Chronicle SOFTWARE

TECHNICAL REPORT

Building an Exchange with Chronicle Matching Engine

Table of Contents

Introduction

This document discusses the design of a self-contained exchange. The exchange demonstrates how to use the Chronicle Matching Engine. The exchange provides a packaged, modular application framework that supports:

- Distributed gateways for client connectivity
- Order validation and risk checks
- Order matching
- Multiple order types, including worked orders such as Stops, Icebergs, Pegged Orders, Hidden
- Multiple time-in-force types, including IOC, FOK, GTT, GTD
- Configurable matching controls (eg self-match prevention, or controls across specific brokers)
- Market Data feeds
- Reference Data feeds
- Aggregating services such as Risk Controls, Drop Copies, Consolidated Books
- Scalable, distributed architecture
- HA/DR

Component Hierarchy

This section describes the hierarchical organisation of the key components of the exchange, illustrating how each layer builds successively on previous layers to provide scalability, functionality, and resilience.

Order Book

The core of the exchange is the per-symbol Order Book. Each Order Book maintains separate lists of Buy and Sell Orders ordered by price (high to low for Buys, low to high for Sells). Orders with the same Side and Price are grouped into the same level of the book, and prioritised within that level. Chronicle Matching Engine contains sophisticated logic to determine and evolve order priority.

Each symbol has a specific tick size which determines the minimum price change, which in turn dictates the separation and grouping of levels in the order book. Tick size is dynamic and under control of Chronicle ME logic.

Orders within an Order Book are matched according to standard CLOB rules.

An example Order Book for one symbol would be as follows. For each level the quantity may be formed by one or more orders at the same price (not shown).

Symbol: ABC		
Quantity (Total)	Price	Quantity (Total)
	...	...
	103	2,500
	102.5	3,000
	101	1,000
5,000	100.5	
3,000	99	
2,000	98.5	
7,250	98	
...	...	

Order Entry

The behaviour of an order which is added to the system is principally dictated by its Type and Time-In-Force properties.

> Note, the following table lists the most typical order types which the exchange supports, or are under development.

Main Order Type options are listed in the following table. Note the priority rules discussed further below:

Simple/Worked	Type	Comments
Simple Order Types	Market	Executes at best until matching stops. Any remainder is expired
	Limit	Executes at or better than the price specified on the order. Any remainder is added to the order book or expired based on the specified Time-In-Force
Worked Order Types	Stop	A Market order which remains inactive until the market reaches a specified stop price, at which point it is injected as a normal Market order
	Stop Limit	A Limit order which remains inactive until the market reaches a specified stop price, at which point it is injected as a normal Limit order
	Iceberg	An order which has both a Quantity and a (smaller) Disclosed Quantity. Only the Disclosed property is added to/displayed on the order book. Once the displayed quantity reduces to 0 the order is replenished by the system up to the minimum of the Disclosed and Remaining Quantity
	Hidden	A fully hidden order which is active on the book, but with no displayed quantity
	Pegged	A hidden order pegged to some mid point (e.g. best bid, offer)

Note that the Order Types fall into two main categories: “Simple” orders where the lifecycle is entirely controlled by the client (Market and Limit), and “Worked” orders where - in addition to client input - the exchange actively works the order depending on various other parameters (Stop, Stop Limit, Iceberg, Hidden, Pegged). For the remainder of this document an “Order” can be a member of either group; the terms “Simple” and “Worked” will be used whenever a differentiation between the groups is required.

The main Time-In-Force options are:

Type	Comments
Day	Expires at end of current trading day
IOC (Immediate or Cancel)	Executed on receipt. Any remainder is immediately expired
FOK (Fill or Kill)	Fully executed on receipt, or immediately expired
GTT (Good Till Time)	Expires at a specified time (UTC, second resolution)
GTD (Good Till Date)	Expires at the end of specified trading day (local calendar)

Amendments, and Priority Rules

Once an order has been entered into the system, the following properties can be amended:

- Quantity
- Disclosed Quantity
- Price
- Stop Price
- Expiration Time (GTTs)
- Expiration Date (GTDs)

However, there are some restrictions around allowed Amends. The details of these are omitted from this document.

Matching and Priority Rules

For any given price level, the priority order (from high to low) when matching is:

- All displayed orders
- Non-displayed parts of Icebergs (via replenishment)
- Fully Hidden Orders
- Pegged Orders

Within each grouping, the priority is then determined by the time the order was added to the book, or when amended (if the Amend caused the order to lose priority).

Auxiliary Order Book State

In addition to the book itself, each Order Book component maintains the following state used within the matching process:

- Off-book components of Worked Orders, e.g. for replenishing Icebergs, injecting Stop orders
- Risk controls for the book's Symbol, including awareness of aggregated content
- Reference Data, eg tick size rules, suspension status
- Matching controls, eg self-match prevention, broker-broker matching rules

Specific details of the above are discussed later. For now it is sufficient to note when continuing to build the hierarchy in this section, that an Order Book component should be understood as including some auxiliary state beyond the per-symbol book itself.

Extended Matching Controls

Additional per-order matching rules can optionally be applied on top of the core matching and priority rules described above e.g. Minimum Execution Quantity & Self Match Prevention. The details of these are omitted from this document

Processing Pipeline

A processing pipeline coordinates the handling of events across one or more Order Books. Note, the pipeline does not itself control the raising of events (e.g. user requests, Stop injections), but is responsible for coordinating the events, and implementing the steps which are applied to any one event.

The pipeline is responsible for assigning unique identities to identify Orders and Executions. These identities are guaranteed to be unique across different trading days (for long-lived orders), and are pseudo-randomised to avoid yielding any information on activity which could potentially be leveraged by users. The Order Identifier is constant throughout the lifecycle of any one order.

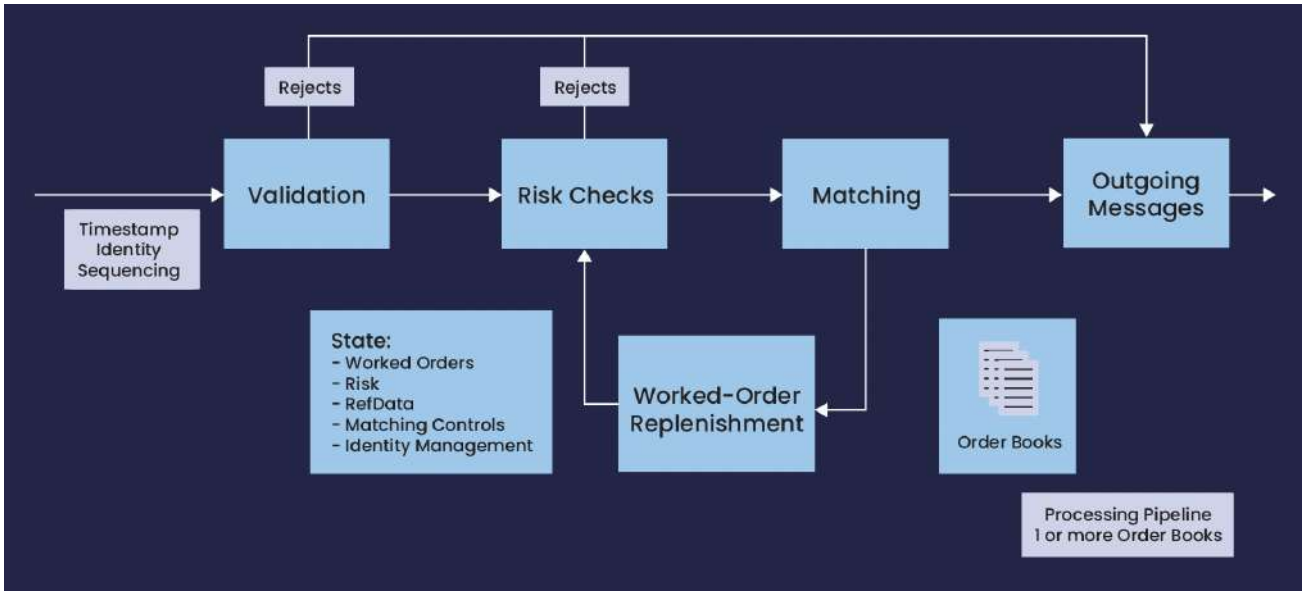

Figure 1. Main processing pipeline of the exchange

Transactions, Status, and Execution Types

The main transactions on an Order handled by a processing pipeline are:

- New
- Amend
- Cancel
- Mass Cancel
- Executions

These can be raised either by user requests or automatically from within Chronicle Matching Engine (see next section).

The status of an order at any one time is one of the following (from high to low precedence):

- Filled
- Cancelled
- Expired
- Partially Filled
- New
- Rejected
- Suspended (not currently on the order book - eg parked Stop/Stop Limit)

Chronicle Matching Engine

A Chronicle Matching Engine instance wraps a single processing pipeline (which may control 1 or more Order Books), and controls the interfaces between the pipeline and the wider exchange environment. A core piece of the instance is a single event loop which controls and coordinates all activity across the instance.

The architecture of the Matching Engine provides for non-functional requirements such as:

- Low, predictable latency with high throughput
- HA/DR with minimal downtime
- Live upgrades
- Off-line querying and analytics
- Determinism so that behaviour can be reproduced offline

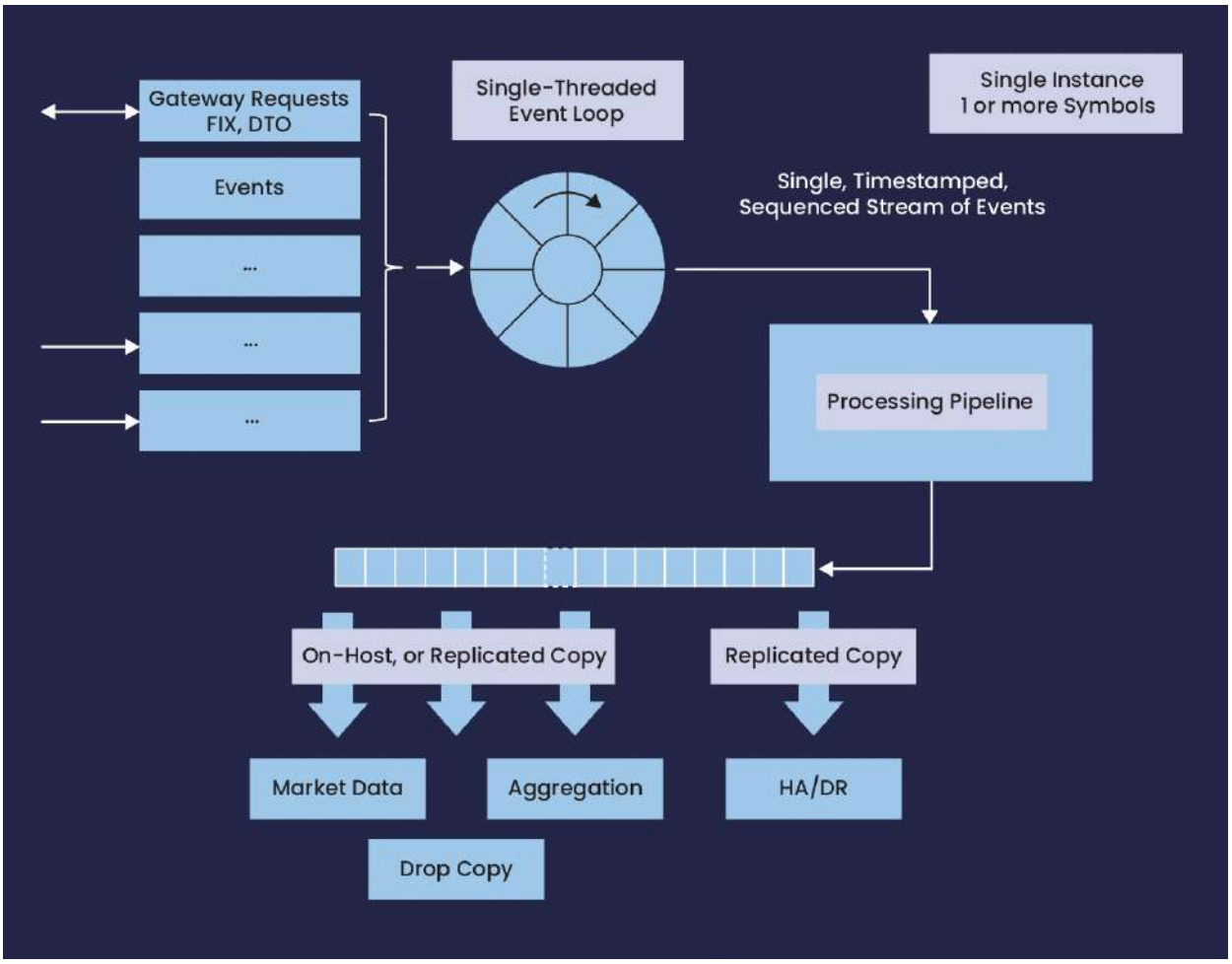

Figure 2. A Chronicle Matching Engine instance processing events

The use of a queue to decouple the instance and other feeds across the wider exchange environment enables great flexibility on how components are distributed in order to best use available resources, and also to segregate services based on latency profiles (e.g. drop copies are likely less latency sensitive than market data and can be handled off the critical path).

Distributed Matching Engines Instances, & Aggregating Services

Matching Engine instances can be freely scaled to meet design load: the only requirement is that all transactions for any one symbol are handled by a single instance. The Transaction Queue feeds from multiple instances can be used to feed other services, and these services can either themselves be distributed, or aggregated into a single instance. Where aggregation is required, Transaction Queues from multiple instances are replicated into the aggregating service, then polled for updates.

Risk Controls

Risk checks can be handled either sharded or aggregated. This is illustrated in the below diagram where three transaction engines replicate their transaction queue to the Aggregating Risk service host. The Risk service then uses those queues to maintain a global risk state which is persisted by a series of atomic updates to an Aggregated Risk Queue (directly analogous to how Transaction Queues are built by the Chronicle Matching Engine instances). This Aggregated Risk Queue is then replicated to each of the instance's hosts, where it is polled by the instance event loop for updates.

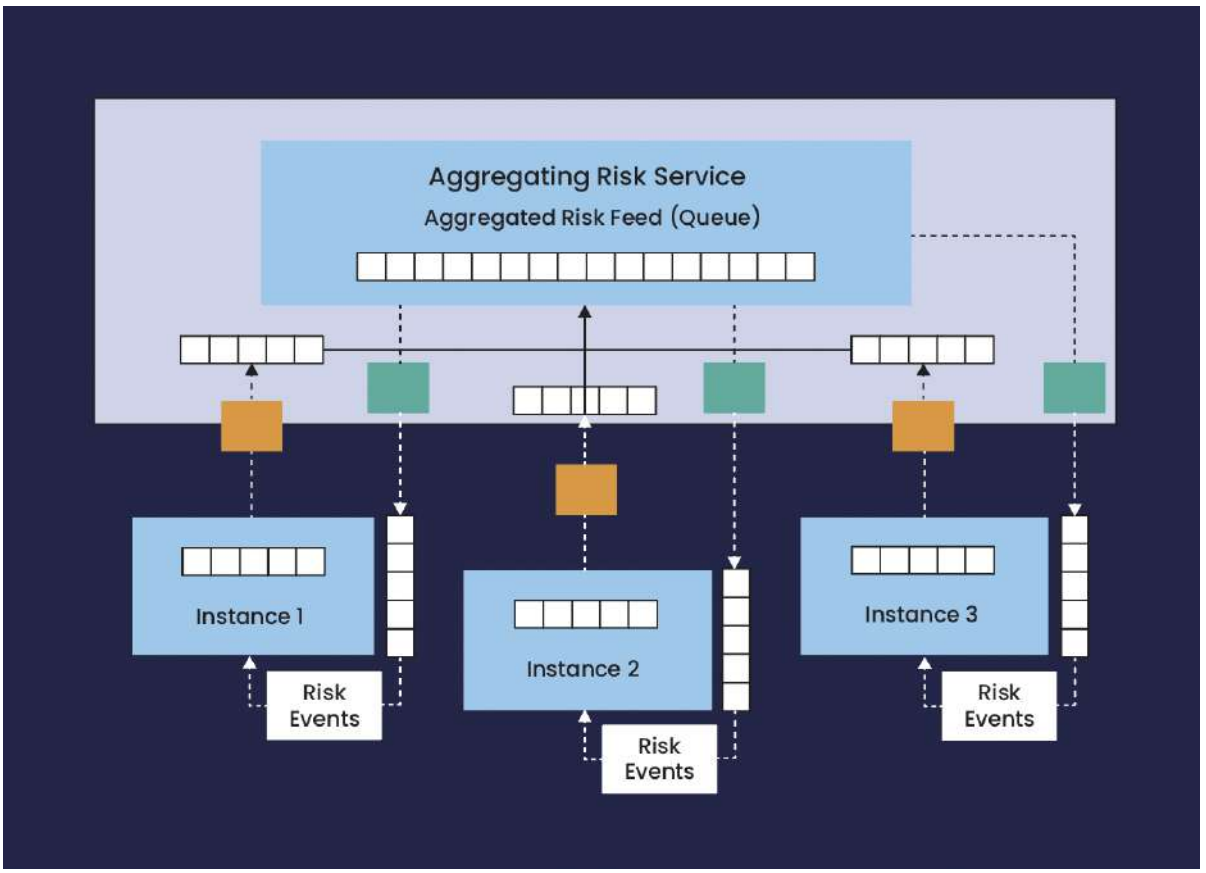

Figure 3. Three transaction engines replicating their transaction queue to the Aggregating Risk service host

A number of risk control types are supported - details have been omitted from this document.

Drop-Copy Service

A Drop-Copy feed can be built in a directly similar fashion to an Aggregating Risk service, although in this instance there is no need to replicate back to the Chronicle Matching Engine instances: any output from the Drop-Copy Service is sent directly to the connected clients.

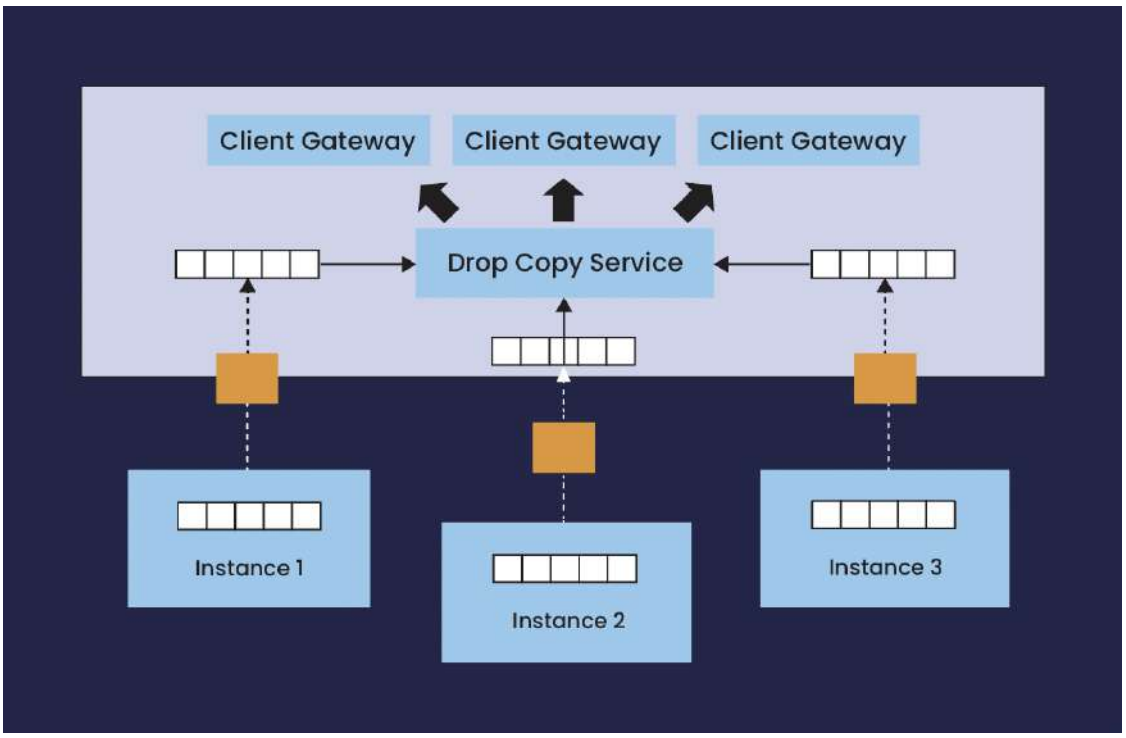

Figure 4. Generating Drop-Copy reports of all relevant execution reports

Consolidated Book

A Consolidated Book Service providing a single view across all Books (for example for driving front-end GUIs) can be assembled following exactly the same approach as a Drop Copy service. In this case the Consolidated Book service monitors (replicated copies of) the Chronicle Matching Engine instances' queues and assembles a real-time summary picture across all books, which can be used to drive queries and live updates to connected clients. Again optionally this service can share the same input queues as Risk and/or Drop Copy.

Market Data Service

A Market Data service can be constructed in a similar fashion to the above as an aggregating service using replicated copies of each Matching Engine's output. Alternatively, each Matching Engine host can publish market data individually given all information is available from the local Transaction Queue for the subset of symbols traded on that instance. Market Data updates would normally be published via UDP.
A separate TCP/IP interface is provided to enable recovery of messages following a gap detection on the UDP stream. This mechanism will provide the ability to recover messages only up to a limited age (e.g. last N messages, or last hour, or trading day), after which the message should be considered lost.

Client Gateways

A distributed cluster of client gateways forms the client-facing surface of the exchange. client gateways can be distributed arbitrarily to match demand and resources, with one or more gateways on each of one or more hosts.

A client is assigned a specific number of sessions and can connect any session to any gateway. Each session can only be connected to one gateway at any time. All symbols can be reached from any one session, and each session is logically independent from any other. Sessions can be configured as part of a group, enabling one session to enter orders on-behalf-of another session if needed.

Optional throttling controls can be set up on a per-session basis to limit the number of messages sent per second (sliding window). Messages exceeding the limit are rejected, and continued breaches will result in the session being dropped.

A connected client may optionally enable Cancel-On-Disconnect for the session, to ensure all open orders submitted through that session are cancelled in the event of a disconnect or logout. Client gateways support FIX (using Chronicle FIX).
All state for a client gateway is incrementally written to a Queue, and can be replicated across one or more hosts for HA/DR using Chronicle Queue Enterprise.

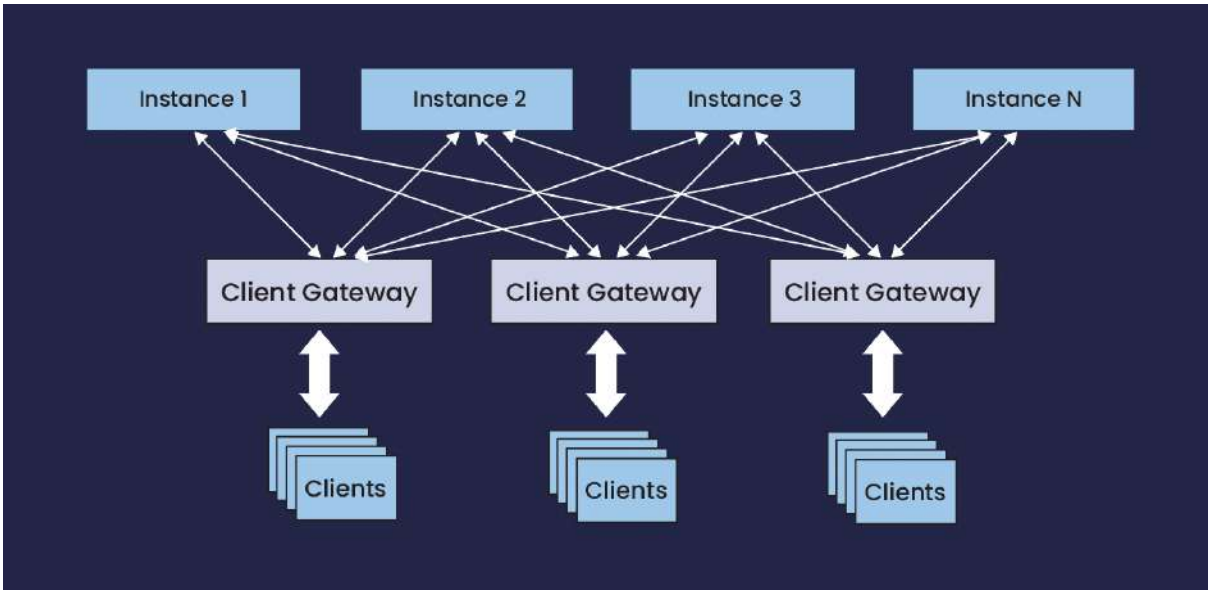

Figure 5. Clients connecting sessions to client gateways

Distribution, Scalability, and HA/DR

Throughout the preceding discussion emphasis has been placed on the modular and hierarchical nature of the design:

- The core per-symbol Order Book
- Processing Pipeline with 1 or more Order Books per Matching Engine
- 1 or more Matching Engines, with Aggregating services providing global views
- All coupling between hosts based on replicated Queues
- Distributed cluster of client gateways

This approach provides great flexibility in how components are distributed over available hosts, and can be freely scaled as needed to match growth in demand.

All state required by an individual component is contained in the component's Queue, and the Queue can be replicated in real time to any number of hosts, which in turn provides considerable flexibility for HA/DR options. Replication can optionally be configured to require acknowledgements from some or all of the secondary instances which minimises the risk of message loss at the cost of higher latencies. This aspect is again freely configurable, so parts of the system which demand the highest latencies but can tolerate some potential message loss can be run without acknowledgement, whereas other parts of the system which demand no message loss but which are less latency sensitive can run with acknowledgement. Chronicle Queue Enterprise allows acknowledgement strategies to be customised based on message content e.g. ensure that a large order is replicated (and acknowledged) by n hosts.

In this model also, only a single component is responsible for appending data to any one Queue. Once a Queue is available on a host, any number of services can read data from the Queue completely independently. This in turn allows - for example - multiple aggregation services on one host to be driven off a single replicated copy of the set of queues from the Chronicle Matching Engine instances. Going the other way, multiple services can be run on the same host as the source Queue without the need for replication: all that is required is access to the Queue data. The number of Queue readers does not impact write performance.

The diagram below illustrates the main components of a typical full exchange environment. Components within the shaded grey boxes can be arranged arbitrarily across any number of hosts to match available resources. The entire environment is then replicated to a secondary DR site. The DR site will require the same components as the

primary in one-to-one correspondence, but these components may be arranged across available resources independently of the primary arrangement.

Clients can connect to any available gateway. Gateways in the secondary site will not accept any connections until the site becomes primary.

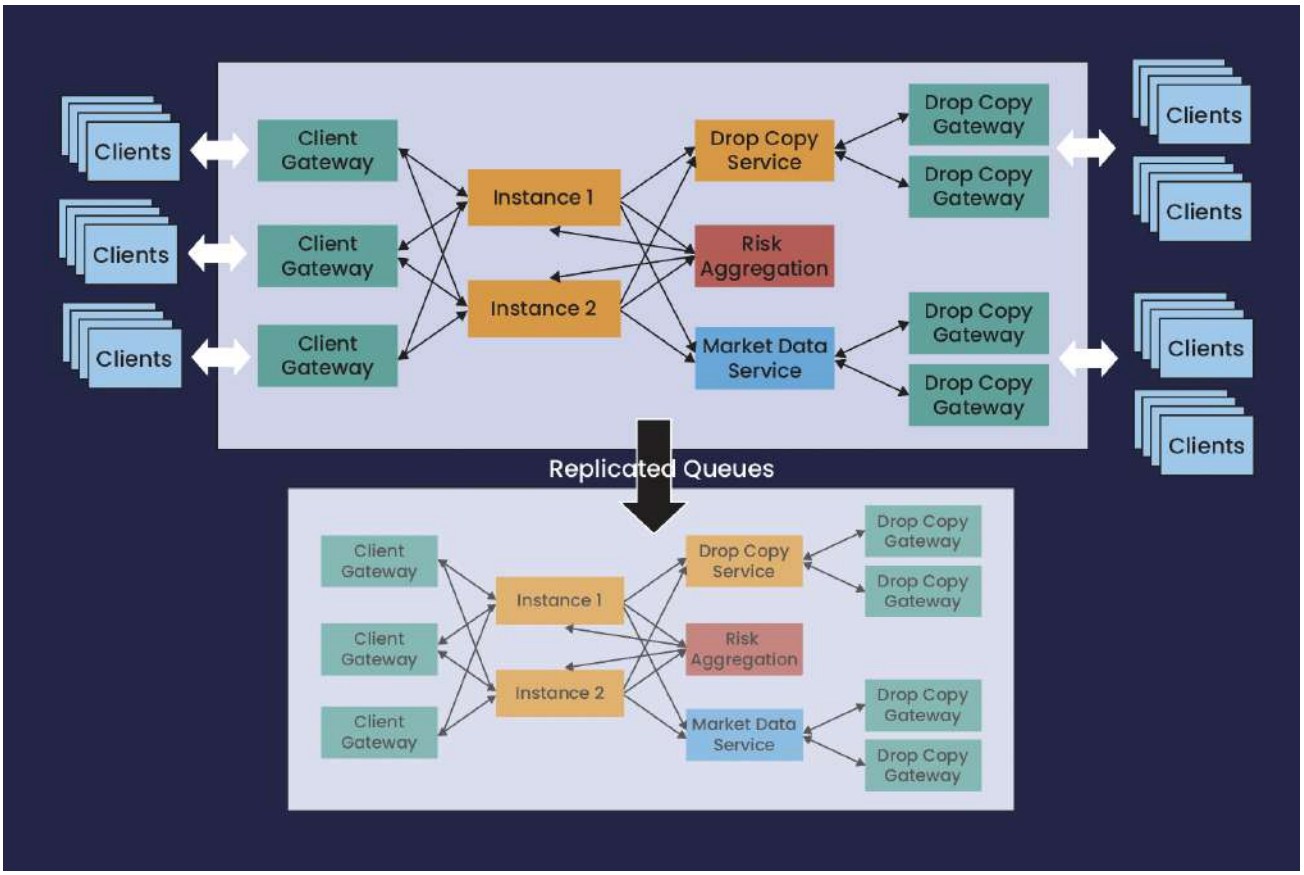

Figure 6. Main components of a typical full exchange environment

Minimal Environment

A minimal exchange environment demonstrating core matching functionality with all symbols supported in one Matching Engine (but neither Drop Copy nor Market Data services) can leverage the flexibility of the design to collapse the client gateway and Risk Aggregation services into the same environment/host as the Chronicle Matching Engine instance, providing a fully self-contained package. The components remain coupled via Queues, and replication to a secondary instance remains as above.

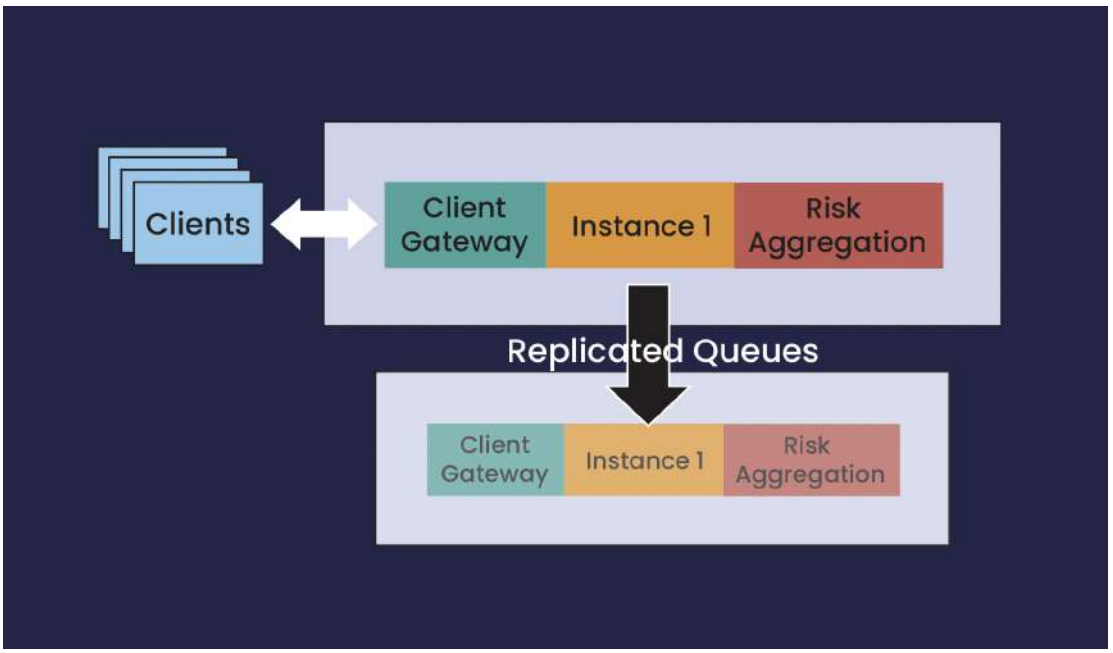

Figure 7. Minimal exchange environment with all symbols supported in one instance of a Matching Engine

Glossary of Terms

Term	Definition
Client	Any computer hardware or software device that requests access to a service provided by a server
Client Gateway	A software component that directs requests from a client to the appropriate service or service instance, possibly in addition performing authentication.
Drop-Copy service	A service which receives copies of trading activity via a separate channel from main order entry. Designed to facilitate real-time monitoring of trading activity (eg back-office, regulatory feeds) off the latency-critical path
ExecutionReport	A report on the status of the execution of Order(s) and Transaction(s)
Order	An instruction to buy or sell on a trading venue
Order Book	A price-ordered list of buy and sell Orders for instruments on an exchange
Risk Aggregation	The real-time collection of multiple metrics across client trading activities in order to maintain an overall risk status and corresponding limits for each client
Time-In-Force	Indicates how long an Order will remain active before it is executed or expires
Symbol	An abbreviation used to uniquely identify an instrument that is being traded on an exchange