PaperTrader Protocol Specification

altffour

May 18, 2020

Contents

1 Introduction

This is the document for the specification of PaperTrader. PaperTrader is an application for 'fake' trading assets, to practice investing. The document contains explainations on how to implement the papertrader application. It should be noted that the document isn't 'production-ready' until this sentence is removed. The document will go over the roles of the master server, and the worker servers, how they interact with eachother and the communication protocol, and finally, suggestions on server side implementations.

2 Overview

This section contains the required terminology and modelling of the PaperTrader infrastructure.

2.1 Terminology

2.1.1 Inner World

This is Master server, and all worker servers. This should be kept under high lockdown. Meaning, critical data should be kept secure.

2.1.2 Outer World

This is the frontend, including the desktop cleint, mobile client, or the website client. The data here is controlled by the authorization of the account.

2.1.3 Critical Data

Cirtical Data are all data types that shouldn't be tampered with without authorization. For example, accounts, personal information, messages, and in this context user's portfolios.

2.1.4 User/Client

In this context it is the frontend, which is either the desktop client, mobile client, or the website client.

2.1.5 User/Client Data

This is the data of the user. The meaning depends on the specific conext. It could mean the personal information, credentials, etc. Most of the time it means data that is attached to a data transfer to identify client (IP?).

2.1.6 Master Server

This is the main server that MUST be run when deploying the application. Contains critical data, it would only interact to the outside world by the worker servers.

2.1.7 Worker Servers

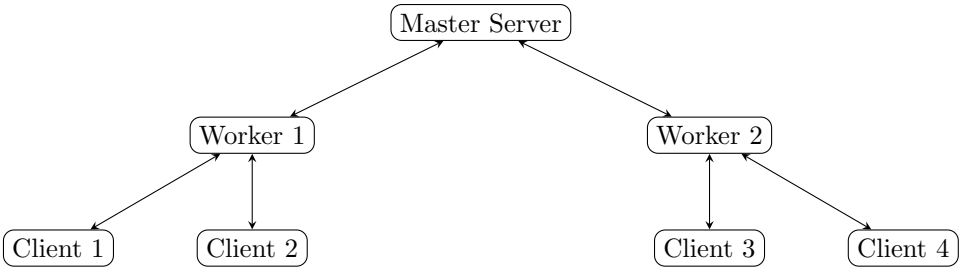
These are servers that contact the outer world. Worker Servers will interact with the Master Server acting like a 'cache' servers. Data should be routed through worker servers to the master server. The main job for worker server is to add timestamps onto commands sent from the user. The data sent to the main server must contain the data of the client/user. There MUST be ATLEAST one instance running to have a functional infrastructure.

2.1.8 User Accounts

This is the account that abstractly is a the data structure that contains information about hte user and their account.

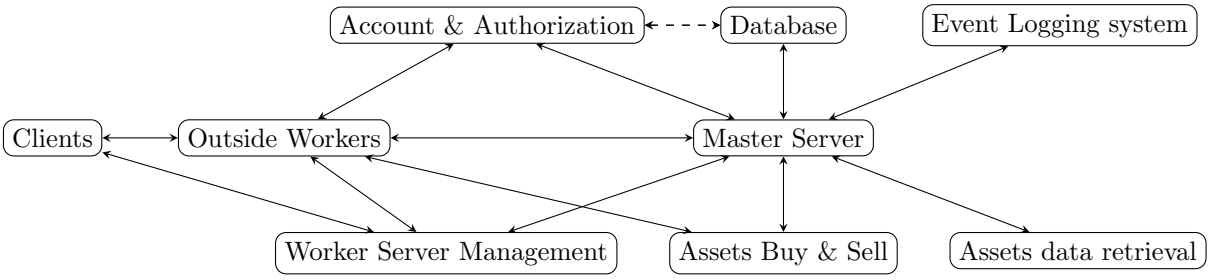
2.2 Infrastructure Model

A fully deployed infrastructure cotains *ONE* master server, *ATLEAST* one worker server, theoretically across the world to maintain speed and reliabilty. An overview diagram of the infrastructure:



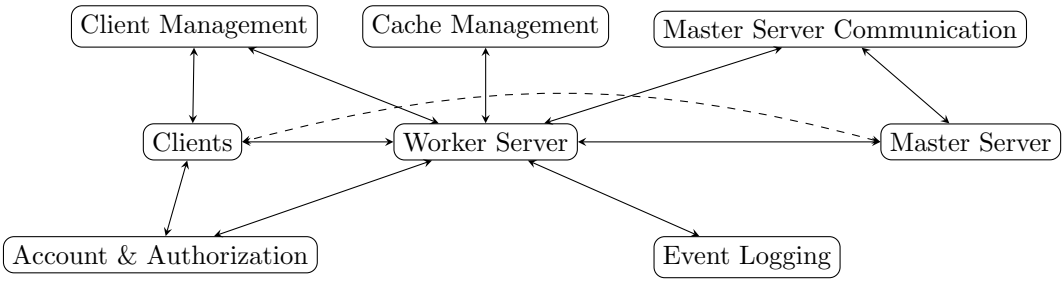
2.2.1 Master Server Infrastructure Model

The master can be defined into modules as demonstrated in the following diagram:



2.2.2 Worker Servers Infrastructure Model

The worker servers can be defined into modules as demonstrated in the following diagram:



2.3 Global Deployment Variables

This section contains an overview of the global deployment variables.

2.3.1 List of assets to retrieve

This is the list of assets to retrieve using the assets/stocks API. The list can be available in a file or hard-coded into the implementation.

2.3.2 Number of Workers

This is the number of workers deployed with the master server. It must be atleast one. The worker server preferably should be deployed regionally.

2.3.3 Memory Size of Log system

This is a technical variable, this is the size of the log in memory before it being flushed to harddisk. Generally the smaller this is the more disk speed is required. And the larger it is the more RAM the instance needs and the faster it is.

2.3.4 Stock Data Update Interval

This is the interaval of the stock data retrieval. The more this is the faster the transactions that can occur in a minute. This should be planned perfectly so that it can maintain the userbase with the API calls.

3 A more Technical Overview

This is the section that describes the functioning parts of the project in detail. We will start with modules, including master server modules, and worker server modules.

3.1 Master Server

The master server has multiple modules:

- Main Module, i.e the driver.
- Database Management.
- Account Management.
- Event Logging System.
- Worker Management.
- Assets Data Retrieval.
- Assets Transaction Management.

3.1.1 Main Module

The main module should be able to do the following things:

- Start the authorization thread.
- Start the Worker Management thread.
- Start the assets data retrieval thread.
- Start the assets transaction thread.
- Be able to parse commands from the worker threads.
- Be able to route the commands to the correct thread.

The main module's functionality in relation of the deployment and running stage is as follows, The binary containing the master server is run -¿ initializes states required to operate the server -¿ start the threads -¿ start listening to workers -¿ parse it -¿ pass it to the appropriate thread. This is usually the set of functions that the main function would call. Putting the workings of the main module on a seperate is advised, since it gives the ability to crash the server and dump the logs from the memory of the event log system. Refer to 2.3.3 for insight on why this is recommended.

3.1.2 Database Management

The database management module sould be able to do the following things:

- Be able to manipulate files (create, delete, write, read).
- Be able to convert data representations (structs) into SQL Databases.

The manipulations of files should be quite straightforward, a couple of functions. The ability to access an SQL database is also necassery.

3.1.3 Account Management & Authorization

The account management & authorization module should be able to do the following things:

- Be able to register new users.
- Be able to login *AND* authorize users.
- Be able to return a session token for the user.
- Be able to manage those session tokens.

The module should be able to take the set of information given and put them into the database (using the database management 3.1.2). All passwords should be hashed and salted, this is up to the implementation on the exact details. The accounts registered may contain third-party logins ex. Google Logins. In that case the account *MUST* be recognized as an account without a password, and the user should be asked to sign in with the third-party credentials. It should also be possible to add a password to the account marked to be 'logginable' with third-party logins, making it possible to login with the password and using third-party logins.

3.1.4 Log System

The log system should be able to do the following things:

- To capture the date and time.
- To capture the caller's file, function, and line number.
- To capture a message and be able to format it.
- To be able to store it in a file.

One thing should be noted, the log system should not store in memory more entries than specified in the global variable: memory size of log system (2.3.3).

3.1.5 Worker Management

The worker management module should be able to do the following things:

- Keep track of worker servers.
- Print information about worker servers.

- Retrieve information aobut worker servers.
- *ONLY* allow worker servers that are registered.
- Verify worker servers with gpg keys.
- Able Boot off worker servers while running.
- Able to give client list of servers.

Most of the functionalities' details can be implementation depended. Keeping track of worker servers can be done in multiple ways. Printing information can print stored information about the connected worker server, or retrive information about the worker server from the worker server. Should only allow registered/allowed worker servers to connect to master server AND show in the list of available worker servers. Registration should be done using GPG keys. A list of IPs should be given to the client when a session is connected. The list of IPs are the workers' IPs.

3.1.6 Assets Data Retrieval

The assets data retrieval model should be able to do the following things:

- Retrieve data in intervals of the global variable: data update retrieval interval (2.3.4).
- Store them in memory and be able to read them (Parsed, into a struct).
- Be able to communicate with the assets API.
- Retrieve list of assets using API (2.3.1)

The assets API is upto the implementation. Assets Data should be stored in the memory and retrieved in a thread-safe manner. This module is preferably to be run on a thread.

3.1.7 Assets Buy & Sell

Thie assets buy & sell module should be able to do the following things:

- Buy assets and store them into users' portfolios.
- Sell assets and store them into users' portfolios.
- Validate transactions of buying and selling.
- Log transactions to users profiles.
- Process queued transactions per update interval (2.3.4).

The module should provide functions to apply the above functionality. The logging is *NOT* to be done with the Log sytem, but rather with storing them on the account transaction history of the issuer of the transaction. The history should be able to show the time of the transaction going through. All transaction go through a queue. The queue is cleared every update interval.