Rust Engineer Technical Challenge

The Challenge

Using Rust, code a simple application that:

1. Connects to two exchanges' websocket feeds at the same time

2. Pulls order books, using these streaming connections, for a given traded pair of currencies (configurable), from each exchange

3. Merges and sorts the order books to create a combined order book

4. From the combined book, publishes the spread, top ten bids, and top ten asks, as a stream, through a gRPC server

Background

Markets

A market is generally a pair of currencies and an exchange where they are traded. For example, **ETH** (Ethereum) and **BTC** (Bitcoin) are a pair that together from a traded "symbol" - **ETHBTC**. This means you can buy or sell **ETH** using **BTC** as the "pricing" currency.

Order books

Orders at which people are prepared to buy and sell are send to an exchange, such as Binance. The exchange will usually match the buy and sell orders that approach a market 'mid-price'. The difference between the best ask price and the best bid price is called the spread.

The final, merged order book should have the best deals first. That means, if I am selling currency and I want to be the first one to sell, I should be at the best position for this which means am selling the largest amount at the lowest price. Think about this when sorting each side of the order book.

gRPC

GRPC (https://grpc.io/) is relatively modern Remote Procedure Call protocol. If you have used anything like GraphQL or Thrift, it should be fairly familiar. If not, it is not difficult to learn! If you are new to RPC you may even find you prefer gRPC's structured and typed protocol over HTTP Methods.

For this challenge you can use this protobuf schema to create your gRPC server:

```
syntax = "proto3";
    package orderbook;
    service OrderbookAggregator {
    rpc BookSummary(Empty) returns (stream Summary);
}

message Empty {}

message Summary {
    double spread = 1;
    repeated Level bids = 2;
    repeated Level asks = 3;
}

message Level {
    string exchange = 1;
    double price = 2;
    double amount = 3;
}
```

Relevance

At Keyrock we calculate and stream prices for markets as part of liquidity services and strategies we provide for clients. By gathering order books we can use the top bids and asks to find a fair mid-price. Once we have a mid-price for a market, we can calculate strategies for opening orders in that market. This journey begins with first streaming the current public market order-books internally to our data aggregation and analytics services. This first step is the background to this challenge.

Further information

The chosen exchanges for the challenge are Binance and Bitstamp.

> ℹ️ **Please note**, exchange documentation can be updated at any time. Ensure to check the links below and let the hiring team know if something doesn't look right.

Binance

WS API Docs	https://github.com/binance/binance-spot-api-docs/blob/master/web-socket-streams.md
Example Feed	https://api.binance.com/api/v3/depth?symbol=ETHBTC
WS Conn. URL	wss://stream.binance.com:9443/ws/ethbtc@depth20@100ms

Bitstamp

WS API Docs	https://www.bitstamp.net/websocket/v2/
Example Feed	https://www.bitstamp.net/api/v2/order_book/ethbtc/
Example WS	https://www.bitstamp.net/s/webapp/examples/order_book_v2.html

Output Example

For the configured symbol (e.g. ETH/BTC) the code should fetch the 10 best/lowest asks and the best/highest bids. The output will be a stream of an orderbook created from merging the exchanges' books.

The output should be standardised to this type of format (with 10 asks and 10 bids):

```
{
    "spread": 2.72,
    "asks": [
        {
            "exchange": "binance",
            "price": 8491.25,
            "amount": 0.008
        },
```

```
    {
        "exchange": "coinbase",
        "price": 8496.37,
        "amount": 0.0303
    },
    ...
],
"bids": [
    {
        "exchange": "binance",
        "price": 8488.53,
        "amount": 0.002
    },
    {
        "exchange": "kraken",
        "price": 8484.71,
        "amount": 1.0959
    },
    ...
]
}
```

Here you can see arrays of `[ price, amount ]` values in JSON representation, but your solution will be streaming from a gRPC server. Values here are made up examples and do not represent what your solution should output.

You should do this from the Web-socket endpoints the exchanges offer (see links below for docs). We want to be able to provide a pair (that will exist in the exchanges) and have your code stream the spread and the top 10 asks and bids for each exchange, on every change of any order book.

This description is intentionally brief so that you have flexibility in how you solve the challenge. You can decide how to present the results, e.g. you may decide to create some kind of CLI wrapper for the tool so it can be used from the command line, or you may decide to make it usable from a URL in some way. This part is not as important as solving the main challenge but feel free to show off - this is your opportunity to show us what you already know, or as a way to express your opinions about how this type of tool/service should be

built. Bear in mind that speed is one of the most important factor for us operationally.

Notes

- **Ask for clarification** if something doesn't make sense in the challenge description, or you are not sure of it what it means.

- **Communicate** as much as you can regarding the challenge and also during the coding of the challenge. Think of it as time spent working with us. You will not be judged negatively if you ask a simple question just to verify your approach, for example. You can ask questions via email or Slack, where we will be happy to reply when we are online (we'll send you an invite).

- Code as much or as little as you feel is necessary to solve the challenge. It is an **opportunity to demonstrate** your coding approach/philosophy/knowledge. Whatever you create we will ask you about it and expect you to be able to talk around your approach.

- The challenge is an opportunity to show **what** you know and **how** you work / communicate during a technical task.

- The codebase should be saved to a git repo and pushed to a service like Github, and you should send us the link.

- If you find you cannot finish the challenge, we would still encourage to send as much of your solution as you have completed.