CS7NS6 Distributed Systems - Exercise 2 Report

Title: Implementing a Globally-Accessible Distributed Traffic Service
Group: D
Date: March 10, 2025
Authors: Khaund Abhigyan, Gong Linyun, Guo Jiaqi, Duddupudi Daya Lokesh, George Sibin,

1. Introduction

This project aims to design and implement a globally accessible distributed traffic service that enables drivers to book or cancel journeys while ensuring that no driver begins a journey without a confirmation notification. The system is designed to deliver high performance, scalability, availability, and reliability for millions of users worldwide while maintaining cost efficiency. A key feature of the system is its ability to handle region-level failures by ensuring that if a region fails, the load balancer redirects requests to another healthy region.

2. System Requirements

2.1 Functional Requirements

- Drivers can book journeys by providing details such as route, start and end locations, booking time, driving license, vehicle details, email, and a secret code, direction.
- Drivers can cancel booked journeys.
- The system notifies drivers of journey acceptance or rejection.
- Journey booking must be made at least one hour before the journey begins.
- A threshold is set at 80% occupancy for routes; if exceeded, drivers must try again later.
- The system validates locations to prevent booking errors.

2.2 Non-Functional Requirements

- Performance: Sub-second latency for booking, cancellation, and notifications.
- Scalability: Supports millions of users with horizontal scaling.
- Availability: 99.99% uptime, even during region-level failures.
- Reliability: Ensures data consistency and prevents loss of critical information.
- Global Accessibility: Provides low-latency access worldwide via geo-routing.
- Cost Efficiency: Optimized for dependability while controlling operational costs.

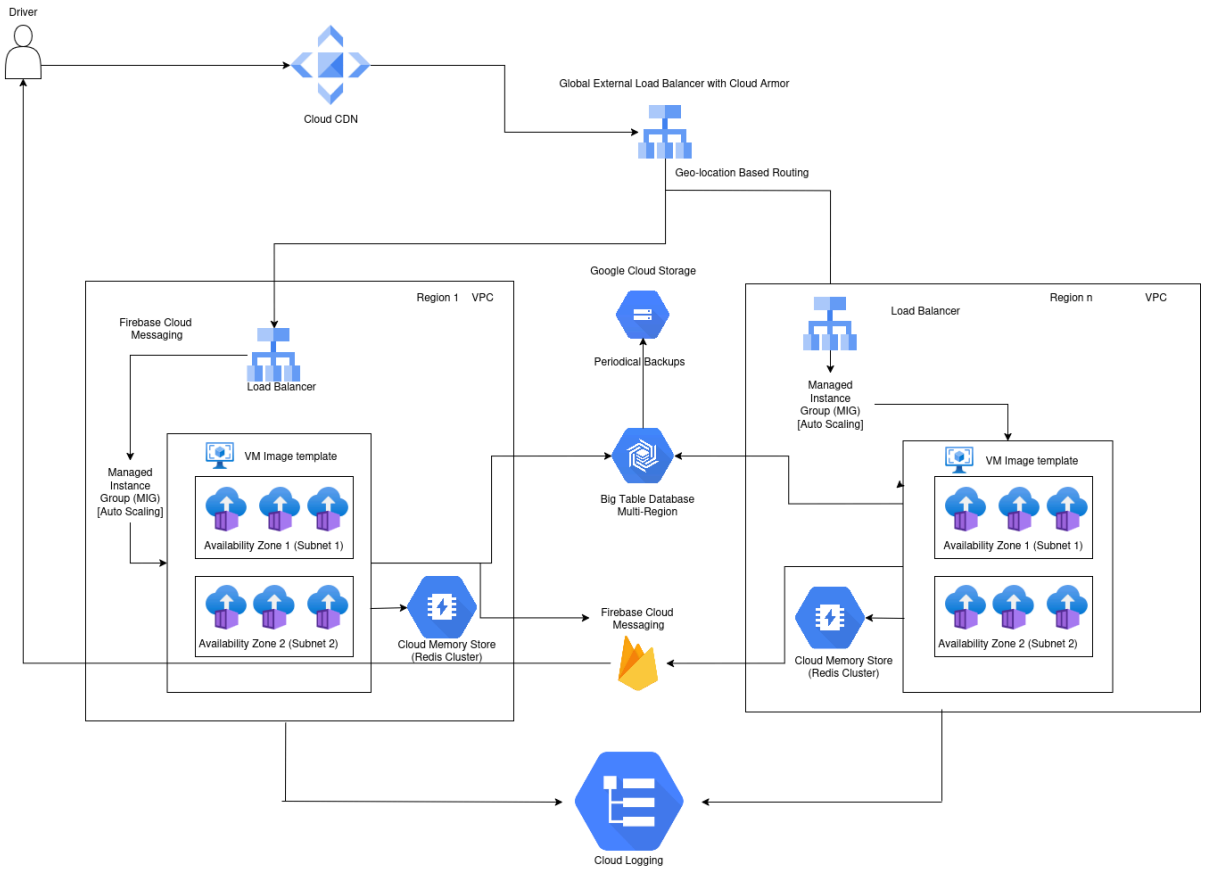
2.3 Usage Patterns Expected

- Global users require low-latency access.
- Write-heavy traffic during peak hours.
- Frequent reads for journey status updates.
- Popular routes benefit from caching.

2.4 Failure Model

- Server failure: Considered unhealthy in the Managed Instance Group (MIG), and traffic is no longer routed to it.
- Zone-level failure: Other availability zones (AZs) handle the workload.
- Redis node failure: Other nodes in the cluster handle requests.
- Redis cluster failure: Bigtable database is used as a backup.
- Region-level failure: The Global Load Balancer redirects requests to another region.

3. Technical Architecture



The system is built on Google Cloud Platform (GCP) with a focus on high availability, scalability, and failover mechanisms.

3.1 Architecture Overview

- Cloud CDN: Caches the static frontend application across edge locations for low-latency access, ensuring quick loading times and global availability.
- Global External Load Balancer with Cloud Armor: Routes traffic to the nearest region using geo-location-based routing. If a region becomes unavailable, the load balancer redirects requests to another healthy region. It also protects against DDoS attacks.

- Regional HTTP(S) Load Balancer: Distributes incoming requests across multiple availability zones within a region, ensuring fault tolerance and high availability.
- Managed Instance Groups (MIGs) with Auto-Scaling: The backend services are deployed in MIGs, which dynamically scale based on a Target CPU Utilization Policy (70%), ensuring optimal resource allocation.
- Cloud Memorystore (Redis Cluster): A high-throughput in-memory database that caches active booking data to minimize latency and improve read performance.
- Bigtable (Multi-Region, Eventual Consistency): A NoSQL high-throughput, low-latency database used for storing persistent booking data. It provides eventual consistency to optimize system responsiveness.
- Google Cloud Storage: Used for periodic backups to ensure data durability and disaster recovery.
- Cloud Logging & Monitoring: Tracks application performance, detects anomalies, and alerts administrators about failures.
- Firebase Cloud Messaging (FCM): Delivers real-time push notifications to drivers regarding booking status updates.

3.2 Technologies Used

1. Google Cloud CDN: Caching static frontend content at edge locations to improve load times and reduce latency for users worldwide.
2. Global External Load Balancer with Cloud Armor: Ensures efficient geo-location-based routing while protecting the system from DDoS attacks.
3. Regional HTTP(S) Load Balancer: Distributes traffic efficiently across instances within the VPC.
4. Google Cloud Managed Instance Groups (MIGs): Enables automatic scaling of backend services based on demand.
5. Google Cloud Memorystore (Redis Cluster): Provides high-speed caching and minimizes Bigtable query loads for frequently accessed data.
6. Google Cloud Bigtable: Used as a high-throughput, multi-region NoSQL database with support for eventual consistency.
7. Google Cloud Storage: Ensures long-term data durability and disaster recovery through scheduled backups.
8. Google Cloud Logging & Monitoring: Provides a centralized dashboard to track system health and troubleshoot issues.
9. Firebase Cloud Messaging (FCM): Sends real-time notifications to drivers for booking confirmation and cancellations.

4. Design Considerations

1. Caching & Eventual Consistency:
    - Redis is the primary database for high-throughput operations.
    - Bigtable is updated asynchronously to maintain eventual consistency.
    - The 80% route occupancy threshold helps manage consistency issues by balancing driver allocations.
2. Failover & High Availability:

    - If Redis fails, Bigtable is used as a fallback.
    - If a region fails, the Global Load Balancer ensures traffic is routed to another region.
    - If a zone fails, workloads are redistributed to healthy zones within the region.
3. Scaling Strategies:

    - Increase instance capacity in MIGs based on CPU utilization thresholds.
    - Expand Redis Cluster nodes to handle increasing loads.
    - Scale Bigtable's Global Tables to optimize performance for large datasets.

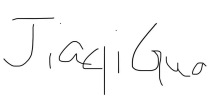
5. Logging & Monitoring

- Cloud Logging: Offers real-time insights into API requests, database interactions, and system performance.
- Cloud Monitoring: Provides automated alerts and anomaly detection.
- Cloud Armor: Protects against DDoS attacks and malicious requests.

6. Conclusion

This design ensures high availability, fault tolerance, reliability, and scalability. The Global Load Balancer guarantees regional failover, Cloud Memorystore & Bigtable provide a robust data layer, and MIGs auto-scale dynamically based on demand. The system meets all project goals efficiently.

Signatures

Things to add
Queue to architecture to handle race conditions in eventual consistency