Architecture

Software Systems Scalability

Software Engineering 468

University of Victoria

Spring 2020

Team: Extreme Workload

Kian Gorgichuk V00855771

Devlyn Dorfer V00846516

Goh Sato V00850584

Table of Contents

Glossary

System	Refers to the entire Day Trading Application software system.
MongoDB	A document oriented database program.
NoSQL Database	A type of database.
Golang	A programming language created by Google.
SSL (Secure Socket Layer)	Security technology that establishes an encrypted link between a server and a client.
Master slave pattern	A database replication technique where a single database is the source of truth, while an arbitrary number of slave databases are synchronized to it.
Horizontal scaling	Scaling by adding more machines to a distributed system.
Internal Server	Refers to a single internal server within the system.

1.0 Introduction

This document describes the planned architecture of the Day Trading Application. It is intended to document and justify the key decisions that were made in designing the architecture of the system. By doing so, the document addresses the main objectives of the project such as the reliability, performance, and fault tolerance.

2.0 Technologies

Here are the technologies that Extreme Workload will use to build the system.

2.1 Golang

The system is to be written in a programming language called Golang because it offers several key features. First, Golang provides powerful and built in concurrency tools (goroutines) that are both performant and easy to use. This allows the system to take advantage of multicore processors in the lab machines that the system will be run on. Additionally, Golang provides an attractive balance of low level control for high performance while maintaining ease of use with helpful abstractions. Golang is also a compiled language resulting in better performance than scripting or virtual machine based languages.

2.1 MongoDB

The system will use the database program MongoDB since the data that must be stored is simple (does not require relational database operations) and to take advantage of the speed that a NoSQL database, such as MongoDB provides.

2.2 Docker

The system is to be built and deployed on docker, an OS level virtualization tool, to ensure it can run in any environment. This ensures the system runs on the provided lab machines. Each component of the system has its own Docker container making the system easily deployable into many distributed configurations.

3.0 System Architecture

The following section describes the architecture of the system. More specifically, it identifies the internal components of the system and how they will be organized to meet the project objectives.

3.1 Overview

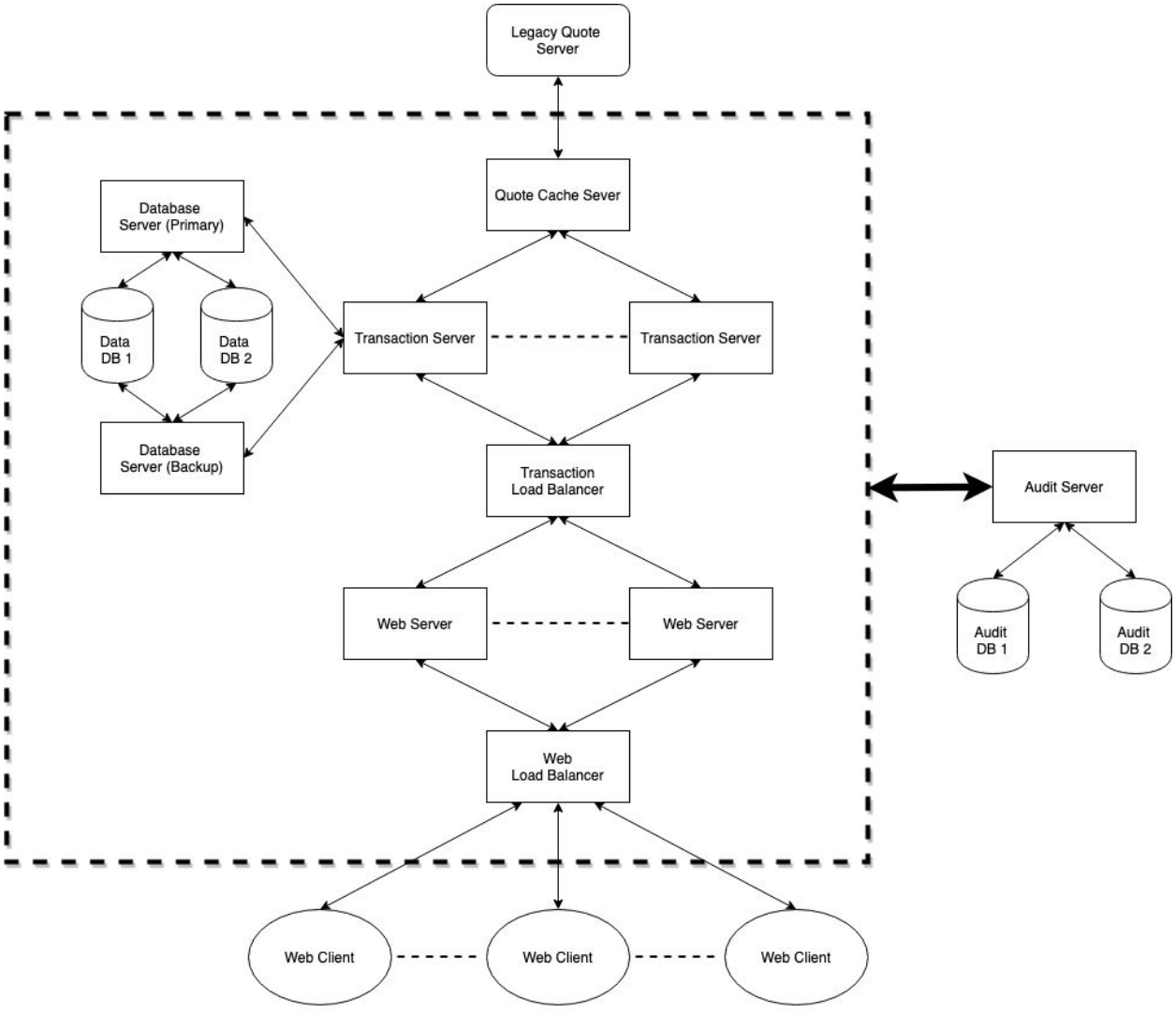

Figure 1. High Level Overview of Architecture.

Web clients serve as the entry point to the system. Requests from web clients are sent to the web load balancer. The system contains many web servers to prevent a single point of failure, as well as to scale horizontally given a large workload. This is represented in figure 1 by the small dots between the two web servers.

Similarly, the system contains a transaction load balancer and multiple transaction servers to prevent a single point of failure and to enable horizontal scaling. This is represented in figure 1 with the small dots between the two transaction server. Transaction servers interact with both the quote server and the database server. Note that every transaction server will have a connection to the database servers.

The database server interfaces with multiple database instances that follow the master slave pattern. This provides redundancy and reduces the chance of data loss to a database error.

The legacy quote server provides quotes that are valid for 60 seconds. To reduce the amount of requests to the quote server, the system utilizes a caching layer to cache already retrieved quotes.

Except the web clients and the legacy quote server, all components of the system interact with the audit server. This is represented in figure 1 by the large box surrounding several components within the system and the arrow between it and the audit server. Similar to the database servers the audit server utilizes multiple database instances that follow the master slave pattern to provide redundancy and reduce the chance of data loss for the system logs.

3.2 Web Load Server

The web server has three main responsibilities. These responsibilities are to:

1. Decrypt the SSL connection
2. Determine if the parameters for a command are valid
3. Route the request to the transaction load balancer or the audit server

The web server acts as the first contact between the client and the entire system. All commands from web clients will reach web servers within the system. By validating requests at this stage, any bad requests are immediately denied and incur minimum cost to the system. The web server also logs requests directly to the audit server, thus skipping the transaction server. This reduces the workload of the transaction server.

As described in the system overview the system maintains multiple instances of web servers for fault tolerance if one or multiple web servers go down. Requests are routed from web clients into the web load balancer as described in the load balancer section of this document.

3.3 Transaction Server

The transaction server is where the majority of the processing within the system is completed. The transaction server handles all the business logic of the system. As described in the system overview, it interacts with the database server, audit server, and quote cache server to handle the following commands:

- ADD
- QUOTE
- BUY
- COMMIT_BUY
- CANCEL_BUY
- SELL
- COMMIT_SELL
- CANCEL_SELL
- SET_BUY_AMOUNT
- CANCEL_SET_BUY
- SET_BUY_TRIGGER
- SET_SELL_AMOUNT
- SET_SELL_TRIGGER
- CANCEL_SET_SELL
- DISPLAY_SUMMARY

The system maintains multiple instances of the transaction server to provide horizontal scaling, as well as the ability to recover from failure. The transaction load balancer routes requests between the web server and the various transaction servers. The transaction load balancer also ensures that requests are sent to operational transaction servers. For example, if a transaction server becomes unavailable, the transaction load balancer will not send requests to that server.

3.4 Quote Cache Server

The quote cache server will serve as a proxy to the legacy quote server. The following commands require quote information and will therefore send requests to the quote cache server:

- QUOTE
- BUY
- COMMIT_BUY
- COMMIT_SELL

As described in the system overview, the purpose of the quote cache server is to reduce the number of requests to the legacy quote server. This is to reduce the overall cost of transactions. Rather than request a new quote from the legacy quote server every time a quote is required, the quote cache server caches requested quotes until they are no longer valid.

To mitigate that fact there is only one quote cache server in the architecture, transaction servers will automatically bypass the quote cache server if the quote cache server is unavailable. The servers would directly connect to the legacy quote server. This bypass would only occur until the quote cache server is restarted. While this may increase the quote cost to users, the bypass mechanism increases reliability and fault tolerance of the architecture by eliminating the quote cache server as a single point of failure.

3.5 Audit Server

The purpose of the audit server is to store audit data generated by the system. Every component of the system sends requests to the audit server whenever it is sending or receiving data to or from another component. This is done to ensure each aspect of the system is traceable and by extension debuggable.

The system will contain only one audit server instance. To increase the fault tolerance of the system, requests to the audit server are queued by the client server. For example, a web server

attempts to send a request to the audit server and the audit server is unavailable. The web server will then queue the audit server request until the audit server comes back online. If the web server's queue becomes full, then the request would be written to the web server's logs and the request is discarded. This enables the system to continue to operate if the audit server fails and restarts while maintaining complete auditing of the system. Additionally, to decrease the chance of data loss and increase fault tolerance, the audit server manages several database instances that follow the master slave pattern.

Additionally, the audit server also has a heart beat monitor which sends a heartbeat request to each internal server on a regular interval. If a server is found to be unresponsive, the audit server will log the disruption and trigger a process to notify a system maintainer (for example, contacting an on-call engineer). This process is to be defined by the client at a later date..

3.6 Database Server

The purpose of the database server is to store data related to users of the system, and the users' buy and sell triggers. The data models for database are shown in figure 2.

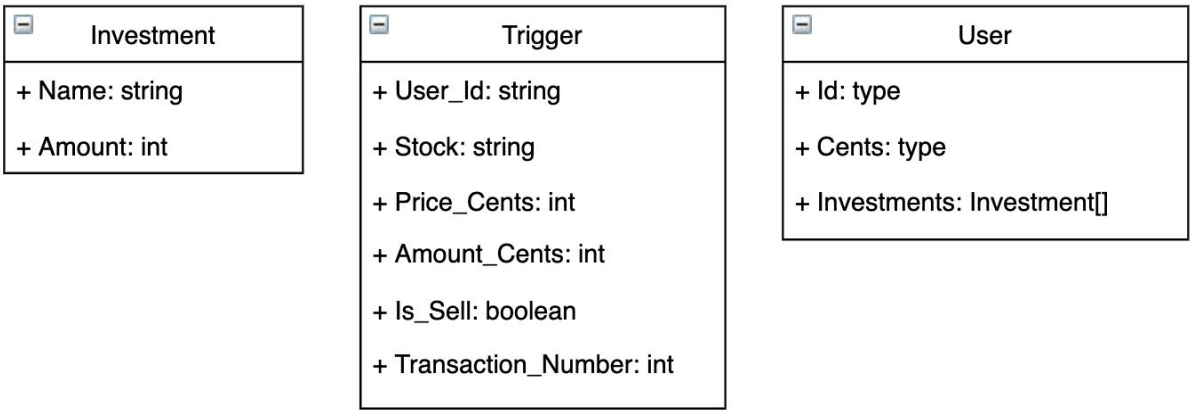

Figure 2. Data models for data server.

Specifically, the database server it utilized in the following requests:

- ADD
- SELL
- CANCEL_BUY
- CANCEL_SELL
- COMMIT_BUY
- COMMIT_SELL
- SET_BUY_AMOUNT
- CANCEL_SET_BUY
- SET_BUY_TRIGGER
- SET_SELL_AMOUNT
- SET_SELL_TRIGGER
- CANCEL_SET_SELL
- DISPLAY_SUMMARY

Note that the system will contain two database server instances. One database server instance is designated the primary server while the other is designated a backup server. If the primary database server goes offline then all transaction servers use the backup database server. The use of the backup database server will occur until the primary database server becomes available again. The reason a two server approach is used for the database servers is to prevent the system from going offline if a single database server goes down (when this happens the audit server will start the server recovery process). Likewise, to further increase the fault tolerance of the system the database server will manage and utilize multiple database instances that follow the master slave pattern to reduce the chance of data loss.

3.7 Load Balancing Servers

The system contains multiple instances of transaction servers and web servers for reasons stated in their respective sections. This requires the use of a load balancer for each component to route requests to the various instances as shown in figure 1.

The load balancers ensure that a failed instance is not given requests, allowing the system to continue operating and recover. Load balancers will initially use the round robin method to route the requests but this load distribution method may be changed when testing of the system is completed.

Note that the transaction and web load balancing servers are considered single points of failure in the architecture, as all requests for both the web server and transaction servers must be handled by the load balancers. Therefore, the load balancers' responsibilities will be minimized to only load balancing to reduce the chance of failure.

4.0 Conclusion

The system will be developed with Golang, utilize MongoDB, and be deployed using Docker. The system is composed of seven main components: web servers, transaction servers, the data server, the audit server, cache server, as well as two load balancers. The web server acts as the entry point to the system for web clients and is responsible for input validation, SSL encryption and routing requests to either the audit server or the transaction server. The transaction server interfaces with several other system components to handle the business logic. The data server is responsible for storing data related to both users and triggers. The audit server interacts with all components of the system to store logs for all events and to ensure that all internal servers are available. There are two load balancers within the system one is responsible for sending requests to various instances of web servers, while the other is responsible for sending various requests to the transaction server. Overall, Extreme Workload is confident that this architecture fulfills the requirements of the project and provides a scalable, reliable, and fault tolerant system.