Cheap stock system design by Peter Chencha

Users

1. Stock brokers.
2. Business owners.
3. Business stakeholders.
4. Employees of businesses listed in the stock exchange.

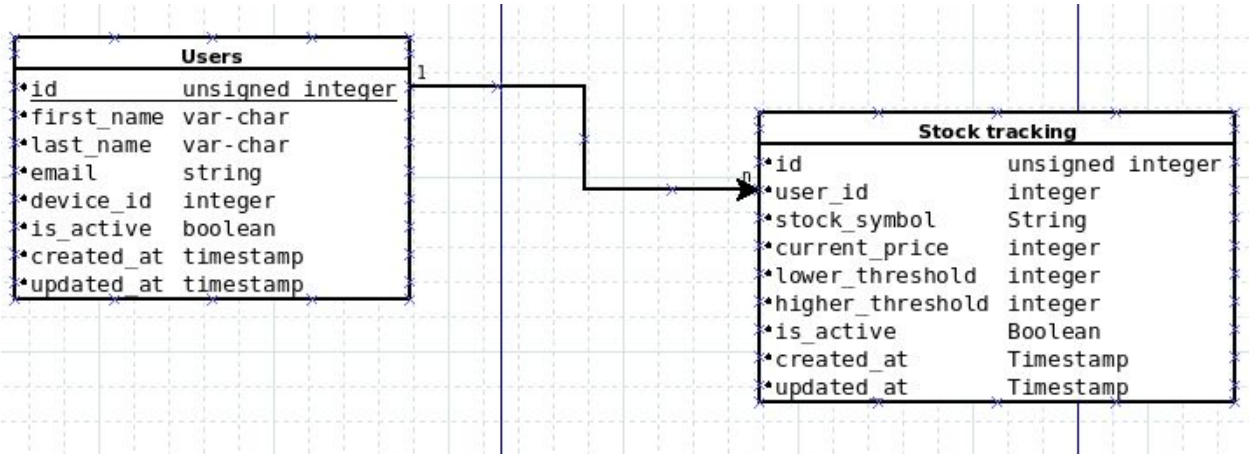
Use cases:

1. User specifies which stock symbol to track.
2. Uses specifies the threshold of stock price to receive notification.
3. User is not anonymous.
4. Service has high availability.
5. Service needs to evolve from serving a small number of users to millions of users.

Constraints and Assumptions:

1. Only stocks of companies in the USA are tracked.
2. No historical data of stock price is required.
3. Need to track multiple stocks simultaneously.
4. Traffic is not evenly distributed. Popular stocks will have more searches such as apple.
5. Need for relational data.
6. Stock is no longer tracked when either of the thresholds is passed.
7. Scale from one user to tens of thousands. Increase in user will be denoted by; user+, user ++ and user +++.
8. Owing to the real time nature of stock prices, implementing a cache to store common queries is not advisable as we'll get outdated prices.

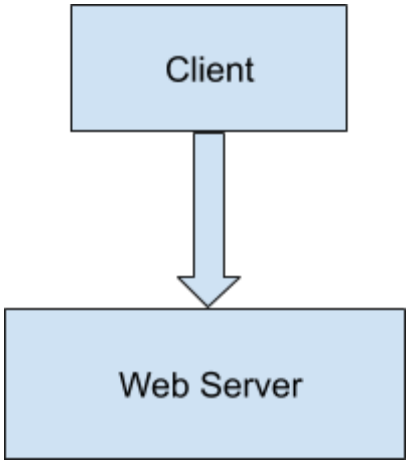
Database schema



We will have two tables; a users table(index is id) to store our users' information and stock tracking(index is id with user_id as foreign key) where the details of the stock being tracked will be stored.

The relationship between them is one to many i.e one user can be tracking multiple stock symbols simultaneously.

High-Level Design.



Goals.

With only 1 to 10 users, we'll only need a basic setup:

- Single box for simplicity.
- Vertical scaling when needed.
- Monitor to determine bottlenecks.

How the system will track current price.

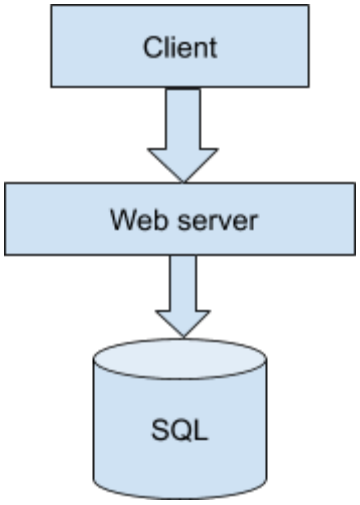
- We'll have a query service running inside the server which will take stock symbol as input, scrape google for current price and store the results in the stock trading table.

How the system will know when to notify users.

- We'll have a notification service running inside the server. It will pick the stock symbol from the stock trading table, run it in the query service and compare the results with the thresholds set in the stock trading table, if above or below will send a notification to the user using the user_id to pick user details from the users table.

Scale the Design

USER +



Assumptions:

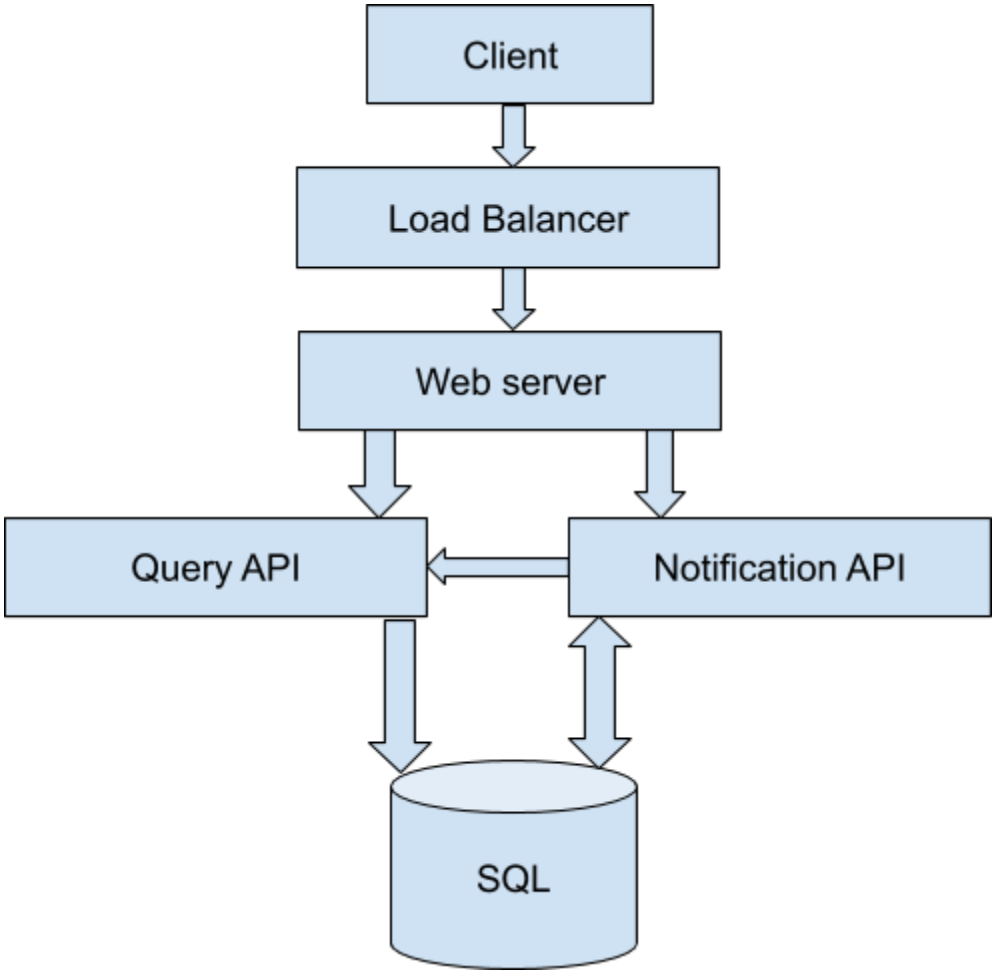
Our user count is starting to pick up and the load is increasing on our single box. Our **Benchmarks/Load Tests** and **Profiling** are pointing to the **MySQL Database** taking up more and more memory and CPU resources, while the user content is filling up disk space.

We've been able to address these issues with **Vertical Scaling** so far. Unfortunately, this has become quite expensive and it doesn't allow for independent scaling of the **MySQL Database** and **Web Server**.

Goals

- Lighten load on the single box and allow for independent scaling
    - Move the **MySQL Database** to a separate box
- Disadvantages
    - These changes would increase complexity and would require changes to the **Web Server** to point to the **MySQL Database**
    - Additional security measures must be taken to secure the new component.
    - Hosting costs could also increase.

USER ++



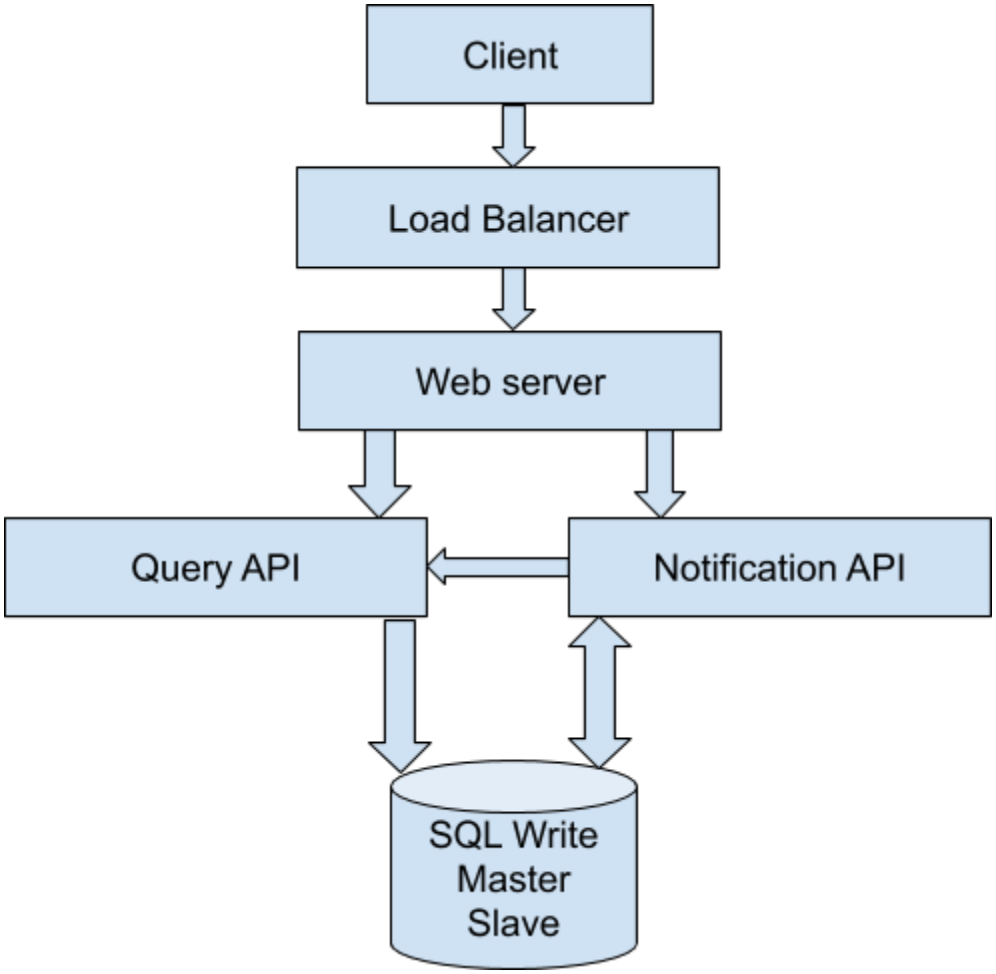
Assumptions

Our **Benchmarks/Load Tests** and **Profiling** show that our single **Web Server** bottlenecks during peak hours, resulting in slow responses and in some cases, downtime. As the service matures, we'd also like to move towards higher availability and redundancy.

Goals

- We will use horizontal scaling to handle increasing loads and to address single points of failure
    - Add a load balancer such as Amazon's ELB or HAProxy
        - ELB is highly available
        - If you are configuring your own **Load Balancer**, setting up multiple servers in active-active or active-passive in multiple availability zones will improve availability
        - Terminate SSL on the **Load Balancer** to reduce computational load on backend servers and to simplify certificate administration
    - Use multiple **Web Servers** spread out over multiple availability zones
- Separate out the **Web Servers** from the **Application servers**.
    - Scale and configure both layers independently
    - Web Servers can run as a reverse proxy server.
    - For example, you can add Application Servers handling **Query APIs** while others handle **Notification APIs**

USER +++



Goals

- To improve redundancy (since our system is expected to be high availability) we will set up a mysql instance with master-slave failover mode.

USER ++++

Assumptions

Our Benchmarks/Load Tests and Profiling show that we are read-heavy and our database is suffering from poor performance from the high read requests.

Goals.

Add MySql read replicas to reduce load on the write master:

- Add logic to **Web Server** to separate out writes and reads
- Add **Load Balancers** in front of **MySQL Read Replicas**