Go Language Take Home Challenge v2022.11

Nature of The Challenge

We want to get to know you much deeper on how you think as an engineer and the level of your craftsmanship when building software.

In this challenge we want to know your approach and your solution on given complex tasks in a limited time-frame. We weight each task with **points** and you are expected to optimize it your way to record your best score given **limited time to complete the challenge**.

Taking longer than the allocated time will negatively affect on our evaluation of your submission.

Please send your solution to go-backend-hiring@stockbit.com.

This challenge is confidential and you must not share nor publish it anywhere else.

Guidelines & Rules

1. You have four (4) days to implement the solution.
2. We are really interested in simple solutions, but if you also account for these factors as well, we will surely reward you with extra points:
    a. Scalability
    b. Testability
    c. Maintainability
3. Use Go version `1.20` and `go mod` for package manager.

4. We’re also interested in the understanding on how you make assumptions when building software. If a particular workflow or boundary condition is not defined in the problem statement below, what you do is your choice.
5. Code your solution using Go language.
6. Stockbit is a huge fan of Linters, especially `golangci-lint`. We provide you with the `.golangci.yml` which we use on a daily basis. We include it as `golangci.yml` so you must rename it to `.golangci.yml` to make it work. We reward points for linted code.
7. Your solution must be built and be runnable on Linux. If you don't have access to a Linux machine, you can easily set one up using Docker.
8. Use Git for version control. We expect you to send us an archive of your source code when you've done that including **Git metadata (the .git folder)** in the archive so we can look at your commit logs and understand how your solution evolved. Frequent commits are a huge plus. Failed to include git history will result in point deduction. If you have a problem uploading your solution via mail, kindly use Google Drive or Dropbox to share it with us.
9. **Do not make either your solution or this challenge publicly available** by, for example, using GitHub or by posting this problem to a blog or forum. Failed to follow this guide will severely impact your score.
10. Please provide clear instructions using README file.

Challenge; Part 1

OHLC with Volume & Value

In a modern online trading, we know the concept of *OHLC with Volume & Value* to indicate a stock’s performance in real-time, this concept is highly leveraged by investors to guide their investment decisions.

The nature of the data is small but moving ultra-fast, we can’t afford to delay a single second. Worry not, in this challenge, we only provided you a tiny subset of the real data.

A crash-course of terminologies used in this challenge for you to know before we continue:

- **Stock**: a tradable equity in the stock market, for example: BBCA, BBRI, GGRM.

- **OHLC**: stands for *Open Price, Highest Price, Lowest Price, Close Price* in addition to *Previous Price* and *Average Price*.
- **Close Price**: the very last price of today.
- **Previous Price**: the Close Price of the previous day.
- **Open Price**: the first price of today.
- **Highest Price**: the highest price that is ever achieved today.
- **Lowest Price**: the lowest price that is ever achieved today.
- **Transaction**: elements of a Transaction are `Quantity` and `Price`. `Quantity` and `ExecutedQuantity` are the same. `Price` and `ExecutionPrice` are the same.
- **Volume**: the sum of `Quantity`.
- **Value**: the sum of (`Quantity * Price`).
- **Average Price**: is `Value / Volume` rounded. Example: `4780.4` is rounded to `4780` and `4780.5` is rounded to `4781`.

Every Transaction will cause a change in the Stock's *OHLC with Volume & Value*. Transaction that has `Quantity = 0` is the Previous Price of a Stock. Previous Price is not accountable for Open Price, Highest Price, and Lowest Price of a Stock.

Every Transaction with `type` of `E` and `P` are accountable for Volume, Value, and Average Price of a Stock.

For example, consider this subset of Transactions:

Tran. Number	Type	Stock	Quantity	Price
1	A	BBCA	0	8000
2	P	BBCA	100	8050
3	P	BBCA	500	7950
4	A	BBCA	200	8150
5	E	BBCA	300	8100
6	A	BBCA	100	8200

From the above Transactions, a summary of BBCA looks like this:

- Previous Price ⇒ 8000
- Open Price ⇒ 8050
- Highest Price ⇒ 8100

- Lowest Price ⇒ 7950
- Close Price ⇒ 8100
- Volume ⇒ (100+500+300) ⇒ 900
- Value ⇒ ((100*8050)+(500*7950)+(300*8100)) ⇒ 7210000
- Average Price ⇒ Value / Volume ⇒ 7210000 / 900 ⇒ 8011.11 ⇒ 8011

Tasks Breakdown

1. (Total Score = 60%) Given Transactions inside a bunch of `.ndjson` from the `rawdata.zip`, calculate *OHLC with Volume & Value*. Failed to include unit tests in the implementation will result in a deduction of 30% to the Total Score.

   Score Breakdown:

   a. Calculations are correct. (Score = 30%)

   b. The calculation must be scalable to be suitable for the largest set of Transactions, with that in mind you must use Kafka *or* Redpanda as a message broker. (Score = 30%)

   c. Save results into Redis using *the most suitable* data types which available in Redis. (Score = 25%)

   d. Use Git for version control, we want to look at your commit logs and understand how your solution evolved. Frequent commits are a huge plus. (Score = 5%)

2. (Total Score = 30%) As a Client, I must be able to access a Summary of a single Stock that I can arbitrarily choose. Provide a way to get the Summary of a single Stock's *OHLC with Volume & Value.* Failed to include unit tests in the implementation will result in a deduction of 30% to the Total Score.

   The Summary must contain:

   - *Previous Price,*
   - *Open Price,*
   - *Highest Price,*
   - *Lowest Price,*
   - *Close Price,*

- *Average Price,*
- *Volume,*
- *Value.*

Score Breakdown:

a. Calculations are correct. (Score = 40%)

b. Use gRPC & Protobuf as the transport layer. (Score = 30%)

c. Use Redis as the data store using *the most suitable* data types which available in Redis. (Score = 25%)

d. Use Git for version control, we want to look at your commit logs and understand how your solution evolved. Frequent commits are a huge plus. (Score = 5%)

3. (Total Score = 10%) As a future engineer on the team, we need to have a good knowledge on how the existing system works so we can understand it better. Please make a documentation on how your solution for task [1] and [2] work, it can be a Data Flow Diagram, Services Diagram, or both, or any other form of documentation that you think are essential.

Challenge; Part 2

1. This part of the challenge **is not correlated** to the *Part 1*. Therefore you must provide a separated solution for this part.
2. (Total Score = 100%) Given a code with lack of quality, we expect you to adhere to the “The Boy Scout Rule” which **“always leave the campground cleaner than you found it”**. In this task we require you to refactor the Go code inside `main.go` so it becomes more maintainable and having higher quality. We expect you to write the explanation of your solution to this task whether in a code comment or in a separate README file.

   a. Results must be correct. (Score = 50%)

   b. Having good score in golangci-lint. We provide you with the `.golangci.yml` config file. We include it as `golangci.yml` so you must rename it to `.golangci.yml` to make it work. (Score = 50%)

Good luck!