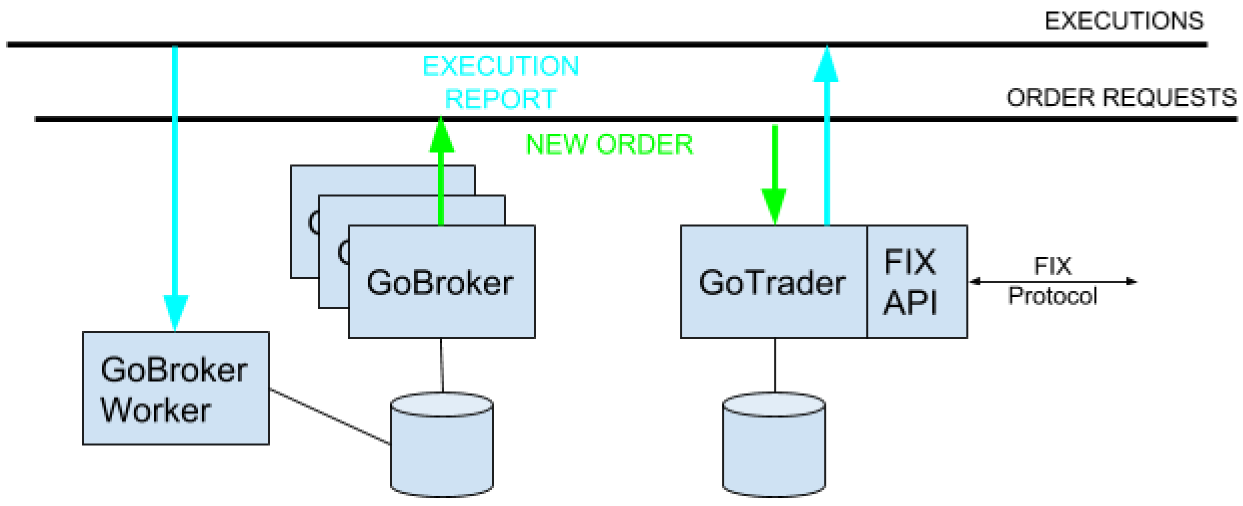
Purpose of this feature: Trade Flow Validation

We need a mechanism to prove that the flow of trade messages is correct when we start up the trading services. Prior to this feature, the servers could be started successfully after passing their individual startup tests, but they would sometimes fail in production due to messaging or process compatibility issues between the servers. With Trade Flow Validation, we can be sure that the group of servers are able to pass trade messages without errors prior to allowing production trading to flow.

System Architecture



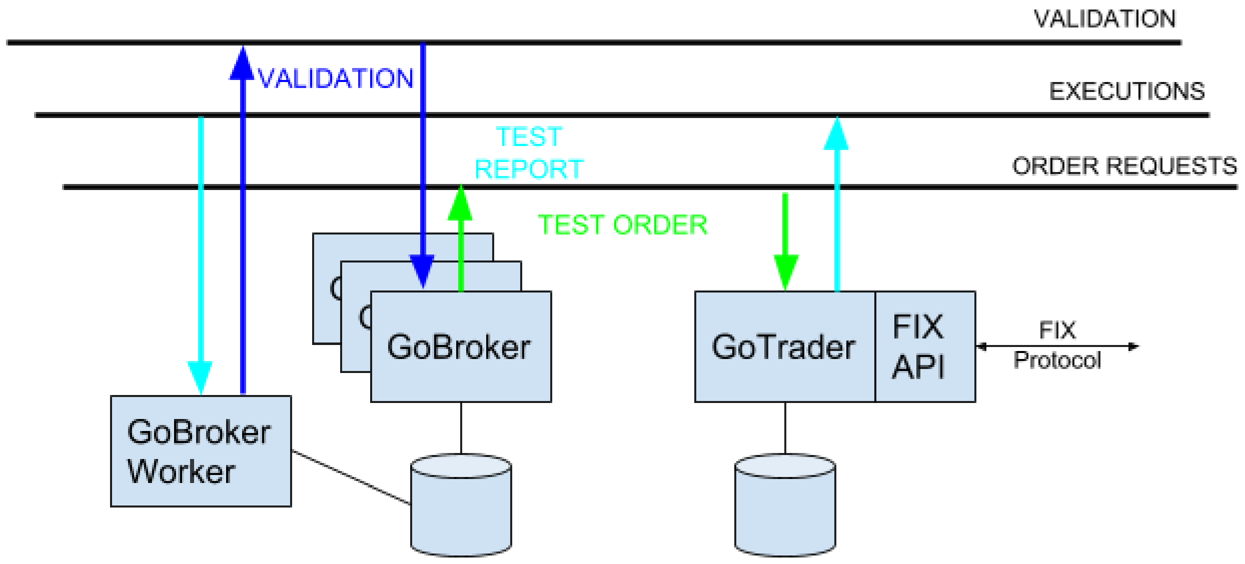
A new order is initiated by one of the GoBroker servers and is placed on the ORDER REQUESTS queue. The single GoTrader server consumes the NEW ORDER message, places the trade using the FIX API, then places an EXECUTION REPORT message on the EXECUTIONS queue. The GoBroker Worker consumes the EXECUTION REPORT and updates the trading database with the order status, etc.

Our task involves verifying that a message sent from GoBroker to GoTrader is processed correctly. Since the flow of trades is initiated and managed by the GoBroker server, we want to drive the validation process from GoBroker - it will be initiated with a message sent by every GoBroker server on startup, and each GoBroker server will receive a validation message that ensures that the flow of trade messages is working correctly for that GoBroker server.

Note that in the existing architecture, there is no bidirectional communication between the GoBroker servers and the GoTrader server. We have to solve the problem of “closing the loop” for a message initiated by a GoBroker server to be acknowledged as having made a full circuit through the trading system.

Validation Approach

We change the existing system by introducing another queue named VALIDATION to the system, which allows for exchange of messages among the servers as needed to close the loop:



Because of the many-to-one relationship between GoBroker and the GoBroker worker, we have a problem to solve WRT having each GoBroker receive a completion message from the GoBroker worker. Using the above approach, each GoBroker will send a test order message when it starts up. With three GoBroker instances starting simultaneously, the GoBroker Worker will send three completion messages on the VALIDATION channel at roughly similar times. Because we are using a message queue that is shared among the three servers, it is possible that one server may consume all three validations, leaving the other two waiting, rather than our intention which is to have all three get a single validation message.

Protocol Design

When a GoBroker initiates a “TEST ORDER” message, it will format a new order structure with blank or null fields and these fields set:

```
Order = {
        ID=”TEST ORDER”,
        ClientOrderID=<GoBrokerID>,
}
```

The ID field will be intercepted before normal order validation to ensure test messages aren’t discarded, and when GoTrader receives the message, it will create an execution message like this:

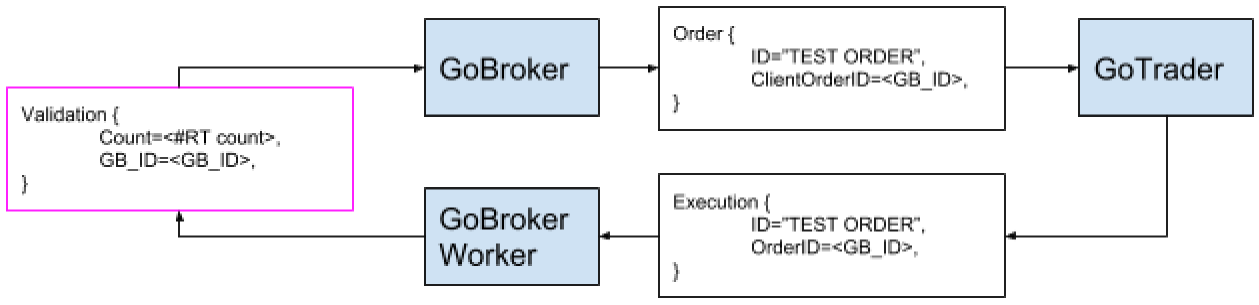
```
Execution = {
        ID="TEST ORDER",
        OrderID=<GoBrokerID>,
}
```

and place it on the EXECUTIONS queue. The intention of this flow is to mimic the normal trade message flow as closely as possible.

To close the loop, we send a Validation message from the GoBroker Worker that handles the Execution message on the VALIDATION queue. The content of a validation message is the count of the number of times the message has been sent (more on this in a bit) and the GB_ID of the originating GoBroker that sent the test order:

```
Validation = {
        Count=<#RT count>,
        GB_ID=<GB_ID>,
}
```

When a GoBroker gets a message on the VALIDATION channel, it checks the GB_ID from the message - if it matches the GB_ID of that GoBroker instance, then the validation of order flow has been completed successfully. If the GB_ID does not match that instance, the Count is incremented and the message is replaced on the VALIDATION queue. The next reader of the message may be the matching GoBroker instance, or it may be the same (wrong) instance that replaced the message on the queue, in which case the counter will indicate that another "round trip" has occurred on the VALIDATION queue without successful receipt of the validation by the other instance. Each round trip involves the receipt of the non-matching message by the instance, incrementing the message counter and replacement on the queue. If enough round trips have occurred without success, the receiving instance can act in a number of ways - but in the first implementation, it will stop re-queueing the message and make an Info level log message that the validation for <GB_ID> has failed.


GoBroker
Order { ID="TEST ORDER", ClientOrderID=<GB_ID>, }
GoTrader
Validation { Count=<#RT count>, GB_ID=<GB_ID>, }
GoBroker Worker
Execution { ID="TEST ORDER", OrderID=<GB_ID>, }

Once validation has succeeded for the initiating GoBroker instance, the process has completed and production trade flows can be initiated.