CS 443 - Cloud Computing & Mobile Applications

Design Report

Group No: 6

Members:

Umer Shamaan - 21701563

Taha Khurram - 21701824

Sayed Abdullah Qutb - 21701024

Table of Contents

1.Overview

In this project, we are tasked to create a URL shortener application, in which a rather long URL is given as input and the URL is shortened and given back as an output. The application will be both web-based and mobile-based. We aim to implement it on the web, as a stand-alone website and as a mobile application on android devices. Examples of such URL shorteners are bit.ly, goo.gl and TinyURL [1].

The application will be programmed in such a way that when the user enters a URL, the shortened version of it will be generated and stored in the database. In addition, the user can create customized URLs as well. The aim is that the user can share the shortened URL, so that this URL can redirect back to the original long URL. Another importance of such applications is that when there is limitation in the number of characters that can be used in an input box, the shortened version of a URL can help in this case.

There will be authentication for the users of the application, so that the user can login to the system and access the URLs which were saved initially. Each of the shortened URLs will have a lifetime associated with them, so that the database is not overrun by old and unused URLs. The URLs will also have a frequency associated with them showing how often this URL was opened.

The application will consist of the hosting service (Heroku), the cloud service (Kubernetes), cache, backend(Cassandra). The number of requests is handled by a Load Balancer which distributes the load over different servers. There will be an admin GUI for the administrator of the system as well, in which the admin can see the analytics of the system. The following statistics are taken into account to while measuring the requirements of the system:

1. 10 URL requests per second
2. Around 2.5M requests per month

2. System Design

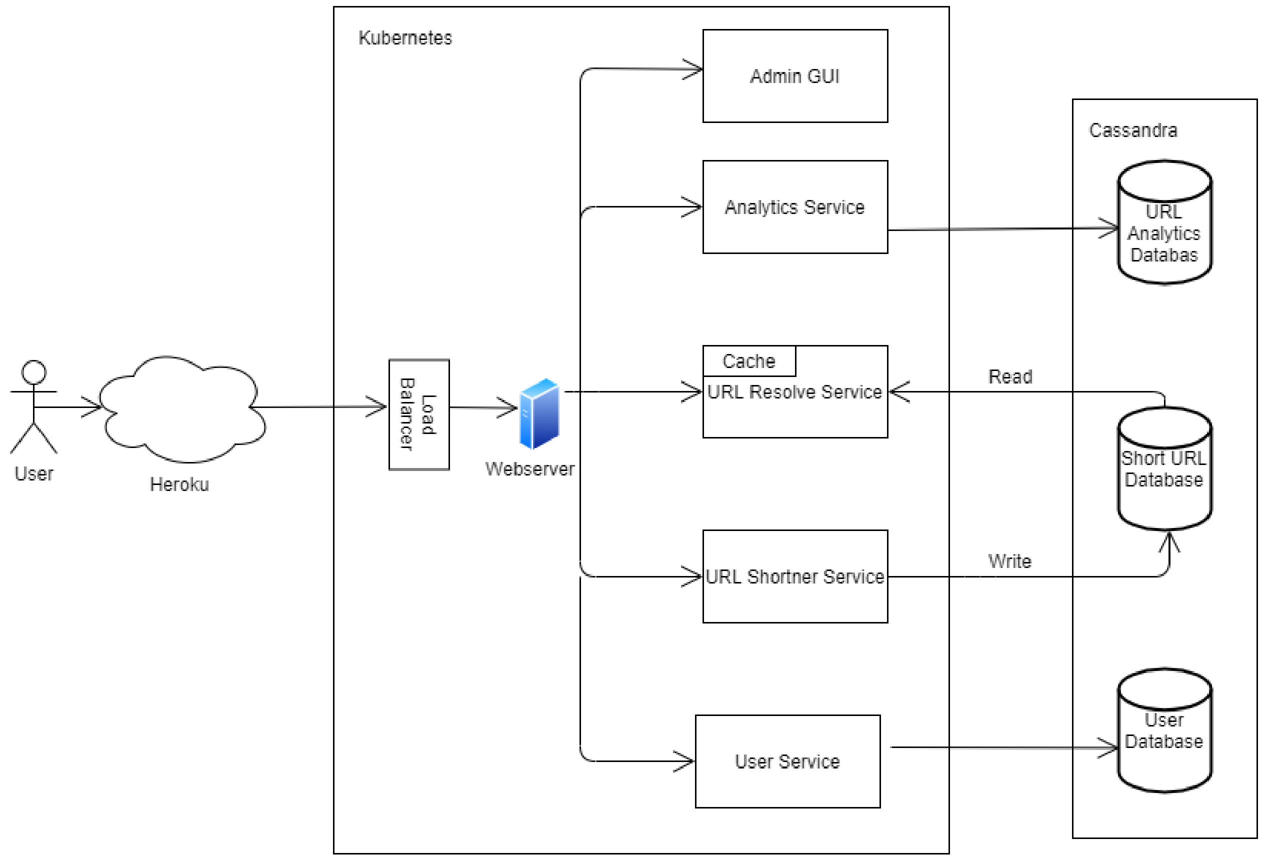

Figure 1 - System Design

<u>Components</u>:

User Service: Provides an API for users to login, register and manage users.

URL Shortener Service: API to generate a short url from a given long url.

URL Resolve Service: Provides an API to regenerate the long url from a given short url.

Analytics Service: Provides API to retrieve relevant analytical data.

Admin GUI: Provides a service for the admins to display relevant information regarding users' data, short urls etc.

3. Service Level Agreements (SLAs)

− The system should be highly available.
The customers will be able to use the URL Shortner and URL resolve services 99.0% of the time or they can request a refund for the unavailability of basic services.

− Redirection should happen with minimal latency.
The short URL should redirect to the original URL within ~400 ms (average) for all regions, upon failure the users will be viable to request for compensation.

− Short links should not be easy to predict.
The URL shortener service will create unique shortened URLs which will be 100% unique, if a predictable URL is created the user can immediately request for a unique shortened URL.

− API should be designed to be used by other parties.
The API will be usable for other parties 99% of the time, if the API is not available/usable the party may request a refund.

Exclusions:
In the case of a cyber attack such as DoS, we will not be held responsible for the downtime of our services and/or increased latency.

4. MicroServices

4.1. User Service

The user microservice will be in charge of handling the users who are already registered in the system or are registering later in the system. All the data of the users are saved in the User Database and is administered by the Admin. There is a sign-in page and can be accessed from the homepage. The user can sign in to the system through this page, by authenticating the credentials, or create a new account through the sign-up page of the application. Data for the user such as name, email, and password are saved in the database. The password is hashed to increase the security of the system.

4.2. URL Resolve Service

This microservice is in charge of redirecting the given URL. When the user clicks a shortened URL, this service takes the request and resolves it by redirecting the user back to the original URL. The shortened URL is being decrypted and then the Short URL Database is looked up to find the corresponding long URL for this short URL.

4.3. URL Shortener Service

This is the service for creating shortened URLs. After the user logs in to the system, there is an input box which the user can paste the original URL, the original URL is taken and a unique shortened version of the URL is generated by the system. This new shortened URL and its original long URL are both written to the Short URL Database.

4.4. Analytics Service

This service helps the users to look for details of a specific URL. This service can be accessed after the user is logged in to the system. Details of a URL such as, shortened URL, long URL and the given title of the URL is shown to the user. This data is stored in the URL Analytics Database.

4.5. Admin GUI

Admin of the system can look at the users registered in the system, the analytics of the URLs, or the shortened URLs itself. Admin can see details of the load on the system and other related statistics.

5. Application Programming Interfaces (APIs)

GET/userDatabase/users/{userID}
The userIDs will be unique for each user and will be used to retrieve the user data using the following API request. The API will return the user details or a message corresponding to no such user exists in the database.

UPDATE/userDatabase/users/{userID}/data
The “data” which contains: email, password, first name and last name will be updated for the userID given.

POST/userDatabase/users/{email}/{password}
This request sends the user’s email and password to be confirmed by the system, if the email or password are wrong/do not exist it will return a message for invalid credentials entered.

POST/userDatabase/users/create
The body of the request will contain the user’s First Name, Last Name, Email, and Password. If the email already does not exist in the database a new user will be created corresponding to the details provided otherwise an error message will be returned.

POST/ShortURLDatabase/shortURL/{userID}/{longURL}
This API request will take the userID and create a corresponding short URL for the longURL provided to the user

UPDATE/shortURLDatabase/shortURL/{shortURL}/{newURL}
The user will provide the old URL (shortURL) and change it to the new URL given.

GET/ShortURLDatabase/shortURL/{shortURL}
This API request will return the longURL for a given shortURL.

GET/URLAnalyticsDatabase/Analytics/shortURL/{shortURL}
The request will return the analytics for a shortURL provided, if the shortURL does not exist an error message will be given.

GET/userDatabase/users
This API request returns all the users that are registered.

GET/URLAnalyticsDatabase/Analytics/users/{userID}
This API request will return the analytics for a certain user.

GET/URLAnalyticsDatabase/Analytics/users
This API request will return the analytics for all users.

6. Tech Stack

1) **Platform:** We will use Heroku as our cloud platform. Heroku provides containers inside a ‘fully managed’ runtime environment which provide features such as configuration, orchestration, load balancing, failover, loggins and security, thus allowing the developers to focus more on code rather than infrastructure. Heroku allows developers to scale their applications both horizontally and vertically, pre-installed addons, application metrics and extensions, allowing the users to focus on innovation, and not operations [2]. Overall, it is an excellent platform for users introducing themselves to Cloud Computing. However, there are a few trade offs to using Heroku. First of all, users don't have full control over the resources. Moreover, it provides a relatively higher pricing for using the resources as compared to other platforms such as AWS and there is no centralized billing and management.

2) **Data (Persistency):** Apache Cassandra will be used as the standard database system. It provides a robust fault tolerant system. Data is replicated throughout multiple data

centres and there is no single point of failure. Faulted nodes can be replaced without any downtime. Moreover, it offers better write performance than other NoSql database systems. Furthermore, it provides great elasticity as read and write throughput increase linearly and no machines are added without any downtime or interruption to applications due to its fundamental architectural style [3]. However, there are a few disadvantages such as ACID properties are not supported (unlike SQL) and higher latency when read operations outnumber write operations considerably. Cassandra preferes AP (from the CAP theorem) which will be explained in detail in the Trade-offs section.

3) **Business Logic:** Spring Boot will be used to implement the Rest APIs of the services. Spring boot is a Framework developed on top of the existing Spring Framework. It provides defaults for code and annotation configuration to quick start new Spring Projects. Unlike vanilla Java, it eases Java based application development, unit testing and integration process, providing the developers with more time with developing the project rather than setting up environments, writing lots of boilerplate code, annotations and XML configurations [4].

4) **Mobile Application:** The mobile application will be developed using React Native. It is an open source mobile application framework developed by Facebook Inc. It combines various components of native mobile application development with React.js in order to render native platform UI. It provides a core set of platform agnostic native components like ‘View’, ‘Text’ and ‘Image’ that map directly to the platform’s native UI building blocks. As such, apps developed using React Native are compatible on both Android and IOS. Furthermore, it provides ‘Fast Refresh’ that allows near instant feedback for changes in the React components, making the transitions in applications look very smooth [5]. However, unlike Flutter, React Native only provides essential UI elements and other third party tools have to be used for functionalities such as navigation and testing.

5) **Testing:** The application will be tested using Locust Load Testing Framework. Locust is a python-based framework used to test the amount of traffic an application can handle concurrently. This framework mimics heavy traffic by artificially adding users working on the system at once. We will write a python application to test the Microservices that are at use, such as multiple users checking the URL analytics, or multiple users trying to input and generate shortened URLs all at once. [9]

7. 12 Factor App Approach

1) Code Base: Every microservice is to be monitored in its own version control codebase. Git will be used for the codebase.
2) Dependencies: Spring boot, which will be used to implement the Rest APIs of the services, comes with Maven which helps in explicitly declaring and managing dependencies. Moreover, Node Package Manager will be used to import and declare dependencies while developing the Mobile App using react native.
3) Config: A configuration file will be maintained to store all global variables like Cache Authentications.
4) Backing Service: The Spring Boot framework will be utilized to make the database and services completely agnostic. As such, in case of any change in compatible backing service, only the config files will have to be changed.
5) Build, Release, Run: Implementation will be divided into three stages. The project will be built using Maven. The released phase will be carried out using Packer. The run phase will be done on Docker where the containers will be orchestrated by Kubernetes and hosted on Heroku.
6) Processes: The multiple endpoints of the Rest APIs will be exposed. Thus, a request on any of these endpoints will be entirely independent of any requests made before it.
7) Port Binding: Spring Boot provides a default embedded Application Server. Hence the jar generated using Maven will be capable of running on any environment having a compatible Java runtime.
8) Concurrency: The use of Kubernetes will allow each microservice to be able to scale up and down as needed. It will allow the microservices to be independent from each other and use multiple threads of the system to run concurrently.
9) Disposability: The services should be independent from each other. Hence if one of the services is stopped, or causes errors, the other services will not be affected. Furthermore, the idempotent services will be exposed.
10) Dev/Prod Parity: The use of Docker and Boot Spring will bridge the gap between development and production environment. Furthermore, as the members of the team are equally responsible for development and production, the dev/prod parity will be at the minimum.
11) Logs: Fluent ID will be used to keep the logs. These logs will be considered independent from the running processes.
12) Admin Process: The admin processes will be considered to be the part of the project rather than to be considered as a standalone software.

8. Estimations

8.1. Cost Estimations

- **Cloud**
  Kubernetes engine hosted on GCP(Google Cloud Platform) using direct deployment:
  53 n1-standard-2 GCP instance: $32,040/year [6]

- **Database**
  Apache Cassandra Developer package: $49/month [7]

- **Web Server**
  Heroku Production package: $25/month [2]

8.2. Capacity Estimations

Capacity estimations are measured by how much capacity each database uses:

- **User Database**
  User's id: Integer - 4Bytes
  User's name: Max. 50 characters - 50Bytes
  User's email: Max. 50 characters - 50Bytes
  User's hashed password: 128 characters - 128Bytes
  Assuming 1 million users at max
  Total User Database storage: 232MB

- **Short URL Database**
  The shortened URLs will be saved as Base62(A-Z, a-z, 0-9) characters
  URL ID: Integer - 4Bytes
  Short URL: 10 characters - 10Bytes
  Long URL: max. 255 characters - 255Bytes
  URL Timestamp: 8Bytes
  Total Short URL Database storage: 88GB

- **URL Analytics Database**
  URL ID: Integer - 4Bytes
  URL Title: String - 50 characters - 50Bytes
  URL Description: 255 characters - 255Bytes
  Total URL Analytics database: 10GB

In total, around **98GB** of storage will be needed.

- **CPU**
  Since we are using an n1-standard-2 GCP instance, following statistics are applied [10]:
  - **# of vCPUs:** Up to 96
  - **Memory:** Up to 6.5GB per vCPU
  - **Bandwidth:** Up to 10GBps

9. Metrics to be monitored

9.1. Service Availability and Uptime

The reliability and availability of the servers ensures all the requests are responded back and none of them are discarded [11]. The service availability can be monitored and measured through Load Testing (Locust Load Testing Framework) [8]. According to the SLA, the system needs to be up 99.0% of the time.

9.2. Latency

As defined in the SLA, the Latency of our application for redirection should be within 400ms (averaged). In order to monitor the latency of our application we will use Prometheus [12].

9.3. ShortURL uniqueness

The mentioned in the SLAs URL shortener service will create unique shortened URLs which will be 100% unique. Prometheus monitoring system can also be used for testing the generation of unique keys [12].

10. Trade-off analysis

10.1. Customization VS Security

The shortened URLs will be saved as Base62(A-Z, a-z, 0-9) characters allowing for a wide range of short URLs to be created for use also allowing the user's will be able to create their own unique custom URLs. However, as a result, malicious short URLs can also be created which when clicked can redirect the user to a malicious website leaving the system vulnerable. Therefore, while allowing for customization for users to create their own short URLs hackers may be able to exploit this feature to redirect people towards compromised websites.

10.2. Scalability VS Cost

As can be seen in section 8.2, the system will use 98GB of storage (estimated). However, if the storage amount increases then as a result the operating expenses (OPEX) will also be increased to allow for more resources to be allocated to the system in order for the system to continue functioning.

10.3. Extensibility VS Complexity

The shortened URLs will be saved as Base62(A-Z, a-z, 0-9) characters, i.e. 0 - 9 (10 chars), a-z (26 chars), A-Z (26 chars) = 62 chars which allows for 62! / (62 - 10)! = 62! / 52! = 3.9016471e+17 possible shortURLs. However, by allowing a large number of possible short URLs, the complexity of the system increases making it difficult to maintain the database and the program. It is to be noted that in the future based on users' requirements the length of the ShortURLs will be susceptible to change.

10.4. Availability and Partition Tolerance VS Consistency

The database used for our system is Cassandra. Cassandra supports AP in the CAP theorem. As a result our system will be prioritizing Availability (all nodes must have the same processing speed at maximum reasonable amount of time) and Partition Tolerance (if there is a partition between nodes and they are unable to talk to each other, the system should still be functioning) over Consistency (Each node gets to read or write into the same data i.e. the data appears similar to every node).

11. Testing Strategies

We will implement a set of testing strategies to find out whether our application runs based on the requirements mentioned in the SLA (Service Level Agreement). Based on the SLA and the testing strategies needed to be run on the system, we planned to use Locust for the testing purposes. [8]

11.1. Load Testing

Load testing will be used to find out how much traffic can the system handle when multiple users are using the system actively and concurrently. It is a subsection of Performance Testing. [9]

11.2. Functional Testing

This testing strategy ensures all functional requirements of the system are being addressed according to the SLA.

11.3. Security Testing

Security testing is for maintaining the authentication of the system, so that only registered users can generate shortened URLs, or see the analytics of a shortened URL.

References

[1] A. Chakraborty, “System Design Analysis of TinyURL,” *Medium*, 12-Oct-2020. [Online]. Available: https://towardsdatascience.com/system-design-of-url-shortening-service-b325b18c8f88. [Accessed: 01-Apr-2021].

[2] “Heroku,” *Cloud Application Platform*. [Online]. Available: https://www.heroku.com/. [Accessed: 02-Apr-2021].

[3] “Manage massive amounts of data, fast, without losing sleep,” *Apache Cassandra*. [Online]. Available: https://cassandra.apache.org/. [Accessed: 02-Apr-2021].

[4] “Spring makes Java simple.,” *Spring*. [Online]. Available: https://spring.io/. [Accessed: 02-Apr-2021].

[5] “Spring makes Java simple.,” *Spring*. [Online]. Available: https://spring.io/. [Accessed: 02-Apr-2021].

[6] “The Ultimate Kubernetes Cost Guide: AWS vs GCP vs Azure vs Digital Ocean,” *Kubernetes Governance and Cost Management for the Cloud-Native Enterprise*. [Online]. Available: https://www.replex.io/blog/the-ultimate-kubernetes-cost-guide-aws-vs-gce-vs-azure-vs-digital-ocean. [Accessed: 01-Apr-2021].

[7] https://www.g2.com. 2021. *Managed Apache Cassandra*. [online] Available at: <https://www.g2.com/products/managed-apache-cassandra/pricing> [Accessed 1 April 2021].

[8] “An open source load testing tool.,” *Locust*. [Online]. Available: https://locust.io/. [Accessed: 01-Apr-2021].

[9] B. Lyddon, “Load Testing with Locust,” *Medium*, 24-Jan-2018. [Online]. Available: https://blog.realkinetic.com/load-testing-with-locust-part-1-174040afdf23. [Accessed: 02-Apr-2021].

[10] “Machine types | Compute Engine Documentation | Google Cloud,” *Google*. [Online]. Available: https://cloud.google.com/compute/docs/machine-types. [Accessed: 02-Apr-2021].

[11] “The 4 types of metrics you should monitor to keep your servers under control,” *The 4 types of metrics you should monitor to keep your servers under control - Site24x7 Blog*. [Online].

Available:
https://www.site24x7.com/blog/the-4-types-of-metrics-you-should-monitor-to-keep-your-servers-under-control. [Accessed: 02-Apr-2021].

[12] Prometheus, “Prometheus - Monitoring system & time series database,” *Prometheus Blog*. [Online]. Available: https://prometheus.io/. [Accessed: 02-Apr-2021].