as producers and put several metrics such as short URL, long URL, Ip address of the accessor, browser language, user agent etc., to the kafka queue

We can have our consumer service which will parse those messages from the subscribed topic into meaningful data for presentation.