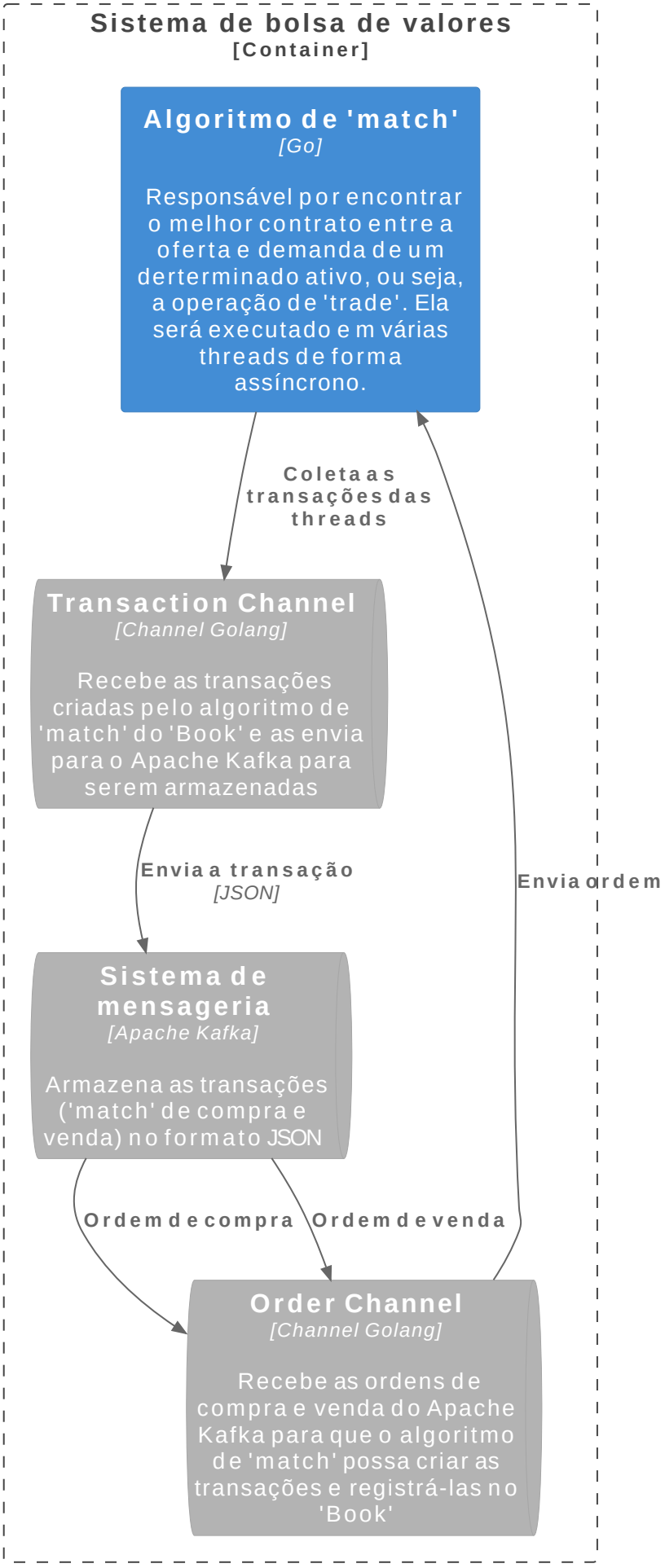
c2-container

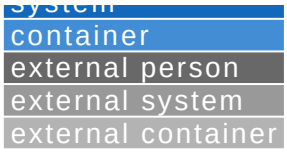
/c2-container



Legend

person

system



No segundo nível podemos verificar, de forma mais detalhada, como a arquitetura do *Code Invest* está interligada aos seus componentes, além das tecnologias que fazem parte deste ecossistema.

Requisitos funcionais

- Simulador da bolsa possui um algoritmo ligeiramente complexo para fazer o match das ordens de compra e venda

Requisitos não funcionais

- As operações devem ser "*in memory*" a fim aumentar a agilidade na execução do mesmo, e para isso as principais alocações precisam ficar na memória *heap*
- Garantia de *memory safe* a partir do recurso de *channels* da linguagem Go

Algoritmo de *match* das ordens de compra e venda

Esse algoritmo consiste em 2 filas (uma de compra e uma de venda) e será feito comparações entre elas para conseguir conciliar a oferta com a demanda da melhor forma possível. Veja a ilustração abaixo:

Cada vez que uma ordem de compra e venda dão "match", é gerado uma transação que será publicada no Apache Kafka no formato JSON.

Memory safe com *channels*

A fim de evitar erros como *race condition* o qual ocorre quando 2 ou + threads tentam alterar um valor ao mesmo tempo.

Para isso, como solução foi adotado o recurso de *channels* da linguagem Go que fornece um "canal" de comunicação entre 2 ou + threads e o dado que está nesse canal será coletado por uma dessas threads conectadas a esse canal.

O papel das *channels* na aplicação é para que todas as ordens de compra e venda sejam enviadas a um único *channel* de input para serem registradas no "livro" (*Book*) de compra/venda.

Além disso, quando ocorre um "match" é criado uma transação que será enviada a um *channel* de output para assim ser armazenado no Apache Kafka