Enabling Capital Markets Developers to #BUILD FASTER

www.veloxfintech.com
info@veloxfintech.com
@veloxbuildfast
Linkedin/veloxbuildfaster

A NEW APPROACH TO APPLICATION DEVELOPMENT

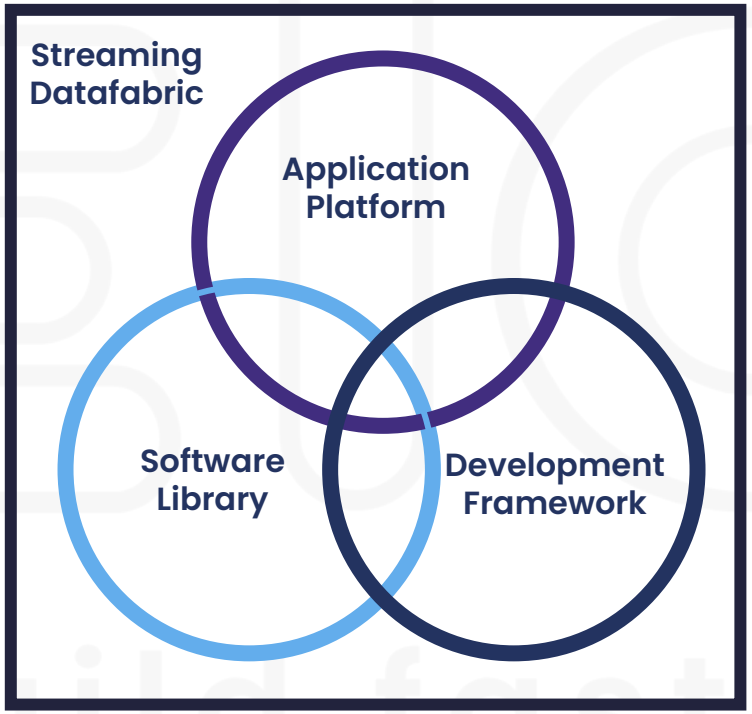
The architecture is based on our experience of what gives maximum development acceleration and its not about conforming to historical categories. We have ripped up the playbook and evolved aspects of different approaches based on actual experience of what works over the lifecycle of a system. Depending on the problem at hand, Velox behaves like :

An Application Platform

providing services that applications rely on for all standard operations, not just development, but testing, deployment, data management, runtime management and more.

A Streaming Datafabric

that can aggregate and normalize real-time and static data and system APIs, eliminating the need to know where the data is mastered.



A Development Framework

for accelerating the build of real-time full-stack business applications

A Software Library

providing a collection of implementations (abstractions), accessed through well-defined interfaces that can be extended and specialized.

A FULL STACK SOLUTION TO A FULL STACK PROBLEM

		Data Layer			Application Layer				Platform Layer	
		Datastream Transformation & Management	Realtime In-Memory Data Cache	Transaction Engine & State Management	Rules, Workflow , Notification Engine	Web/Javascript Framework	UI & Data Virtualization	Data Visualization / Reports	Full-stack Application Platform	User Configuration / Entitlements
	vCore	✓	✓	✓	✓	✓	✓	✓	✓	✓
OTHER TECHNOLOGY VENDOR PRODUCTS	AMPS / SqlStream	✓	✓							
	Kafka		✓							
	Spark		✓							
	Hazelcast / Redis	✓	✓							
	React.JS					✓				
	Angular.JS					✓				
	Spring Boot									
	Tomcat						✓			
	Camunda				✓		✓			
	PowerBI							✓		
	Tableau							✓		
	Wildfly (JBoss)								✓	

vCore brings a complete solution with only the features you need, limiting complexity and learning time and avoiding the brain damage associated with getting different vendor tech to work together

INTRODUCING THE *VCORE* APPLICATION DEVELOPMENT PLATFORM

Application Layer
Specialized full-stack adaptions of "low/no code" accelerators

Low/No-Code tools do not work for high performance mission critical software. We have adapted some of the underlying techniques, such as model-driven development (MDD) and code generation to work in the high-performance space

Platform Layer
A custom web stack that can handle high performance loads

A proprietary presentation layer that is combined with the MDD and code-gen accelerators to significantly reduce the amount of hand-crafted code required to stand-up responsive, professional looking UI's.

Data Layer
A proprietary streaming database and transactional engine

For clarity we re-use SQL & datastream idioms but have little in common with an RDMS. The runtime API is code-generated and our "data-in-motion" paradigm, gives the developer up-to-date transactional and streaming data, when and where needed

Custom Messaging Protocol

Kafka Streaming Event Bus

Data Layer

build faster

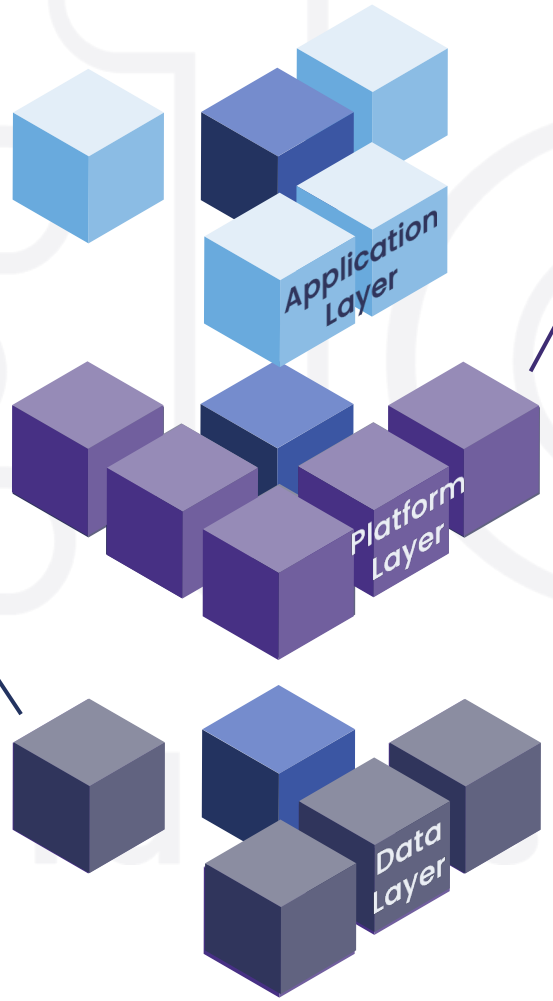
vCore is a full-stack solution to full-stack problems

Application Layer
Specialized full-stack adaptions of "low/no code" accelerators

- High performance, highly customizable grid (blotter) UI component
- User workspace builder & management capability
- User configuration management

Data Layer
A proprietary streaming database and transactional engine

- Streaming data operators
- Transactional state engine
- In-Memory data grid
- Data virtualization



Platform Layer
A custom web stack that can handle high performance loads

- Highly scalable and reliable app & service container
- HTML5/JS Framework w/vue.js bindings for React.JS
- MVVM implementation & session manager
- Operational support tools
- Security, entitlements, and configuration management

vCore design choices that deliver maximum acceleration with minimum limitation

Adaptive throttling

Only apply throttling when necessary and exactly as much throttling as the Consumer needs.

Ultra-thin client

Strictly no business logic allowed. The UI is purely a read/write projection of the data views in the app server.

Non-functional code-generation

Proprietary model-driven development paradigm that generates code based on the data dictionary.

Low-level micro-building blocks

Low level abstractions deliver the sweet-spot between time-to-market and adaptability.

Reactive datastreams

Makes it easy for developers to deal with real-time propagation of change through their applications.

Data virtualization

Ensures very predictable resource utilization during high volume periods with no tuning required.

Decoupled data model

Schema driven architecture allows fast adaptation to different problem spaces and domains.

Data centric

A ubiquitous datacache, means the developer can assume that data is where it's needed, when it's needed.

VCORE BUILDING BLOCKS : BASIC ABSTRACTIONS & STREAMING OPERATORS.

Application Layer

Layout
Coarse-grain layout instructions in HTML5 and CSS..

Throttle
On-the-fly calc of virtual tables and columns for analytics and other meta data.

Injector
Split the computational load of a join or aggregation across many cpu's/jvm's.

Action
Act on a resulting data set using bound source system API.

Aggregations
Reusable and configurable engine allows workflows to be built on fast data streams.

Decorator
On-the-fly calc of virtual tables and columns for analytics and meta data..

Main
On-the-fly calc of virtual tables and columns for analytics and meta data..

Data Layer

Screen
A combination of tables, values and actions rendered within a browser SPA.

Joins
Streamlined "JOIN" logic, abstracts away the complexity of synchronizing sources.

PubSub streaming tables
Logic can be written without explicit synchronization with the subscribers.

View
A rationalized representation that stands up information efficiently for readers.

Formula
Logic can be written without explicit synchronization with the subscribers.

Filter
A rationalized representation that stands up information efficiently for readers.

Platform Layer

User Config Management
Record and replay user actions to detect faults.

Support Manager
Runtime data table inspector.

Entitlements
Programmable at the row, column and action level

Virtualizer
Predictable UI resource utilization on high volume days or on slow networks.

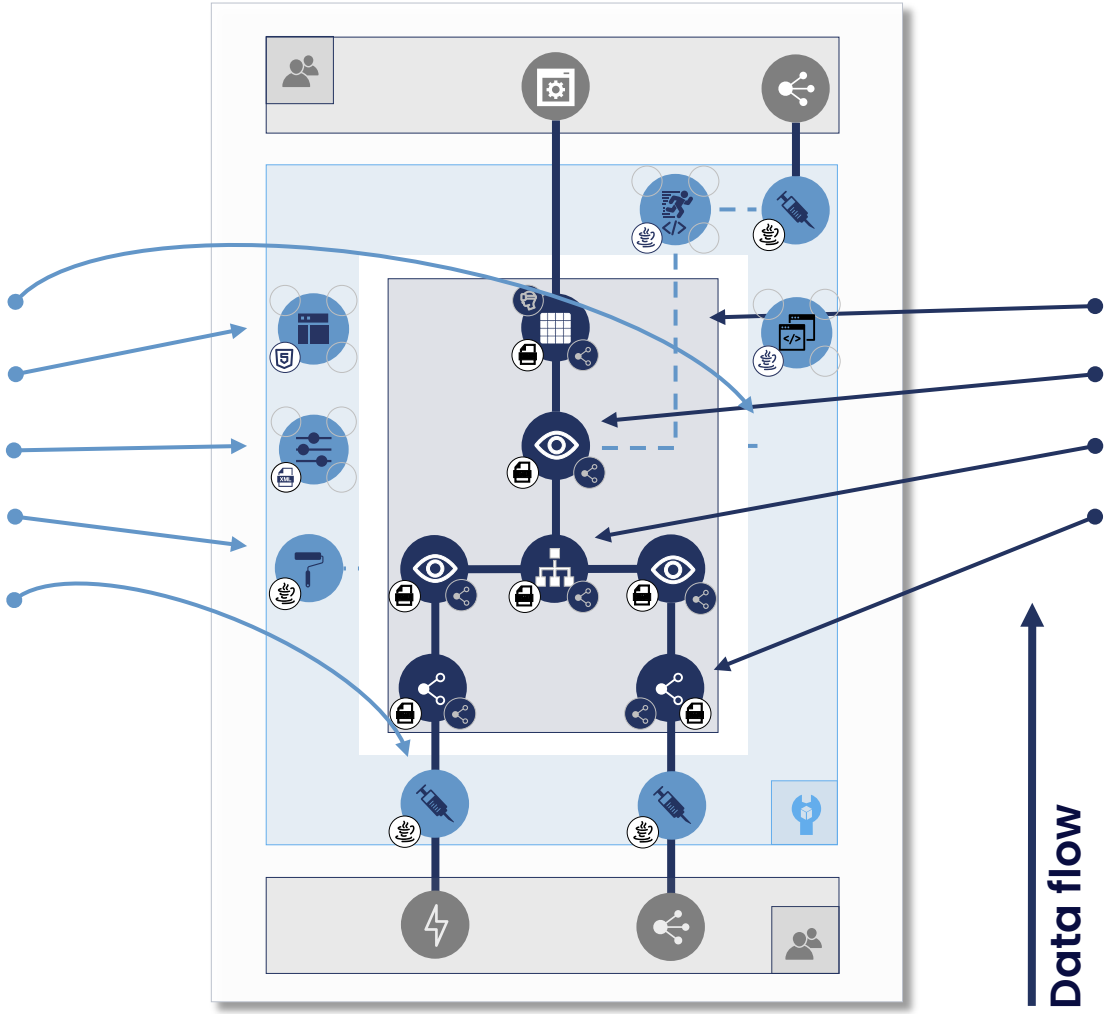
Visualizer
Generated HTML/JS visual components and workspaces

build faster

BUILDING APPLICATIONS : CONSTRUCTING THE GRAPH

Application Layer
Specialized full-stack adaptions of "low/no code" accelerators

- Aggregators
- Layouts
- User Configs
- Decorators
- Data Adaptors
- App Container

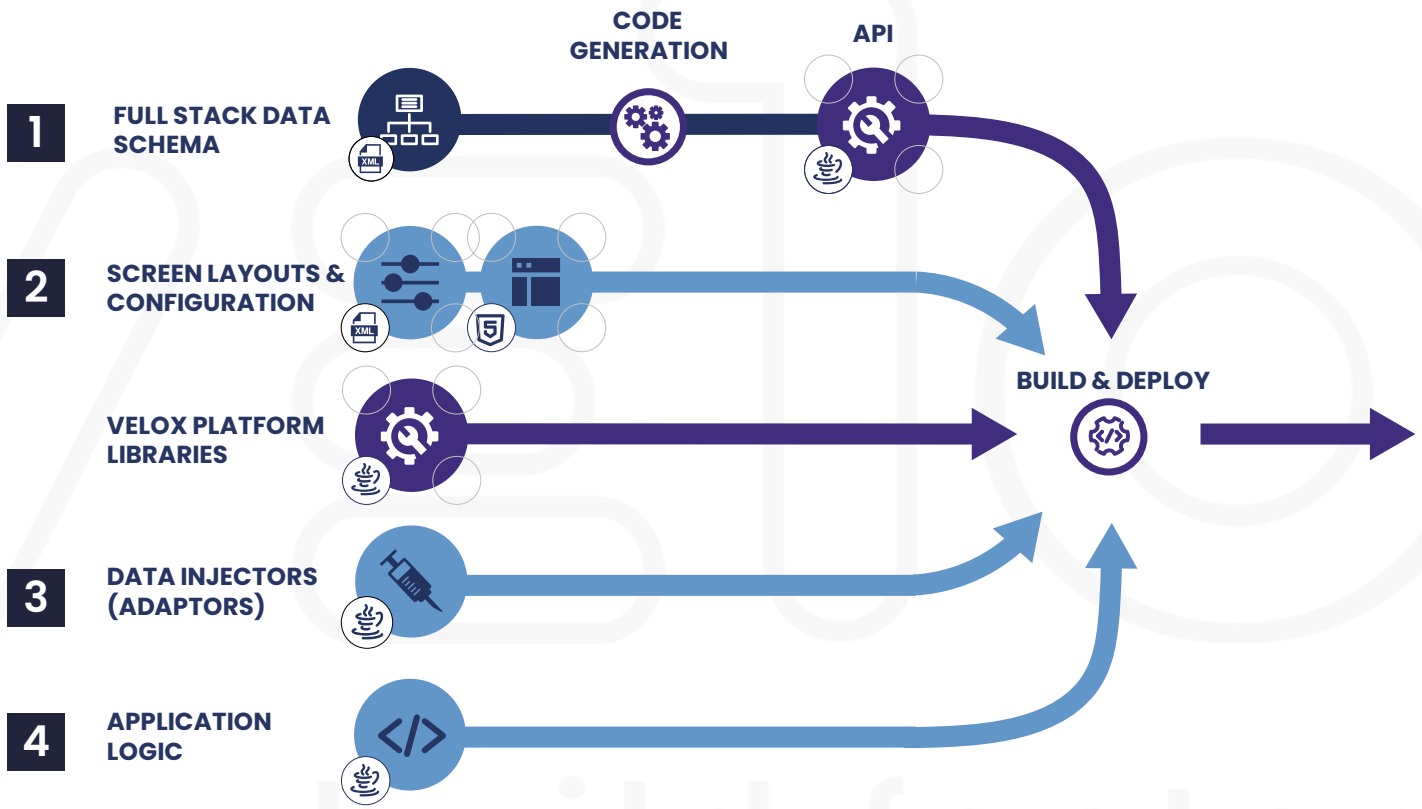


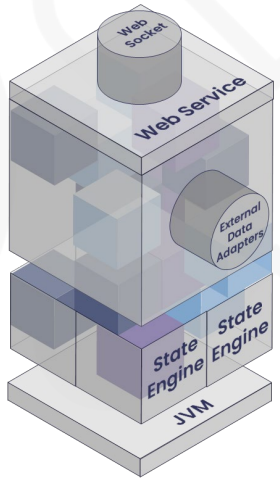
Data Layer
A proprietary streaming database and transactional engine

- Screen Tables
- View Tables
- Join Tables
- Tables

Data flow

THE *VCORE* BUILD PROCESS





XML	**Full Stack Data Schema**	Define the data entities (tables) and their relationships, from data sources to output screen.
HTML/ XML	**Screen Layout & Config**	Coarse-grain layout instructions in HTML5 and CSS. Workspaces, grids.

Java	**Data Injectors (Adaptors)**	Maps data elements from an external API to Velox table elements; apply pre-processing.
Java	**Application Logic**	Business logic, analytics, aggregations, screenhandlers, commandHandlers etc.

PHYSICAL ARCHITECTURE

Application Server**

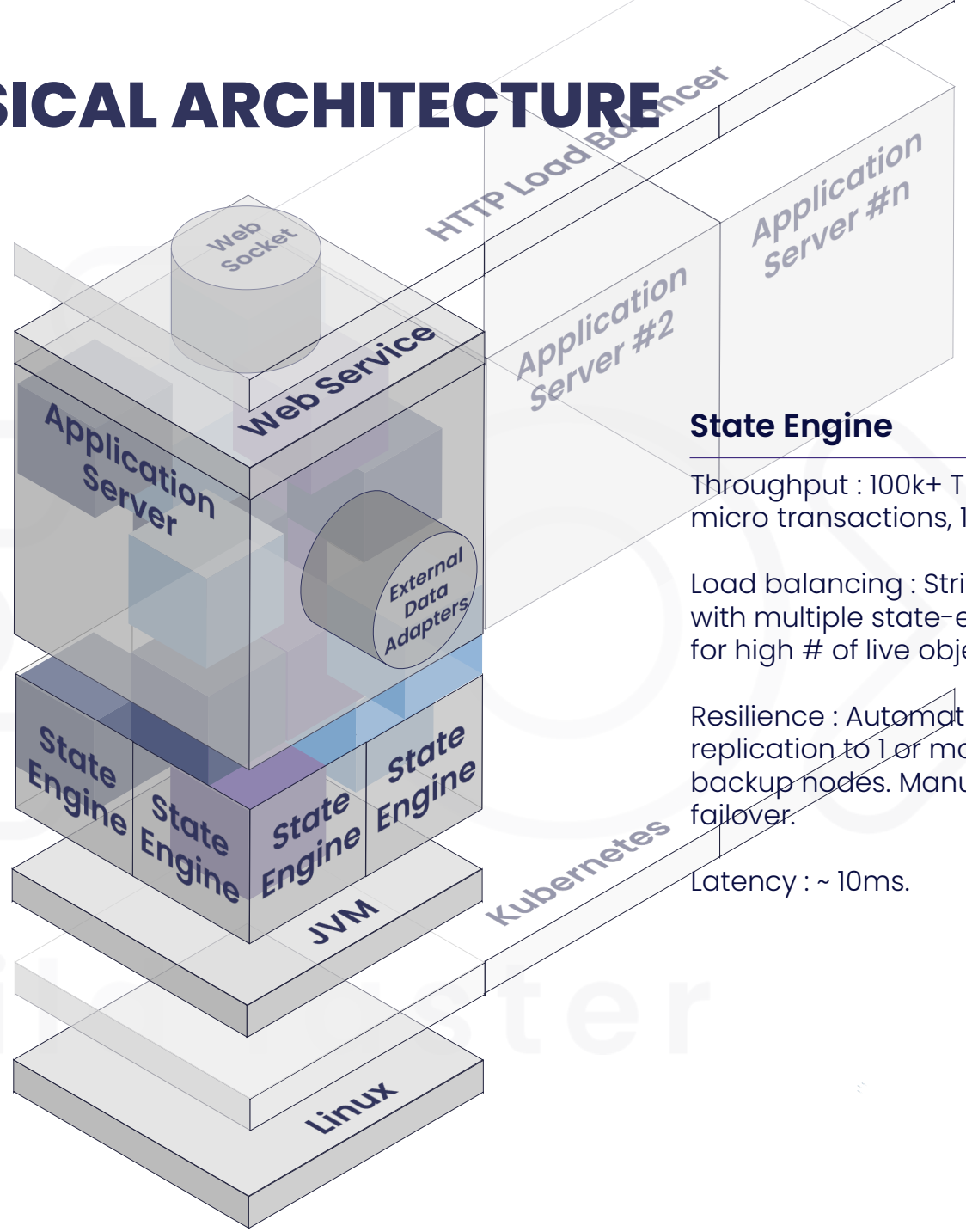
Throughput : ~100 MM rows (with ~100 fields) with 10k updates per sec.

100K concurrent browser sessions, 100 visual components per session

Load balancing : Multiple instances behind an HTTP load balancer.

Latency : ~ 1ms.

** Before tuning



State Engine

Throughput : 100k+ TPS for micro transactions, 10k+ TPS .

Load balancing : Striping with multiple state-engines for high # of live objects.

Resilience : Automatic replication to 1 or more backup nodes. Manual failover.

Latency : ~ 10ms.

build faster

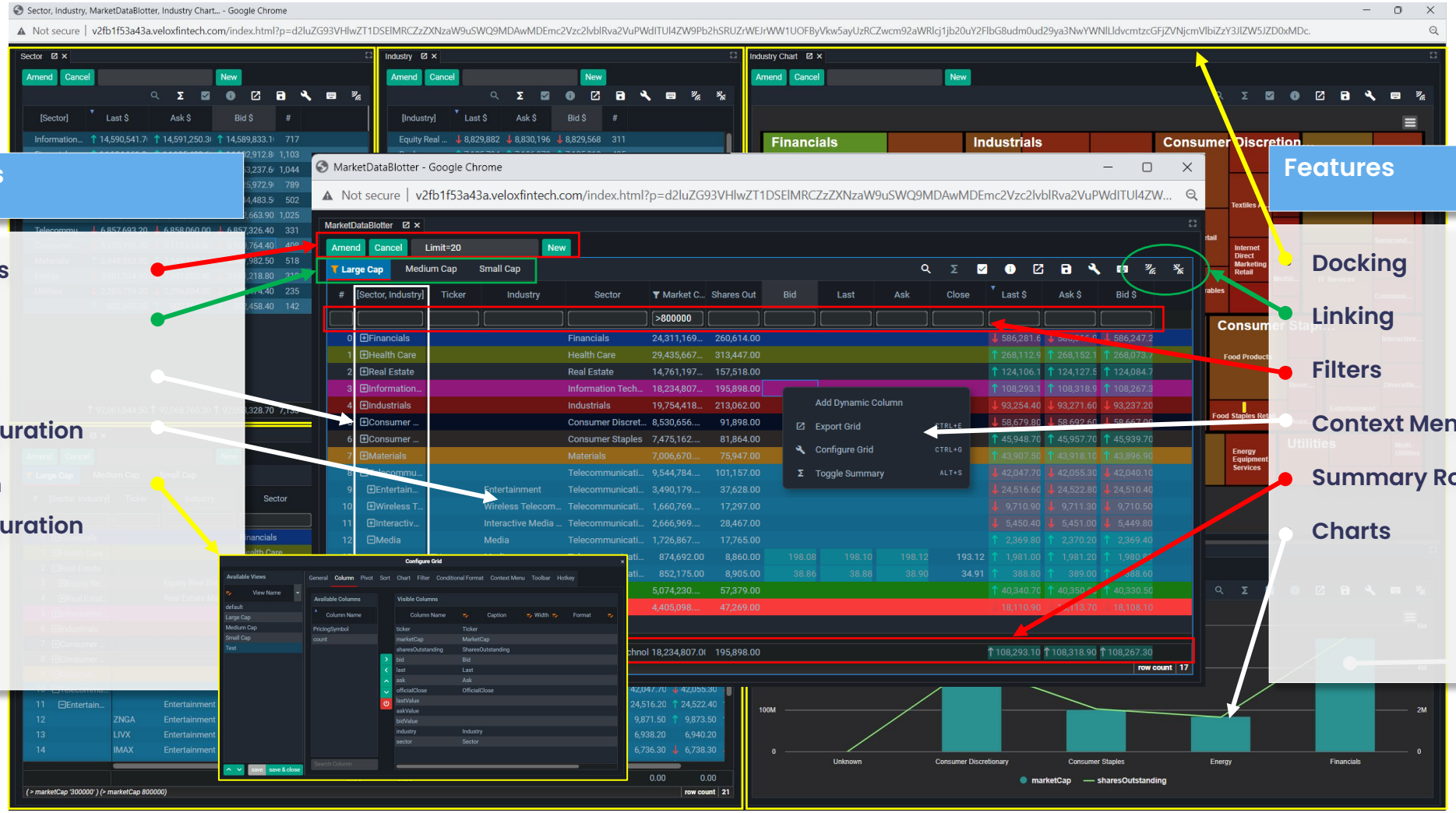
PROJECT DISTRIBUTION

```
root/
+-- gradle/
+-- src/
|   +-- main/
|       +-- java/
|       |   +-- com/velox/demo/
|       |       +-- Application.java
|       |       +-- PivotDemoWideAdaptor.java
|       |       +-- PivotDemoWideScreenProvider.java
|       |       +-- HandlerImpl.java
|       +-- resources/
|       |   +-- configuration/views/
|       |       +-- PivotDemoWideConfig.json
|       +-- velox/
|       |   +-- dictionary/
|       |       +-- PivotDemoWideDictionary.xml
|       +-- webapp/
|       |   +-- layouts/
|       |   |   +-- html/
|       |   |       +-- com/velox/demo/api/
|       |   |           +-- PivotDemoWideLayout.html
|       |   +-- png/
|       |   +-- index.html
|       |   +-- login.html
+-- .gitignore
+-- build.gradle
+-- gradle.properties
+-- gradlew
+-- gradlew.bat
+-- settings.gradle
+-- README.md
```

Part 2 : Building a Simple Data Monitor Application

Sections

A Simple Actionable Data Viewer

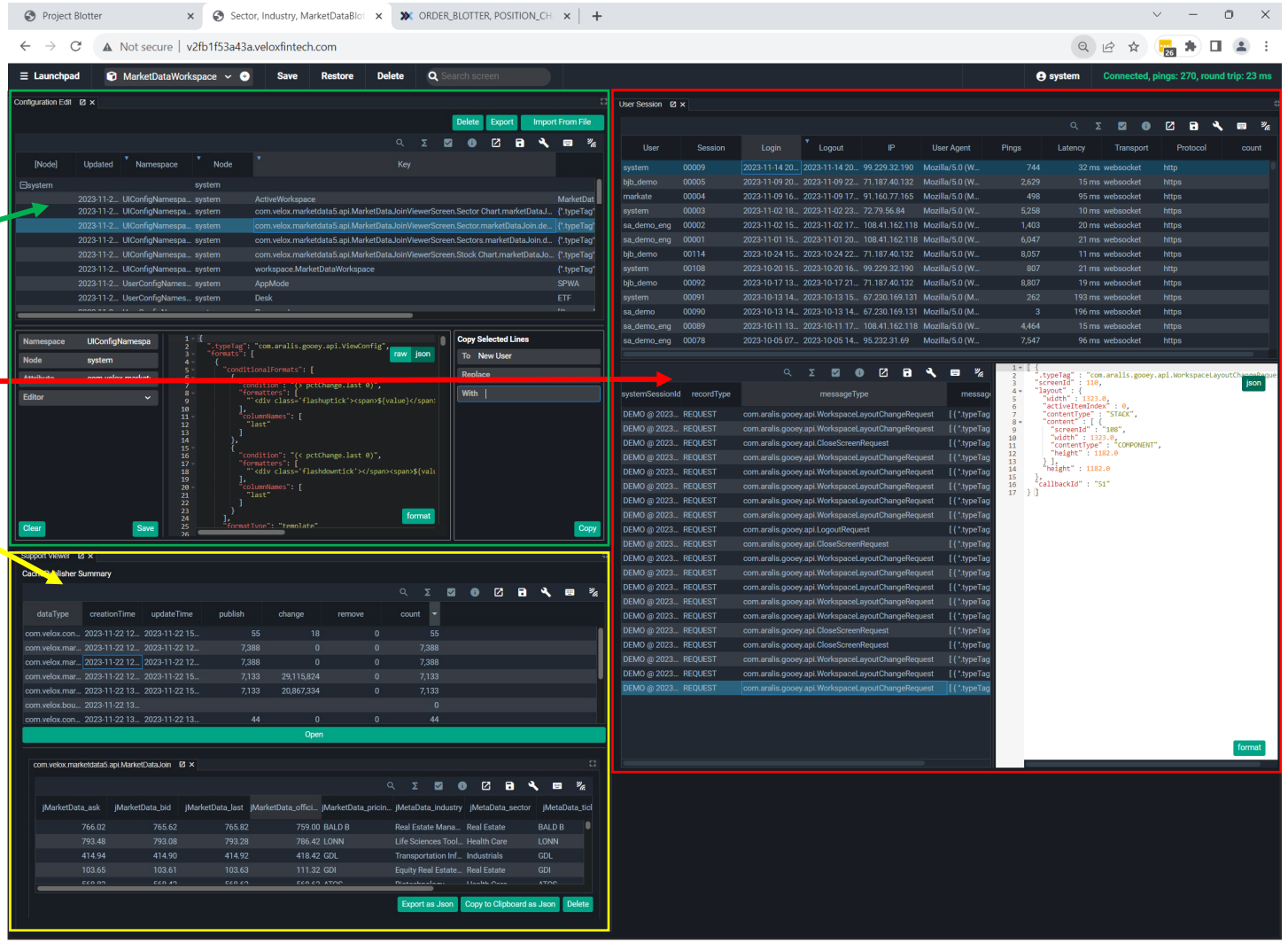


build faster

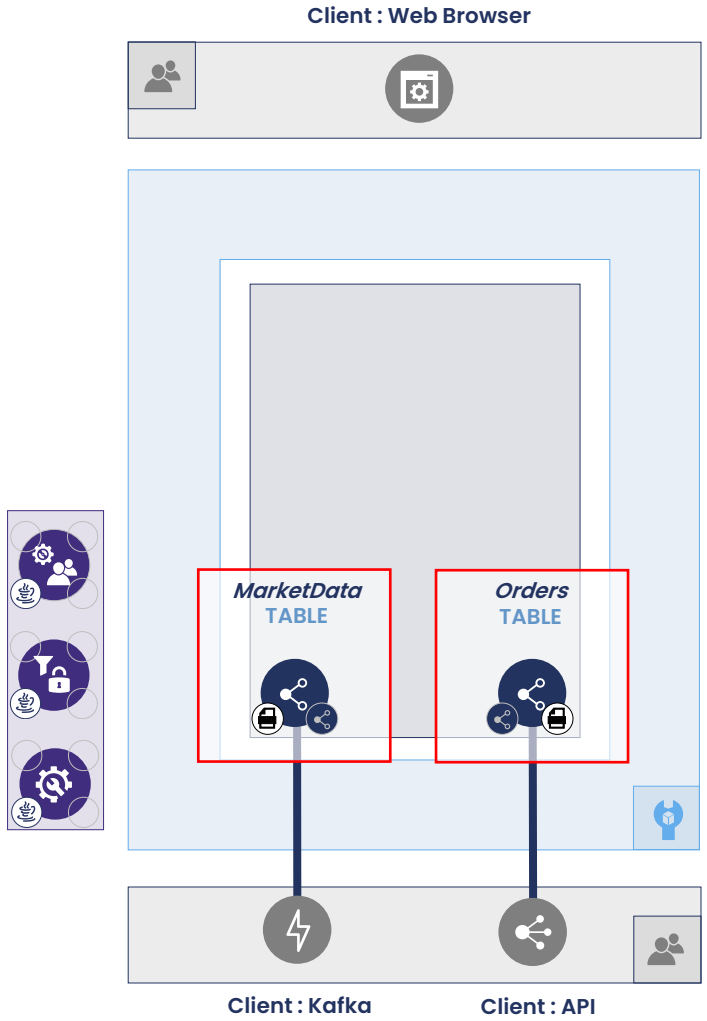
A Simple Actionable Data Viewer

Features

- User Configuration Manager
- Session Manager
- Support Viewer



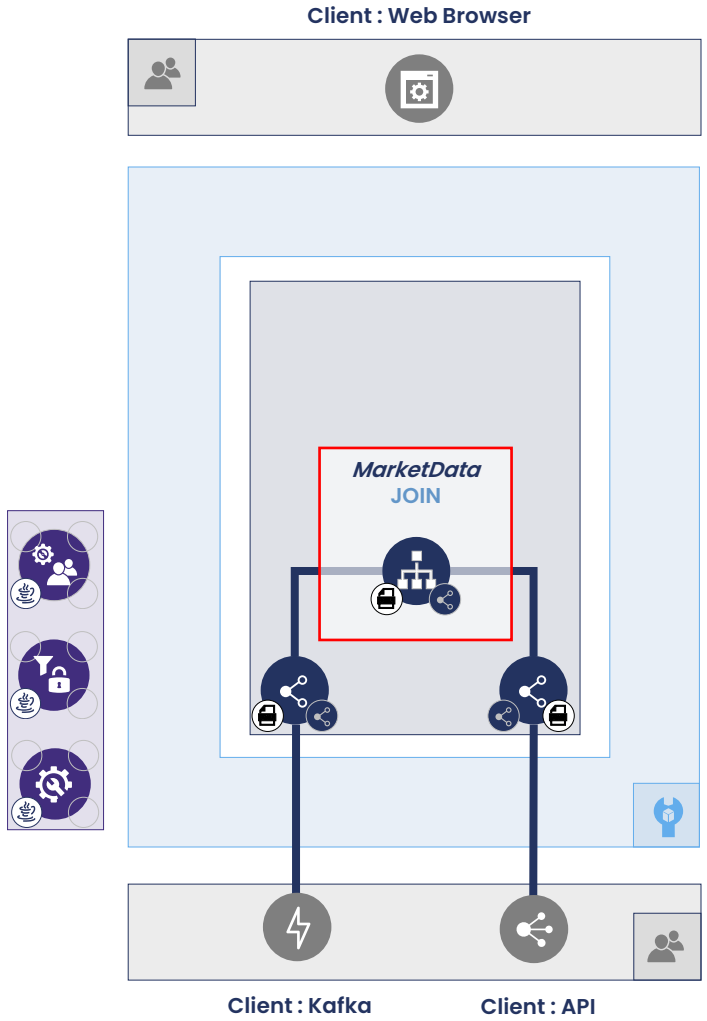
Building the Graph : Defining Tables




```
<table name="MarketData">
    <field name="id" primaryKey="true" type="String"/>
    <field name="bid" type="Double"/>
    <field name="ask" type="Double"/>
    <field name="symbol" type="String"/>
    ...
</table>
```

```
<table name="Orders">
    <field name="id" primaryKey="true" type="String"/>
    <field name="side" type="String"/>
    <field name="countryName" type="String"/>
    ...
</table>
```

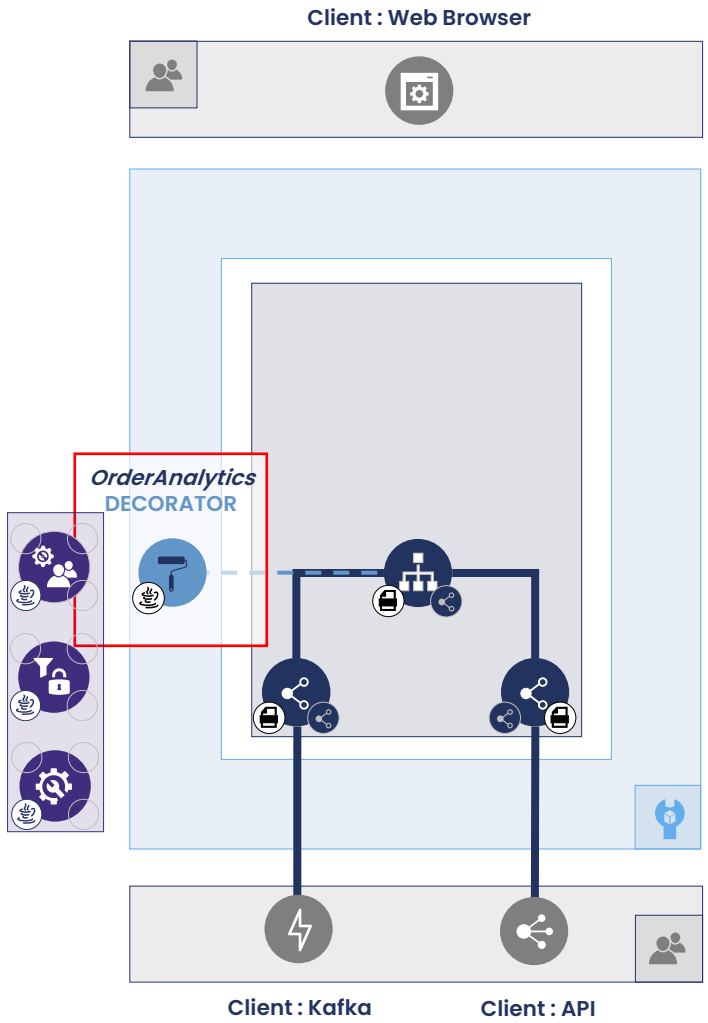
Building the Graph : Defining Joins




```
<join name="OrderJoin" primary="Order" primaryName="order">
    <field name="mktData" type="MarketData" root="ticker" joined="ticker" />
    <field name="orders" type="Orders" root="ticker" joined="ticker" />
    <field name="analytics" type="OrderAnalytics"
           prePublishDecorator="OrderAnalytics::calcOrderAnalytics"/>
    ...
</join>
```

```
<table name="Orders">
    <field name="id" primaryKey="true" type="String"/>
    <field name="side" type="String"/>
    <field name="countryName" type="String"/>
    ...
</table>
```

Building the Graph : Defining Decorators

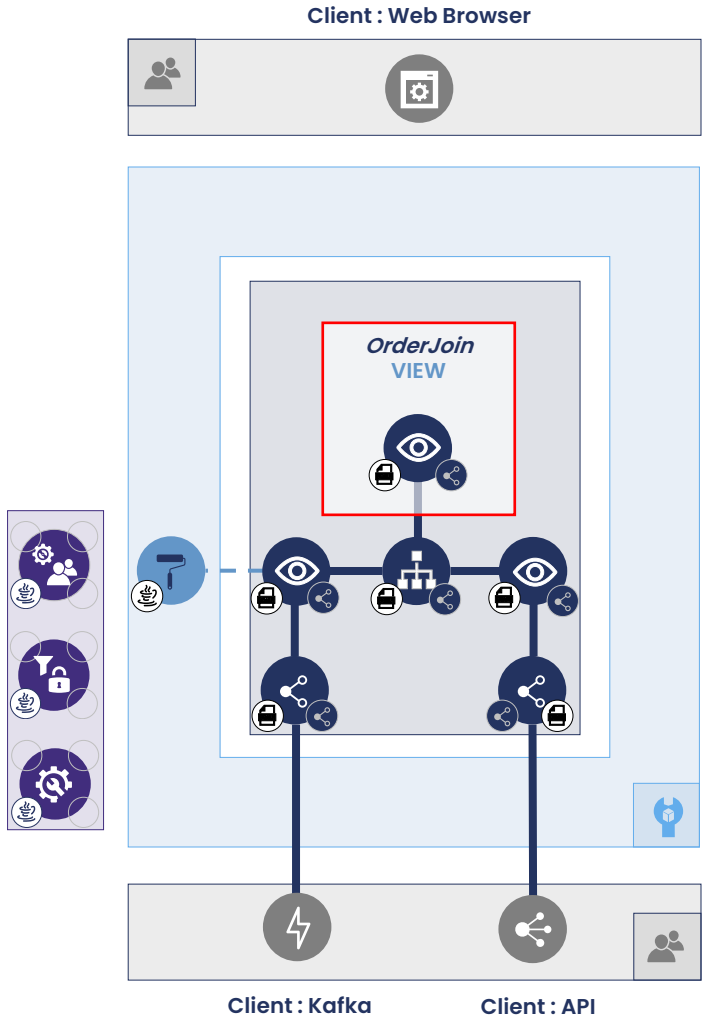



```
<join n
    <fi
    <fi
    <fi
    ...
</join>

<table
    <fi
    <fi
    <field name="countryName" type="String"/>
    ...
</table>
```

```
static public OrderAnalytics calcOrderAnalytics(final OrderJoin oldJoin,
final OrderJoin newJoin) {
   if (newOrder.mktData() == null && newOrder.security() == null) {
      return null;
   }
   double askVal = newOrder.order().orderQty() * newOrder. mktData().ask();
   double lastVal = newOrder.order().orderQty() * newOrder.mktData().last();
   double totalVal = askVal + lastVal();
   double execVal = newOrder.order().execQty() / newOrder.mktData().avgPx();
   return OrderAnalyticsBuilder.newBuilder().execVal(execVal).get();
}
```

build faster

Building the Graph : Defining Views



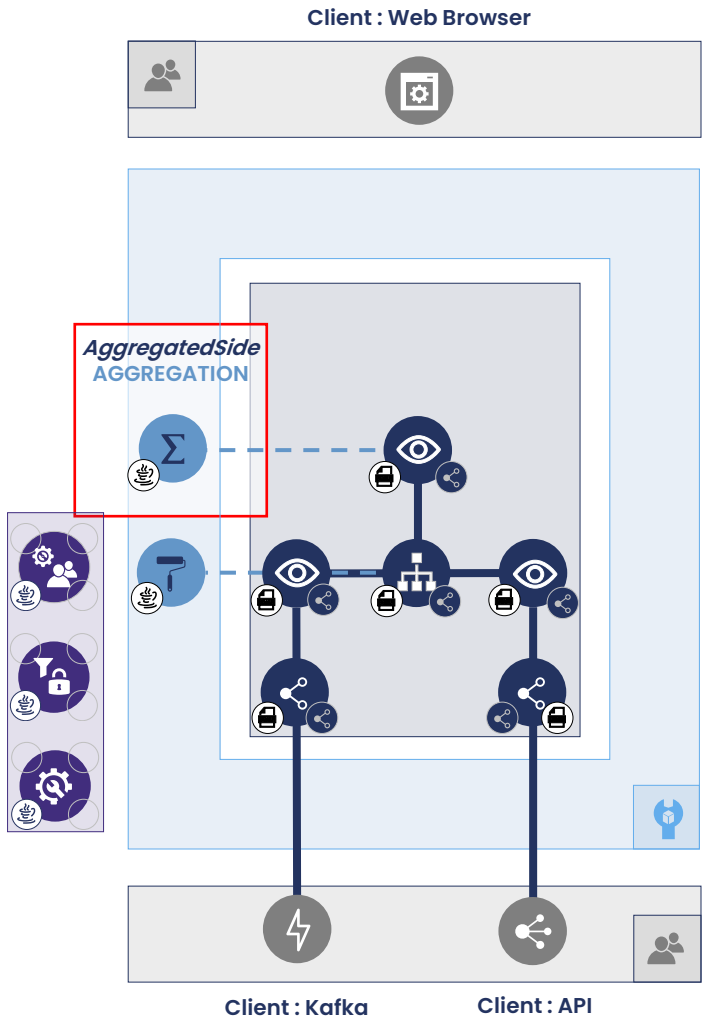

```
<view name="OrderJoinView" table=" OrderJoin ">
    <column name="side" source="side" aggregation="customSet" caption="Side"
      <aggregationParameters type="AggregatedSide"
           converter="com.velox.demo.AggregatedSide" />
    </column>
    ...
</view>
```

```
<join name="OrderJoin" primary="Order" primaryName="order">
    <field name="mktData" type="MktData" root="ticker" joined="ticker" />
    <field name="orders" type="Orders" root="ticker" joined="ticker" />
    <field name="analytics" type="OrderAnalytics"
           prePublishDecorator="OrderAnalytics::calcOrderAnalytics"/>
    ...
</join>
```

```
<table name="Orders">
    <field name="id" primaryKey="true" type="String"/>
    <field name="side" type="String"/>
    <field name="countryName" type="String"/>
    ...
</table>
```

build faster

Building the Graph : Defining Aggregations

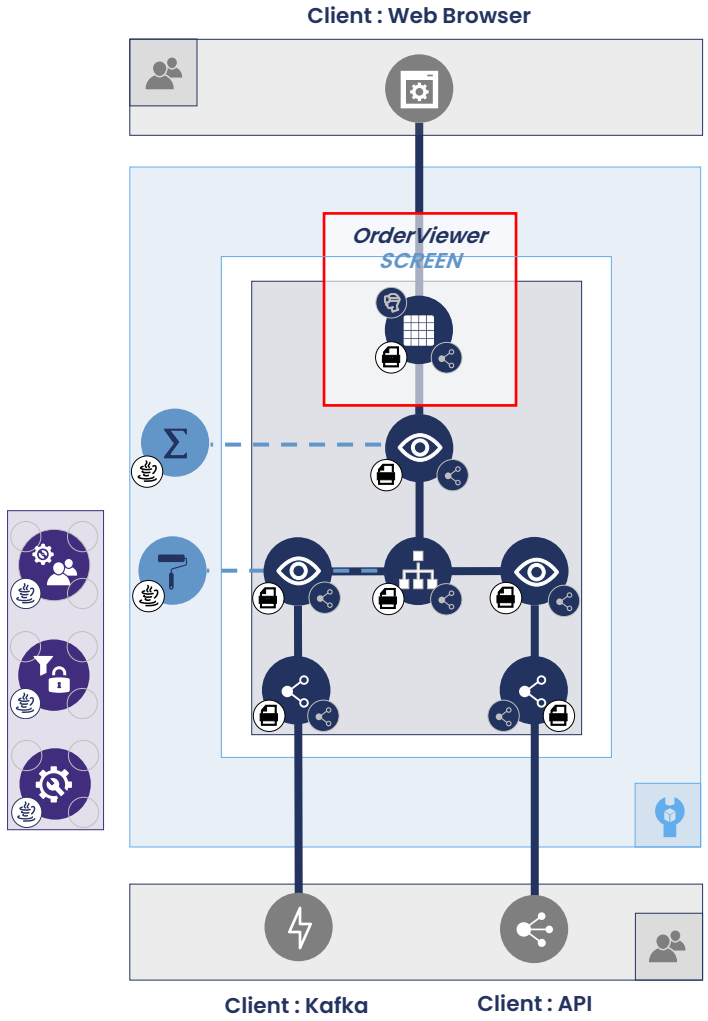



```
public static AggregatedSide aggregatedSideFromOrderSide(Set<Side> sides) {
   boolean containsBuy = sides.contains(Side.Buy);
   boolean containsSell = sides.contains(Side.Sell);
   if (containsBuy && containsSell) {
      return AggregatedSide.BuySell;
   } else if (containsBuy) {
      return AggregatedSide.Buy;
   } else if (containsSell) {
      return AggregatedSide.Sell;
   } else {
      return null;
   }
}
```

```
    <field name="analytics" type="OrderAnalytics"
           prePublishDecorator="OrderAnalytics::calcOrderAnalytics"/>
    ...
</join>
```

```
<table name="Orders">
    <field name="id" primaryKey="true" type="String"/>
    <field name="side" type="String"/>
    <field name="countryName" type="String"/>
    ...
</table>
```

Building the Graph : Defining Screens



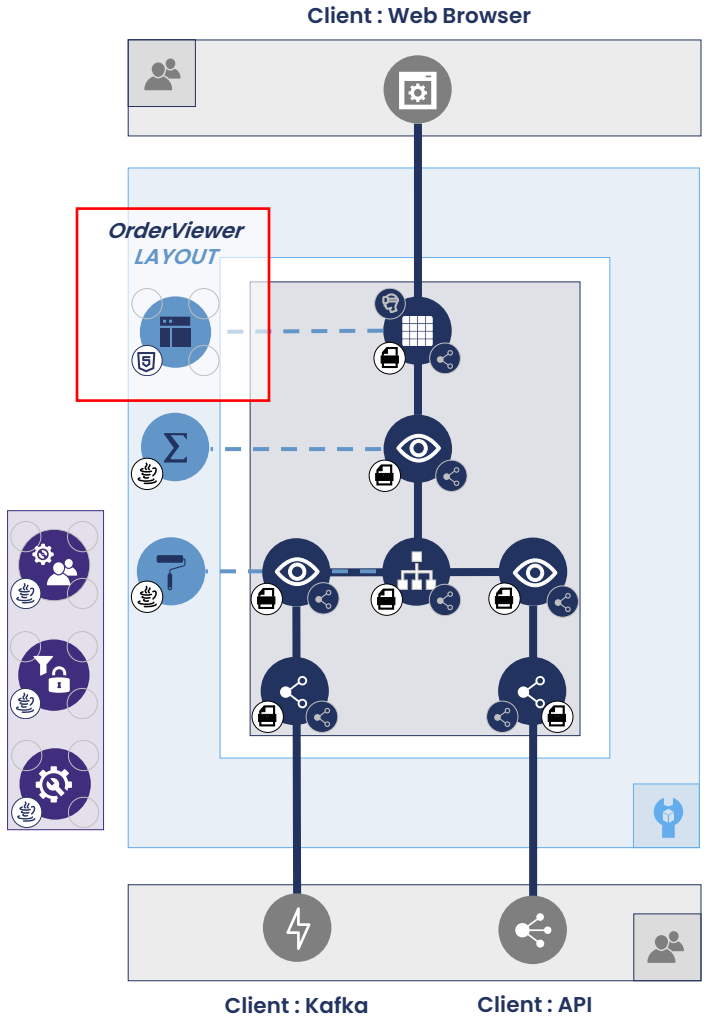

```
<screen name="OrderScreen">
    <control name="orders" type="datagrid" datatype="OrderJoinView"
        keytype="Object" viewname="OrderJoin"/>
    <control name="Amend" type="action"
    <control name="Cancel" type="action"/>
    <control name="New" type="action"/>
    <control name="Limit" type="singlevalue" datatype="String"/>
    ...
</screen>
```

```
<view name="OrderJoinView" table="OrderJoin ">
    <column name="side" source="side" aggregation="customSet" caption="Side"
      <aggregationParameters type="AggregatedSide"
          converter="com.velox.demo.AggregatedSide" />
    </column>
    ...
</view>
```

```
<join name="OrderJoin" primary="Order" primaryName="order">
    <field name="mktData" type="MktData" root="ticker" joined="ticker" />
    <field name="product" type="Product" root="ticker" joined="ticker" />
    <field name="analytics" type="OrderAnalytics"
            prePublishDecorator="OrderAnalytics::calcOrderAnalytics"/>
    ...
</join>
```

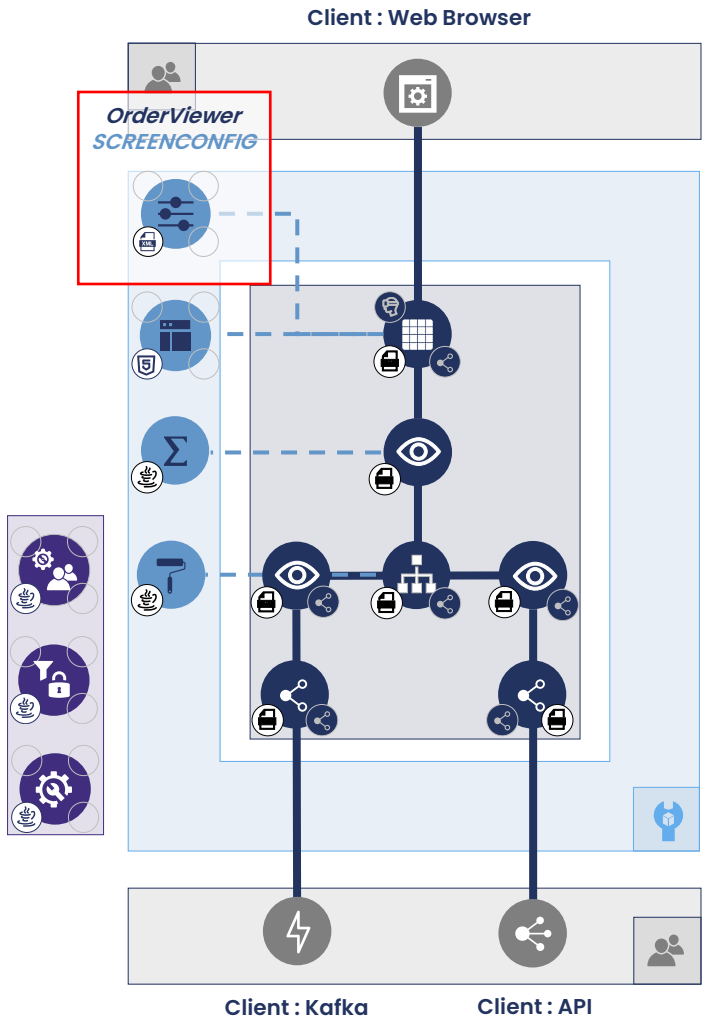
```
<table name="Orders">
    <field name="id" primaryKey="true" type="String"/>
    <field name="side" type="String"/>
    <field name="countryName" type="String"/>
    ...
</table>
```

Configuring Screens : Setting Layouts



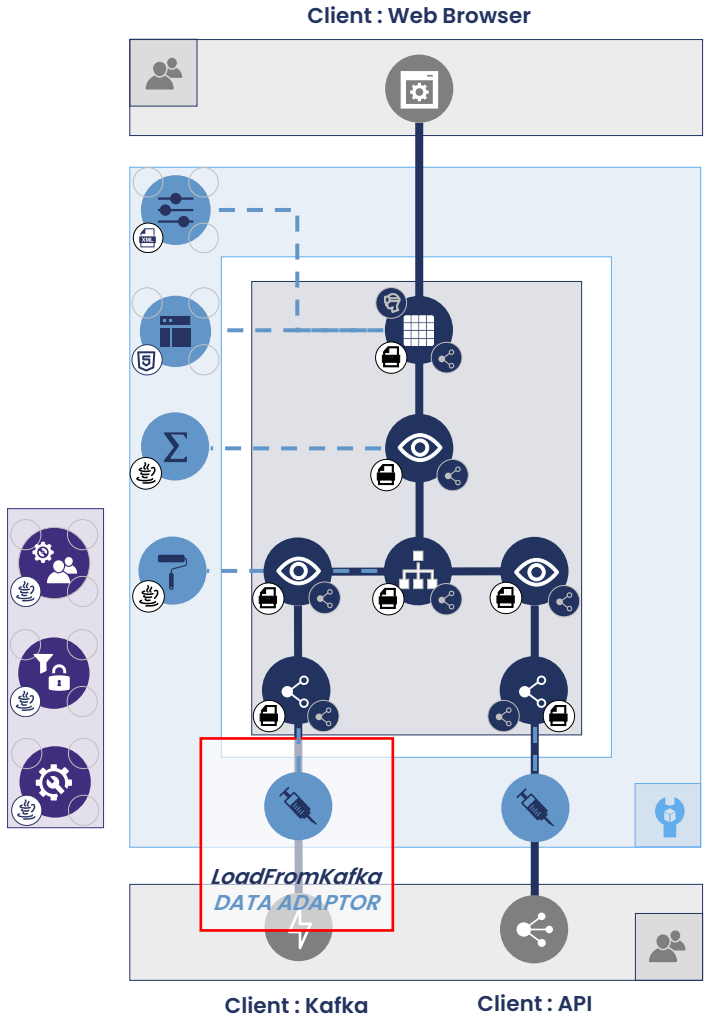

```
<div class="OrderJoinViewerScreen vx-screen">
    <div class="flex-row flex-none gap-between">
            <vx-action class="button btn-primary btn-sm"
                :vm="Amend">Amend</vx-action>
            <vx-action class="button btn-primary btn-sm"
                :vm="Cancel">Cancel</vx-action>
            <div class="labeled-input">
                <label data-content="Limit"></label>
                <vx-input :vm="Limit"></vx-input>
            </div>
            <vx-action class="button btn-primary btn-sm"
                :vm="New">New</vx-action>
    </div>
    <vx-data-grid :vm="OrderJoin"></vx-data-grid>
</div>
```

Configuring Screens : Setting Screen Config



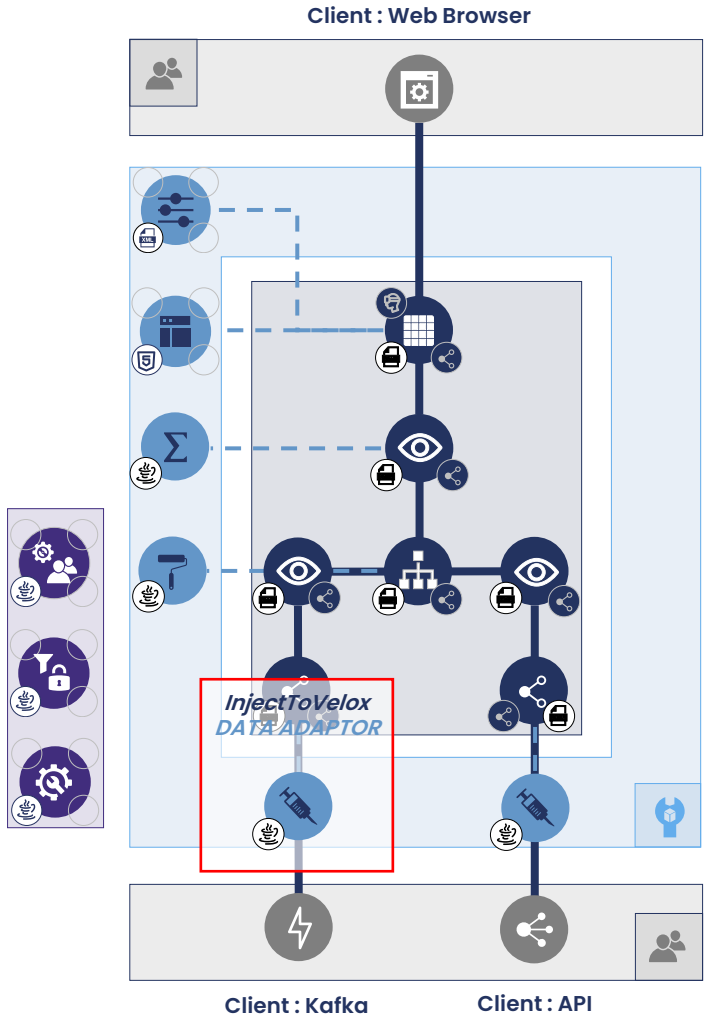

```
"com.velox.config.VeloxConfig" : {
   "node" : "system",
   "value" : "defaul,", // defined views
   "key" : "com.velox.api.rfqEntry._availableViewConfigs"}

"com.velox.config.VeloxConfig":{
   "usernode":"Demo", // value could be ALL USERS
   "value":"com.gooey.api.ViewConfig": {
      "filters": {
         "filterExpr":"(OR(==productType'NDF')_
                        ((==productType'SWAP'))"},
      "sorts": {
         "notional ": "ASCENDING"}},
      "columns":{
         "visibleColumns":[{"columnName":"tradeDate"},
                           {"columnName":"", // . . . . more columns _
      "pivots":["productType"]}
      "columnSettings":
         {"columns":{"name":{"color":"#4a148c"}}
      "formats\":{}
         "conditionalFormats": [{
            "condition": "(== Side 'Sell')",
            "formatters": ["bg-sell"],
            "columnNames": ["Direction"],"attribute": "CUSTOM"}]},{
            "condition": "( < perfClose '-0.9')",
            "formatters": ["#ce4040"], "attribute": "FOREGROUND"        }
   "key":"com.velox.demo.api.marketdatademo.ClientHistory"
```

Building the Graph : Defining Data Adaptors




```
public void loadFromKafka(String uri) {
   var c = new KafkaConsumer(props);

   // connect to relevant KAFKA topic
   c.subscribe(topics);
   ObjectMapper m = new ObjectMapper();
   while (true) {
      ConsumerRecords<String, String> records = c.poll(1ms);
      for (ConsumerRecord<String, String> record : records) {
         // convert update to a velox structure //
         JsonNode node = m.readTree(record.value());
         // write update to velox table //
         InjectPbvTbl (node); }}}
```

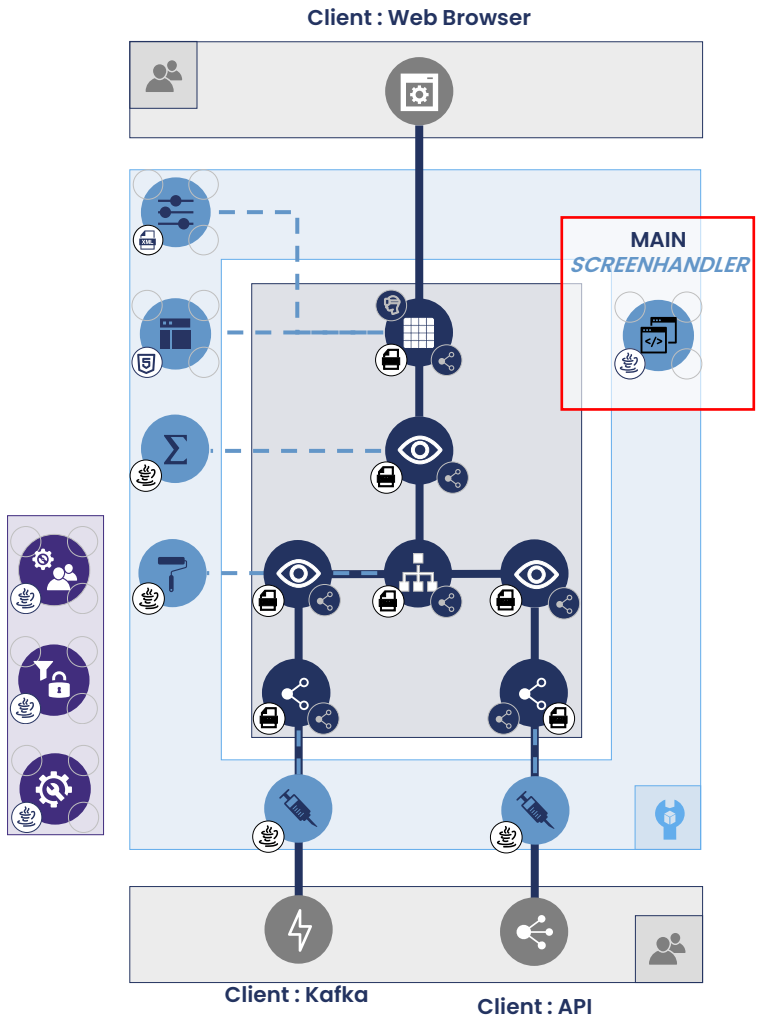
Building the Graph : Defining Data Adaptors




```
public void loadFromKafka(String uri) {
   var c = new KafkaConsumer(props);

   // connect to relevant KAFKA topic
   c.subscribe(topics);
   ObjectMapper m = new ObjectMapper();
   while (true) {
      ConsumerRecords<String, String> records = c.poll(1ms);
      for (ConsumerRecord<String, String> record : records) {
         // convert update to a velox structure //
         JsonNode node = m.readTree(record.value());
         // write update to velox table //
         InjectPbvTbl (node); }}}
```

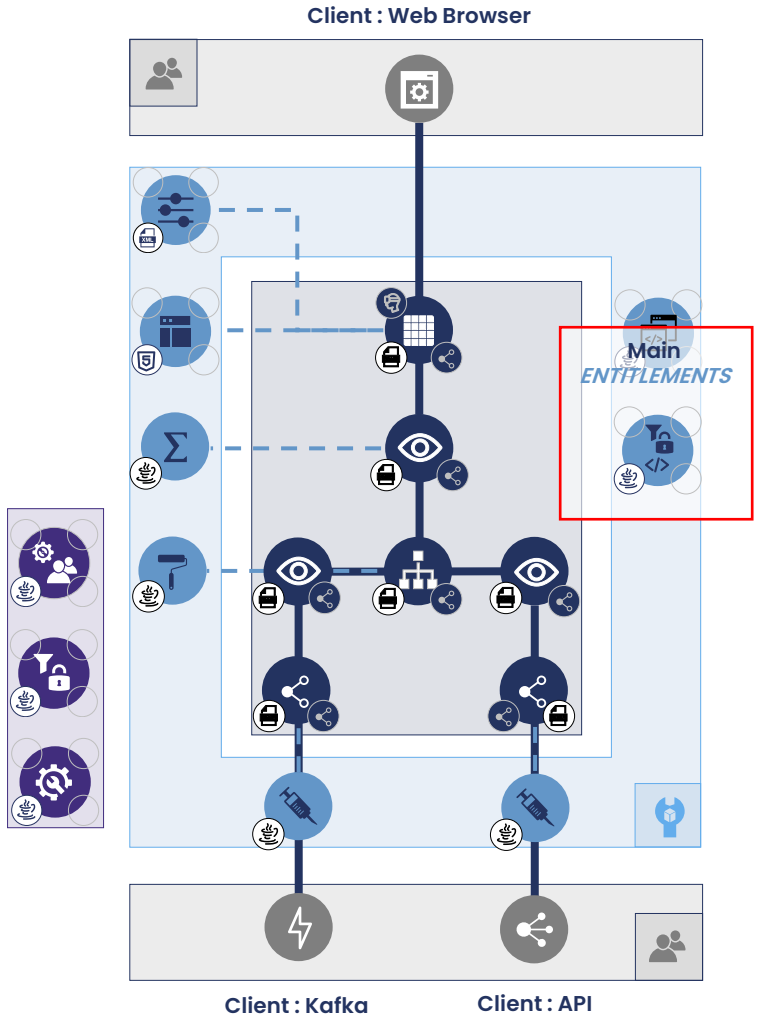
```
import com.velox.demo.api.pbvPub;

public void InjectPbvTbl (JsonNode rootDataNode) {
   JsonNode Json = rootDataNode;
   // get the reference to the pbo table in the velox cache //
   CachePublisher<pbvTbl, ?> pbvPub = m_dc.getPublisher(pbv.class);
   // iterate over each element in the update //
   for (JsonNode JsonElement: Json) {
      Builder Builder = Builder.newBuilder();
      // any pre-processing can be done here //
      // map each kafka field to a velox field //
      Builder.symbol(JsonHelper.getDouble(JsonElement,"symbol",0));
      // remaining fields to map go here //
      // publish to the cache //
      pbvPub.publish(Builder.get()); }}
```

Building the App : Coding Screen Handlers




```
public static void OrderJoinViewScreen createScreen(
    OrderClient OrderStoreClient, // interface to State Machine
    SessionState state, ClientNotifier notifier) {
    DataContextAccessor dc = state.getDataContextAccessor();
    OrderJoinView screen = new OrderJoinView (state));
}
```

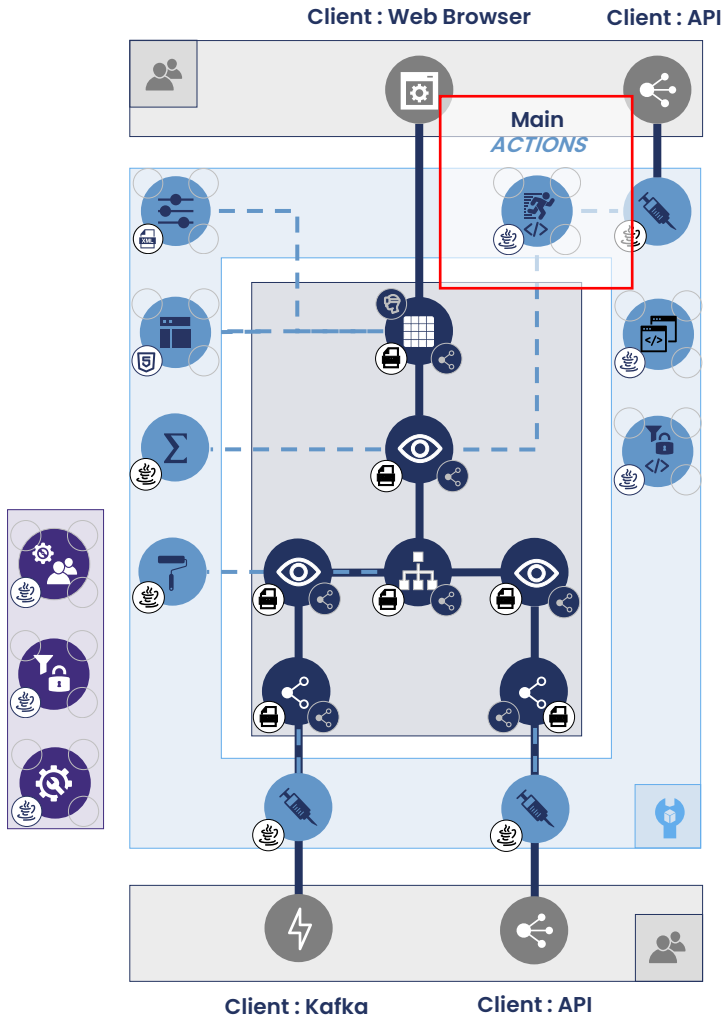
Building the App : Coding Entitlements




```
public static void OrderJoinViewScreen createScreen(
    OrderClient OrderStoreClient, // interface to State Machine
    SessionState state, ClientNotifier notifier) {
    DataContextAccessor dc = state.getDataContextAccessor();
    OrderJoinViewScreen screen = new OrderJoinViewScreen(state));
}
```

```
    UserConfigProvider ucp = state.get(UserConfigProvider.class);
    String strRole = ucp.getValue(UserConfigKey.Role);
    UserRole role = UserRole.Client;

    if (role == UserRole.Trader) {
        screen = new OrderJoinViewerScreen(state, OrdersJoinPub.getTable(),
        OrdersJoinView.ViewDefinition);
        screen.m_newOrder.visible(true);

        screen.m_clientName.setValue(client.clientName());
        screen.m_clientName.setOptions(Arrays.asList(client.clientName()));
        screen.m_clientName.enabled(false);

        MutableFilterListener<OrdersJoin, String> filterListener =
        MutableFilterListener.createForPublisher(FilterExpression.
            alwaysTrue(),ordersJoinPublisher);
        dc.getTable(OrdersJoin.class).subscribe(filterListener,
            screen.cancellationManager())
    }
```

Building the App : Coding Actions




```
public static void OrderJoinViewScreen createScreen(
    OrderClient OrderStoreClient, // interface to State Machine
    SessionState state, ClientNotifier notifier) {
    DataContextAccessor dc = state.getDataContextAccessor();
    OrderJoinViewScreen screen = new OrderJoinViewScreen(state));
    screen.title("ORDERENTRY");

    screen.m_Amend.setListener(onAmend(state, notifier, screen, helper));
    screen.m_Cancel.setListener(onCancel(state, notifier, screen, helper));
    screen.m_New.setListener(onNew(state, notifier, screen, helper));
    screen.m_Limit.setListener(onLimit(state, notifier, screen, helper))

    private static Consumer<Action> onAmend(SessionState state, ClientNotifier
        notifier, OrderJoinViewScreen screen, final OKCancelHelper
        helper) {
        return t -> {
            // TODO: add business logic to take the action
            helper.show("Amend action is triggered.");
        };
    }

    private static SingleValueEventListener onLimit(SessionState state,
        ClientNotifier notifier, OrderJoinViewScreen screen, final
        OKCancelHelper helper){
        return (t, u) -> {
            // TODO: add business logic for the input value change
            helper.show("Limit's new value is " + t.getValue().toString());
        };
    }
}
```

build faster

velox

build faster®