# Contents

[Document history 2](#_Toc41059326)

[Contents 3](#_Toc41059327)

[1. Introduction 4](#_Toc41059328)

[1.1 Summary 4](#_Toc41059329)

[1.2 Background 4](#_Toc41059330)

[1.3 Objectives 4](#_Toc41059331)

[1.4 Assumptions and Limitations 4](#_Toc41059332)

[2 High Level System Requirements 5](#_Toc41059333)

[3 High Level Design Concept 6](#_Toc41059334)

# Introduction

## Summary

The Purpose of this document is to explain various interfaces of Trade Capture of their Architecture and their components connect to capture the trades from source application to various ETRM applications.

## Background

The business requirement of the document is to serve as an important input for redesign and fine-tuning of the trading integration strategy.

As part of the implementation, various aspects have been considered such as business operational view, trading integration needs, existing technical implementation and technology footprint as well as integration operations. More important, the document addresses concrete integration challenges and draws future integration high-level architecture.

The Interface lists has been identified and prioritized as a part of the program, this document is covering only the transformation topics and not the central monitoring solution. Following interfaces are explained in this document.

* ICE
* Trayport
* Common Services

## Objectives

### Objective 1 - Emerging Business & IT Needs Transformation

### Objective 2 - Consolidate IT Landscape

### Objective 3 - Improve Operational Readiness and Associated Processes

## Assumptions and Limitations

|  |  |
| --- | --- |
| **N** | **Assumptions** |
| **A1** | The business functionality of the trade data will not be changed due to transformation. |
| **A2** | The Design of To Be Architecture has been made in considering the Architypes defined in the Frameworks and Standards |

# High Level System Requirements

The interface communication is required via streaming

The transformation is required via stream processing platform

The current mapping logic is required to be migrated to a Database

Central Monitoring System based on decision monitoring for Debugging and Monitoring

Resilience to make it robust towards failures is required

Proper authentication and security of credentials and authorization

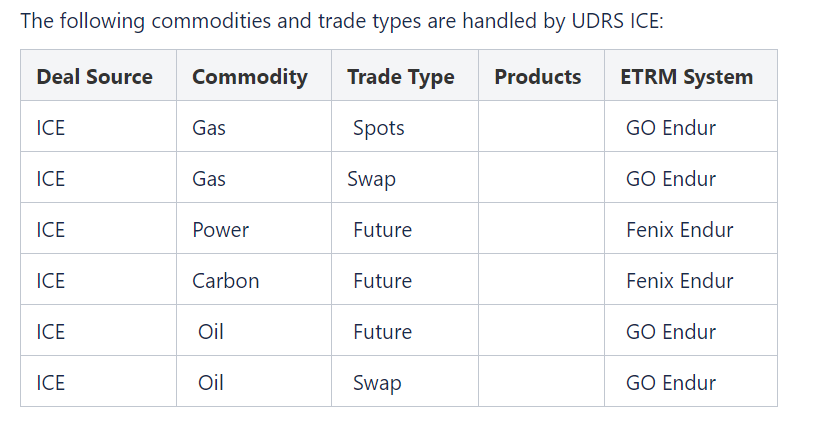
Capacity to work in High Volume of trades

#### **ICE AS – IS Structure**

UDRS ICE is the component of UDRS that gets the trades from Ice trading platform on the ICE exchange and forwards them to the ETRM systems.

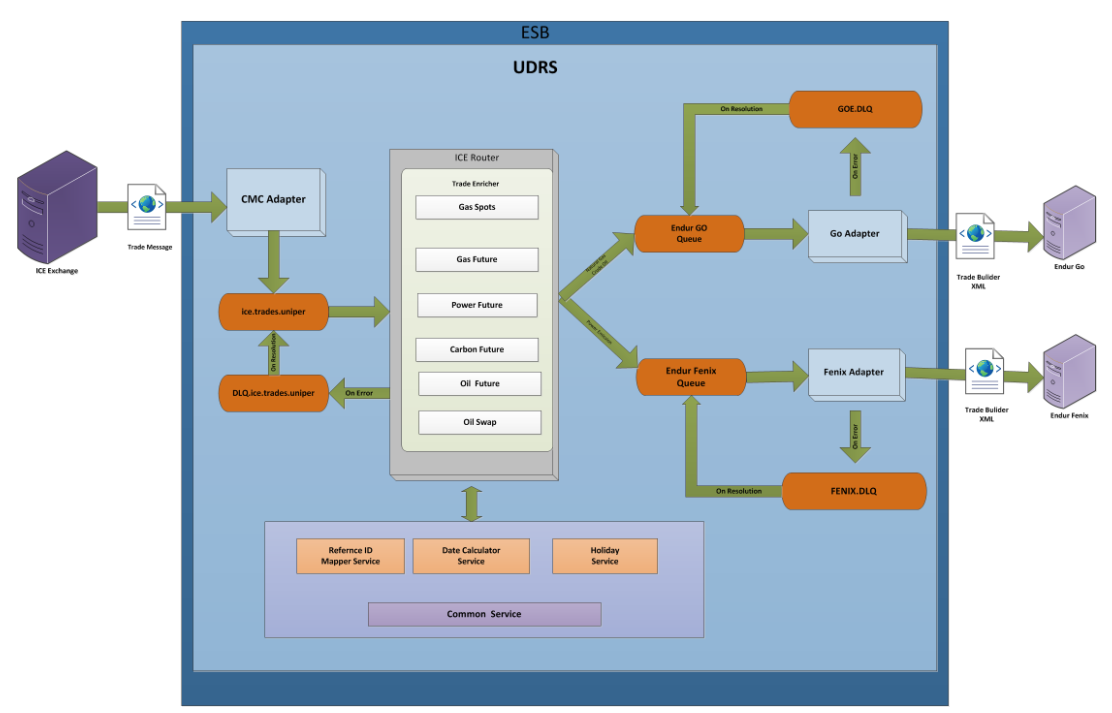
While routing the trades to ETRM systems, the trade data is enriched before they get booked in ETRM systems.

The sequence of routing of trades from ICE to Endur through UDRS is as shown below:



**TRADE Routing (UDRS ICE)**

The routing of Ice deals in UDRS, and their associated enrichers, is as shown below:



## **Trade Processing Steps**

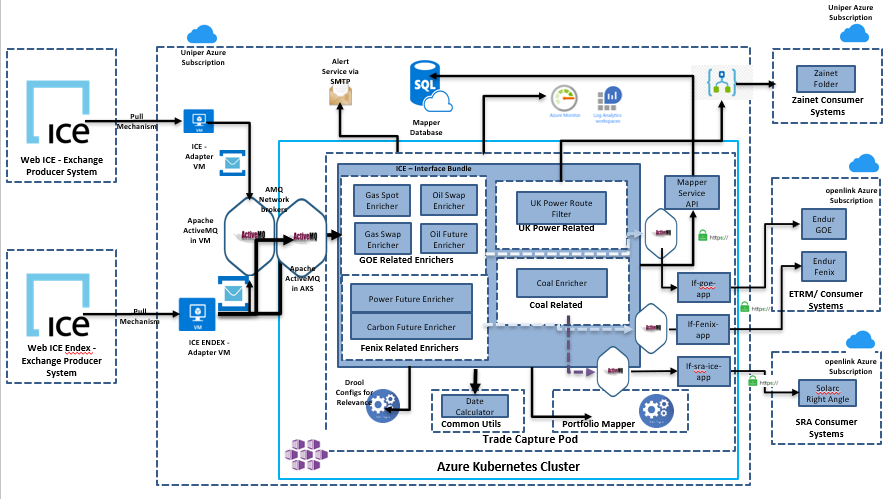
ICE Trade Router gets the trade from CMC Market Adapter, enriches the trades and then routes them to Endur systems. Trade Router has two entry points

1. Trade message from queue ICE.trades.uniper
2. Trade file from the input directory

The flow of trade data via message queue is as follows:

1. UDRS Ice subscribes for the trades from ICE exchange using the CMC ICE Adapter.
2. CMC ICE delivers the trade message to the specific queue in message bus.
3. Trade Router picks up the trade message from the queue.
4. Trade is then sent to trade filter class where trade type is derived based on the Contract code received from Exchange.
5. If the message contains valid Commodity and Trade type, it is sent to appropriate trade Enricher, otherwise it is ignored.
6. If there are any errors in the data it is sent to the UDRS.Ice DLQ for error resolution. Once it is resolved it is sent back to the Trade message queue.
7. Trade enricher uses following common services to enrich trades:
   1. **Reference ID Mapper** service to resolve counterparty and trader name.
   2. **Portfolio Mapper** service to derive the trade book.
8. Upon trade enrichment, trade data is transformed to Endur TradeBuilder XML schema.
9. Trade message in Endur-format is then put in trade message queue based on the target system.
10. Adapter component for target system picks up the message and uses Camel Endur component and drops the message to TIBCO bus with the required configuration parameters.
11. If any error is found in the message by the Adapter component, it is sent to the DLQ for resolution. On resolution it is sent back to the message queue of the target system.
12. Trade gets booked in respective ETRM system and a deal tracking number is generated as a response.

## **AS IS Diagram**



## **Problems of AS IS Diagram**

1. Lacking Resilience – Platforms lack adequate resilience to recover back to shape if in case of any issues.
2. Unable to work correctly due to high Volume trades.
3. Not able to perform data streaming in the AS IS structure.
4. Unable to handle failures with the masters and databases.
5. Lack Central Monitoring System based on decision monitoring for Debugging and Monitoring.
6. Lack proper authentication and security of credentials and authorization.
7. Lack Flexibility to build apps or add modular functionality to the old legacy system.

#### **Common Components**

#### **3.1.2.1 Introduction**

There are many interfaces in Trade Capture and all these interfaces needed few Common components which are re-used in most of the interfaces, currently these common components are executed in an interface specific bundles, now with the new design they are decoupled and executed independently to the interface. Some of the Common components are:

1. Date Calculator
2. Portfolio Mapper & Filter Rules
3. Config XML
4. Mapper Service
5. Holiday Service
6. Common Utils

The above common components are explained in detail in below sections

#### **3.1.2.2 Date Calculator**

This common component service must be used in enriching the start date, end date of a trade using the trade start date & trade end date coming from the source by validating with the holiday calendars.

AS-IS:

1. Currently this service is purely executing as a java program.
2. The holiday calendar is updated every year in last week of December.
3. Trades related to Hungary has a specific Holiday calendar.
4. The Calculation of start date and end date will vary based on the Interval type like BOM (Balance of Month), BOW (Balance of Week)
5. Here attaching the current source code of the Date Calculator

To -Be:

1. This service needs to be converted to a spring boot application.
2. Removing few old java dependencies like org.joda etc.
3. The functionality of the date calculation should not be changed.
4. The major services of this component should expose various API end points, So that these end points can be called out in various interfaces
5. Also Please send the Trade id while calling the API’s as an input so that the central monitoring solution can trace the input and output using the trade id.

## 3.1.2.3 Portfolio Mapper & Filter Rules

This common component service is used to enrich the Portfolio details of a trade.

As-Is:

1. This portfolio information is enriched by executing the rules which are maintained in the Drools application.
2. The Salience is maintained between the rules to provide the priority between the rules which will be used in defining the order of executing of the rules
3. No trade should not be enriched twice during execution of these rules
4. Here attached the sample rules which are configured in Drools.

To Be :

1. The Drool Rules to be migrated to a database in a table.
2. Using Kafka Confluent connector these rules to be migrated to Kafka Topic, So that the interfaces can use these rules to enrich the portfolio information.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Rule** | **ActivationGroup** | **Salience** | **Condition** | **Portfolio** | **Relevency** | **Status** | **User** |
| EMO\_1 | ICE | 100 | memo == "1" | MA\_OIL\_PRODUCTS\_USD | True | Validated | XX |
| EMO\_2 | ICE | 100 | memo == "2" | MA\_OIL\_DECOMPOSITION\_USD | True | Validated | XY |
| REL\_1 | ICE | 1000 | tradeType != null  && tradeType in ("K","E","S","V","O","9","J")  && commodity == Commodity.CARBON |  | False | Validated | XX |

Proposed Table structure

During Design of the interface the table structure can vary.

Note: How the rules in this table are maintained is out of scope for this program.

## 3.1.2.4 Config XML

This service is used to enrich template, commodity, clearing house & instrument type using the hub and product information coming the source applications.

As-IS:

1. Currently this mapping is maintained in a Config xml file.
2. Here attached sample xml file.

To Be:

1. The current config xml needs to be migrated to a database table
2. Using Kafka confluent connecter this mapping data needs to be synched to Kafka topic so that the interfaces can use this information and enrich the interfaces.

## 3.1.2.5 Mapper Service

This service is used to enrich broker, company, location, person, miscode information

As-is:

Currently the queries are executing using an API

Queries:

To Be:

1. Underlying query tables needs to be synched with Kafka topic using Kafka Confluent connector so that the interfaces can use this information and enrich the interfaces.

## 3.1.2.6 Generic Enricher

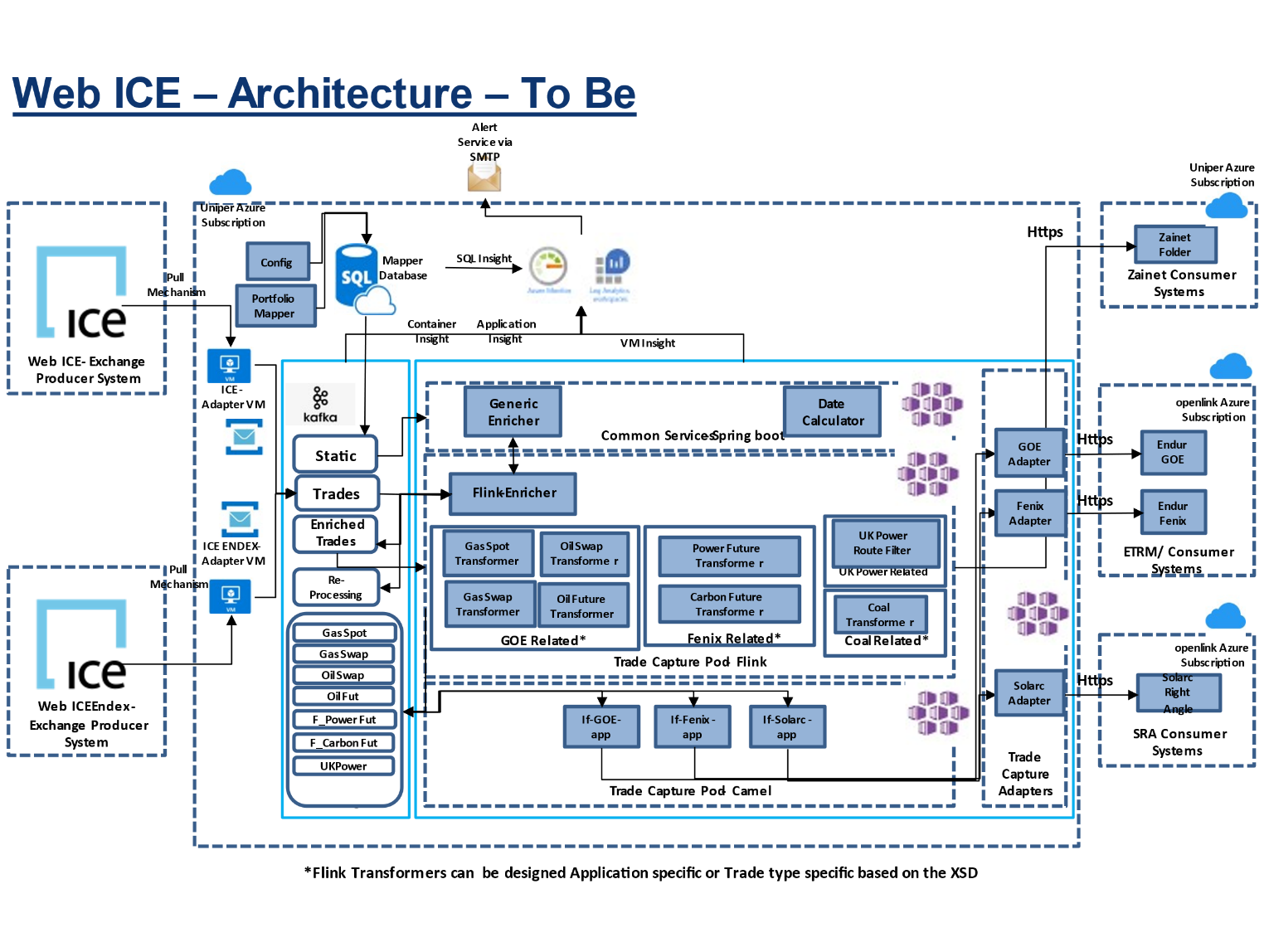
This is a new microservice which is defined in the new To-Be Architecture. This service is used to enrich the trades to a generic structure which has more enrichment attributes which are being calculated using the configurations, Rules, BU Data etc which are stored in the Kafka topic by the above-mentioned common services.

# HL Architecture Landscape (ICE Interface)

## Architecture

The following diagram depicts the various common components and ICE interface related components needed for the transformation.

Note: The topic names in Kafka can differ in the actual technical implementation.



Detailed Steps :

1. ICE Adapter & ICE ENDEX Adapter will retrieve the trade data from ICE Exchange and ICE ENDEX Exchange respectively
2. Trade messages will be published to ice.trades kafka topic
3. Flink Enricher will fetch trades from ice.trades kafka topic, Filter them based on the config data and will call the microservice (Generic Enricher) to enrich the trades into a generic Structure with all required attributes and place them in Kafka topic Enriched Trades.
4. In any case the trades didn’t get enriched then this information needs to be Alerted using central monitoring solution and the trade needs to be retried as per the current configuration by placing them into Kafka topic Re-processing
5. Each Flink job-transformers (Which are Trade type specific or Application specific) will fetch enriched trades and transforms them into application specific target structures and place them in each Trade type Kafka topics
6. Camel Jobs will read trades from different types of Trade type Kafka topics and pushes the trades to target specific adapters using circuit breaker resilience strategy
7. The individual adapters will push the trades to the target applications
8. Azure monitor will create required dashboard by using the Container insights, Application insights.

Note: The Generic Enricher microservice will uses BU Data, Portfolio mapper, Config data and Date Calculator etc common services and will enrich the trades.

## XML Mappings

The Flink jobs & Enricher Microservice needs to be designed in a way considering following XML Mappings

The Mappings are in Progress

## Central Monitoring Solution

|  |  |
| --- | --- |
| **PARAMETER** | **TASK** |
| Onboarding | * *The ICE interface being onboarded would be added in Azure Monitoring tool.* |
| Collection | * *The trade logs of the application would be captured by Azure Monitor.* |
| Analyze | * *The logs captured would then be analyzed with help of Azure Log Analytics.* |
| Visualization | * *The analyzed events would be onboarded on Azure Monitor Dashboard with advanced viewing visualization using Power BI/Graphana.* |
| Reconciliation | * *Comparing the messages at source and destination and sending alert on gaps identified.* |
| Notification | * *The alerts would be configured and notified over Email/SNOW to be resolved further by support team.* |
| ITSM | * *The alerts created and visualized in Dashboard would trigger a ticket in SNOW as per integration of SNOW with Azure monitor* |

## **3.2.4** Application Logging mapper

For logging requirements in the application code, we need to capture the following while defining application log files corresponding to all logging levels.

- Timestamp,

- Correlation id/TradeId

- Interface name

- Start time

- End time

- Any other specific processing info related to the functionality/module.

**Benefits of the New Architecture –**

1. Scalability - Handle scalability in all the four dimensions, i.e. event producers, event processors, event consumers and event connectors. Ability to scale upto thousands of node in a cluster.
2. High Volume - It can work with the huge volume of data streams, easily.
3. Data Transformation - It offers provision for deriving new data streams using the data streams from producers
4. Fault Tolerance - The Kafka cluster can handle failures with the masters and databases.
5. Performance - For both publishing and subscribing messages, model has high throughput. Even if many TB of messages is stored, it maintains stable **performance**
6. Zero Downtime - It is very fast and guarantees zero downtime and zero data loss.
7. Streaming Processor - It has a streaming processor, which can run both batch and stream programs
8. Ease of Integration - Can easily integrate with Apache Hadoop, Apache MapReduce, Apache Spark, HBase and other big data tools.
9. High Throughput - Processes data in low latency (nanoseconds) and high throughput.
10. Extensibility - It offers ways by which to write new connectors as needed. Fuse has a rich collection of APIs and management interfaces. The APIs have a very good ease of programming.
11. Reliability - The new Kafka implemented cluster can handle failures with the masters and databases.
12. Solid Debugging & Monitoring - It incorporates web management console and monitoring tools to make it is easier to track SaaS integrations.
13. Security-Support for different styles of credentials and authentication and authorization leveraging a role-based access control model.
14. Microservices - Containers are supported from the start with Fuse, so can create and deploy appropriate microservices immediately.Using low-code interfaces, create apps quickly that can scale and be resilient. Start in SpringBoot and add your containers.
15. Flexibility - Fuse is a single platform that can be deployed in three different ways, on-prem up to total cloud. You can build your apps once and then run them anywhere with no change in functionality or coding.
16. Enterprise Ready- It starts with the integration framework Camel and then adds a variety of extensions that enterprise developers can use to build in routes and connect JavaEE components. It can use code written in Red Hat, JBOSS, EAP. It can also easily provision scalable applications for the largest deployments.
17. Lightweight Containers - Have a lightweight, dynamic container. Typical J2EE application servers are too cumbersome and heavy, presenting too little value for today’s applications
18. Polyglot Programming Environment - It is language-agnostic and deliver multichannel, multicontainer capabilities wherein a single instance can host native code, .NET, Java and Spring, etc..
19. Out of the Box Capabilities - Out-of-the-box capabilities reduce implementation overhead. New Architecture has the capability to generate most of the boilerplate code for implementing the microservice. If not, it can create a late binding between the interface and the service implementation
20. Intelligent Routing - The system knows how to manage that traffic, which can be addressed through a mediation and intelligent routing process. It effectively route the right request to the right instance of a microservice

Sprint-2

### Nasdaq AS – IS

UDRS Nasdaq is the component of UDRS that gets the trades from Nasdaq and forwards them to ETRM systems. While routing the trades to ETRM systems, the trade data are enriched before they get booked in ETRM systems.

Diagram

Description automatically generated

**Trade Processing Steps**

Nasdaq Router gets the trade from Nasdaq exchange using the Exxeta Market Adapter Nasdaq, enriches the trades and then routes them to Endur systems. Trade Router has two entry points

1. Trade message from queue NASDAQ.MARKETADAPTERS.TRADES.PROD
2. Trade file from the input directory

The flow of trade data via message queue is as follows:

1. UDRS Nasdaq connects to the specific queue in the message bus.
2. XML message received from Nasdaq is validated against schema definition.
3. Trade is then sent to trade processor class where trade type is derived based on the Contract code received from Exchange.
4. Based on the Commodity and Trade type the trade is sent to appropriate trade Enricher.
5. If there are any errors in the data it is sent to the Nasdaq queue for error resolution. Once it is resolved it is sent back to the Trade message queue.
6. Trade enricher uses following common services to enrich trades:
   1. **Reference ID Mapper service**to resolve counterparty and trader name.
   2. **Portfolio Mapper service** to derive the trade book.
7. Upon trade enrichment, trade data is transformed to Endur TradeBuilder XML schema.
8. If any error is found in the trade data by the Adapter component, it is sent to the Fenix.DLQ for resolution. On resolution it is sent back to the Fenix message queue.
9. The trade is inserted into Endur Fenix via Rest

### Nordpool Day Ahead AS-IS

**Overview**

Nord Pool is a leading power market in Europe, and it offers day-ahead and intraday markets to their customers. The day-ahead market is the main arena for trading power, and the intraday market supplements the day-ahead market and helps secure balance between supply and demand. The Nordpool Day Ahead API can be used for entering orders to the Day Ahead Trading system automatically as well as obtaining trade results.

**Day Ahead Auctions API**

The Auction API can be used to retrieve trades and orders as well as order submission for the CWE Day-ahead auction. Auction mechanisms used by Nord Pool from time to time in the Physical Markets for

the submission of Orders and the calculation of the resulting electricity volumes and prices. Below are the two types of orders currently we are supporting as part of this implementation.

**Single hourly order ( Curve Order )**

The largest share of the day-ahead trading is matched based on single hourly orders. Curve order means an Order submitted to the Intraday Auction where a Participant states volumes to buy or sell at

different price levels in a set of Price Steps defined for a specific Delivery Period. Each pair of price and volume is handled as a point on an Order Curve with linear interpolation between each pair. Curve

Orders may have a granularity of either hourly, half-hourly or quarter-hourly (60 min, 30 min and 15 min) duration. Currently we are supposed to capturing only hourly orders.

**Block Order**

A block order consists of a specified volume and price for a certain number of consecutive hours within the same day. There are four types of block orders defined in Nord Pool markets: regular, profiled,

curtailable and linked. A block order is particularly useful for example if the participant wants to run a power station for a longer period than one hour and wish to minimize costs related to start and stop.

**The process of getting the Trades from Nordpool Auctions API involves**

1. Nordpool Day Ahead API service start.

2. The initial step in connecting to a server to obtain an authentication token, it has an expiration limit of 3600 seconds (1 hour).

3. Once the token has been generated need to fetch the trades from auctions api.

4. As the auctions are published once a day at 12:42 pm , we are fetching the trades using the above query on a scheduled basis (every day at 12:45 pm).

5. The trades that we are receiving from the exchange those are in JSON format.

6. Received auctions from exchanges are published to Fuse mq and also to the file share as well.

7. We have implemented the retry mechanism to fetch the trades again in case of below exceptions. (retryCount = 5 , retrydDelay = 15min)

### EPEX AS-IS

UDRS EPEX is the component of UDRS that gets the trades from EPEX and forwards them to ETRM systems. While routing the trades to ETRM systems, the trade data is enriched before they get booked in ETRM systems.  
The sequence of routing of trades from EPEX to Endur throught UDRS is as shown below:

Diagram

Description automatically generated

**Trade Processing Steps**

EPEX Router gets the trade from EPEX exchange using the Exxeta Market Adapter EPEX, enriches the trades and then routes them to Endur systems. Trade Router has two entry points

1. Trade message from queue EPEX.MARKETADAPTERS.TRADES.PROD
2. Trade file from the input directory

The flow of trade data via message queue is as follows:

1. UDRS EPEX connects to the specific queue in the message bus.
2. XML message received from EPEX is validated against schema definition.
3. Trade is then sent to trade processor class where trade type is derived based on the Contract code received from Exchange.
4. Based on the Commodity and Trade type the trade is sent to EPEX Trade Enricher.
5. If there are any errors in the data it is sent to the UDRS.EPEX DLQ for error resolution. Once it is resolved it is sent back to the Trade message queue.
6. Trade enricher uses following common services to enrich trades:
   1. **Reference ID Mapper service** to resolve  trader name.
7. Upon trade enrichment, trade data is transformed to Endur TradeBuilder XML schema.
8. If any error is found in the trade data by the Adapter component, it is sent to the Fenix.DLQ for resolution. On resolution it is sent back to the Fenix message queue.
9. The trade is inserted into Endur Fenix via Rest

### EPEX AS-IS

**Application Overview**

This interface automatically capture purchases of transport capacities that were booked on Prisma and transforms to Endur GO. Front office trades transport capacities for various markets on the capacity-trading platform Prisma. These deals are categorized as FCFS(Booking) deals and Auction deals. FCFS and Auctions can be bookings of single entry or exit transport capacities or bundle bookings which contain a pair of related entry/exit transport capacities.

Diagram

Description automatically generated

**Deal Flow and API Utilization at Middle ware/Fuse:**

Following sequence explains the high-level deal booking process:

1. Fuse polls the Rest API for every even and odd time intervals respectively and pulls/retrieves the data fcfs confirmation / auction confirmation from PRISMA .
2. The retrieved confirmations are of Json files and passed to the interface for further parsing and enrichment.
3. The user ID of the trader who booked the deal on Prisma is captured in the Shipper user email field of the Prisma message. EDRS maps the user email field value to an Endur user ID. In addition, a counterparty mapping is applied. The Prisma message contains different price components, like "REGULATED(regulatedCapacityTariff)" and "MEASURING(measuringOperationsFee)". These are captured separately on the deal in Endur, to enable invoicing as separate line items. In contrast, the "total price" field that is shown in the Prisma message is discarded.
4. EDRS then transforms the message to the Endur-standard trade builder format and posts the resulting XML document via Rest call to Endur.
5. Endur receives the message, adds missing data (comes from template) and books the transaction in status pending. The Endur job captures transport capacity bundles as one entry capacity transaction (COMM-CAP-ENTRY) and one linked exit capacity (COMM-CAP-EXIT). The Prisma ID of the deal has to be stored on the Endur deal, because it is later shown on the invoice. Information like strategy flag / mandate number is set automatically as possible.

In case of auctions the costs for the capacity are calculated. This can happen in two ways:  
a.    Automatic calculation via price template in Endur.  
b.    Manual calculation by traders who booked the deal and enter in Endur.

The trader checks and validates the transport capacity deal in Endur. Front Office is responsible to check completeness and correctness of automatically created deals in Endur. The Front Office trader confirms this by setting the status of the automatically created deal from "pending" to "new". Additional information (e.g. strategy flag / mandate number) are enhanced if required.

Prisma - EDRS Mapping

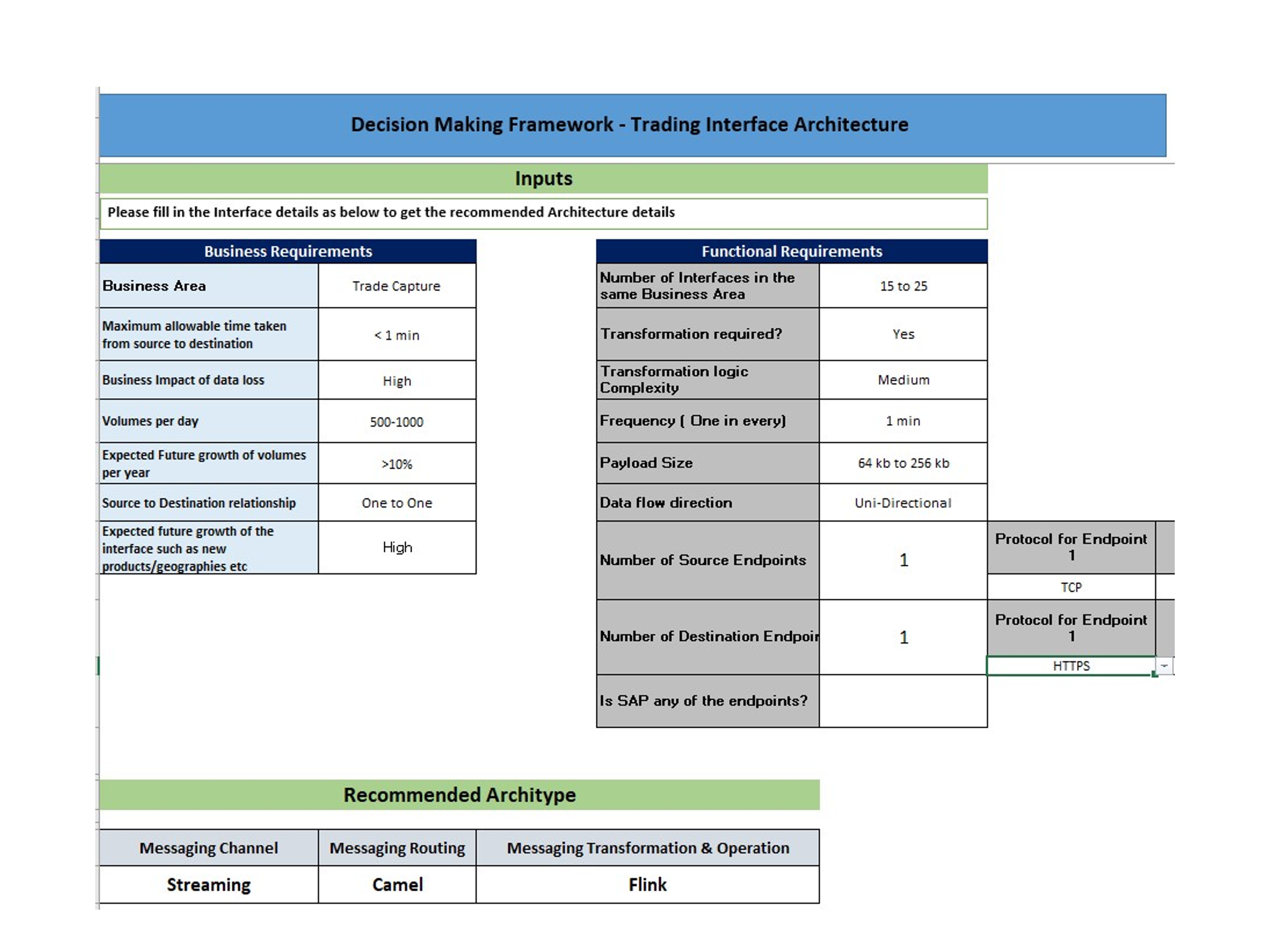
For the deals sent from Prisma to EDRS, there should be respective User, Location and Capacity mappings at EDRS side and this needs to be updated by Middleware team.  
In user mapping Prisma user would be mapped with Endur user, Location mapping contains Prisma location mapped with respective Endur location etc and vice versa. These details are updated in configuration file and would be available in Prisma -> Config.  
Whenever there is a missing mapping, trade will not process to Endur due to missing mapping  and appropriate alert is generated and sent to Fuse support teams.

.

**Problems of AS IS Diagram**

1. Lacking Resilience – Platforms lack adequate resilience to recover back to shape if in case of any issues.
2. Not able to perform data streaming and streaming operations.
3. Unable to handle notifications during failures..
4. Lack Central Monitoring System based on decision monitoring for Debugging andMonitoring.
5. Everything runs in a single bundle and microservice architecture is missing.
6. Re-usability of the components is less.

#### **Architecture Framework**



# To-Be Architecture

The To -Be architectures are explained below, Also the Adapter codes, Config, Drool rules etc can be found in the following link.

[https://dev.azure.com/uniperteamservices/P2591\_Integration%20Transformation%20Project/\_git/AS-IS-Scripts\_of\_Fuse?path=%2F&version=GBmaster](https://apc01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fdev.azure.com%2Funiperteamservices%2FP2591_Integration%2520Transformation%2520Project%2F_git%2FAS-IS-Scripts_of_Fuse%3Fpath%3D%252F%26version%3DGBmaster&data=04%7C01%7CMURALI.KANISETTY%40cognizant.com%7Ce3aaa54e58b84995350a08d960816570%7Cde08c40719b9427d9fe8edf254300ca7%7C0%7C0%7C637646931226213772%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=JEMmLlT4AVjxJOKObkRMiJPKYUvd6w9Nj8TDMmGBIps%3D&reserved=0)

## 

## Nasdaq To-Be Architecture

The following diagram depicts the Nasdaq To-be architecture which needed for the transformation.

Diagram, schematic

Description automatically generated

Detailed Steps :

1. Nasdaq Adapter will retrieve the trade data from Nasdaq Exchange and push the messages to Active MQ.
2. A Generic Kafka custom connect should be used (Based on the Parameter values of AMQ connection and Topic name) to move the trades from Active MQ to Nasdaq.trades
3. Flink Enricher will fetch trades from Nasdaq.trades kafka topic and will call the inbuild Generic Enricher jar to enrich the trades into a generic Structure with all required attributes and place them in Kafka topic Enriched Trades.
4. If the Trade is not relavent then the trades has to be moved to a Not Relevant topic with Source system, Payload & Comment about the Config
5. If the trade is relavent and in any case the trades didn’t get enriched then this information needs to be Alerted using central monitoring solution, and the trade needs to be retried as per the current configuration by placing them into Kafka topic Nasdaq.Re-processing
6. Each Flink job-transformers (Which are Trade type specific or Application specific) will fetch enriched trades and transforms them into application specific target structures and place them in each Trade type Kafka topics
7. Camel Jobs will read trades from different types of Trade type Kafka topics and pushes the trades to target specific topic and another camel job which reads all the trades and push them to target using circuit breaker resilience strategy.
8. Azure monitor will create required dashboard by using the Container insights, Application insights.

## Nordpool Day Ahead To-Be Architecture

The following diagram depicts the Nordpool Day Ahead To-be architecture which needed for the transformation.

Diagram, schematic

Description automatically generated

Detailed Steps :

1. Nordpool Day Ahead Adapter will retrieve the trade data from Nordpool Exchange and push the messages to Kafka topic Nordpool.trades
2. Flink Enricher will fetch trades from Nordpool.trades kafka topic and will call the inbuild Generic Enricher jar to enrich the trades into a generic Structure with all required attributes and place them in Kafka topic Enriched Trades.
3. If the Trade is not relavent then the trades has to be moved to a Not Relevant topic with Source sytem, Payload & Comment about the Config
4. If the trade is relavent and in any case the trades didn’t get enriched then this information needs to be Alerted using central monitoring solution, and the trade needs to be retried as per the current configuration by placing them into Kafka topic Nordpool.Re-processing
5. Each Flink job-transformers (Which are Trade type specific or Application specific) will fetch enriched trades and transforms them into application specific target structures and place them in each Trade type Kafka topics
6. Camel Jobs will read trades from different types of Trade type Kafka topics and pushes the trades to target specific topic and another camel job which reads all the trades and push them to target using circuit breaker resilience strategy.
7. Azure monitor will create required dashboard by using the Container insights, Application insights.

## EPEX To-Be Architecture

The following diagram depicts the EPEX To-be architecture which needed for the transformation.

Diagram, schematic

Description automatically generated

Detailed Steps :

1. EPEX Adapter will retrieve the trade data from EPEX Exchange and push the messages to Active MQ.
2. A Generic Kafka custom connect should be used (Based on the Parameter values of AMQ connection and Topic name) to move the trades from Active MQ to EPEX.trades
3. Flink Enricher will fetch trades from Nasdaq.trades kafka topic and will call the inbuild Generic Enricher jar to enrich the trades into a generic Structure with all required attributes and place them in Kafka topic Enriched Trades.
4. If the Trade is not relavent then the trades has to be moved to a Not Relevant topic with Source sytem, Payload & Comment about the Config
5. If the trade is relavent and in any case the trades didn’t get enriched then this information needs to be Alerted using central monitoring solution, and the trade needs to be retried as per the current configuration by placing them into Kafka topic EPEX.Re-processing
6. Each Flink job-transformers (Which are Trade type specific or Application specific) will fetch enriched trades and transforms them into application specific target structures and place them in each Trade type Kafka topics
7. Camel Jobs will read trades from different types of Trade type Kafka topics and pushes the trades to target specific topic and another camel job which reads all the trades and push them to target using circuit breaker resilience strategy.
8. Azure monitor will create required dashboard by using the Container insights, Application insights.

## Prisma To-Be Architecture

The following diagram depicts the Prisma To-be architecture which needed for the transformation.

Diagram, schematic

Description automatically generated

Detailed Steps :

1. Prisma Adapter (Build in Sprigboot) will retrieve the trade data from Prisma Exchange and push the messages to Kafka topic Prisma.trades (JSON)
2. A Camel job should use Filter stream and should fetch the relavent trades and push them to an Http endpoint which internally has a logic app which writes data to a Blob storage which then latter accessed by application Xceptor.
3. Another job which is Flink Enricher will fetch trades from Prisma trades kafka topic and will call the inbuild Generic Enricher jar to enrich the trades into a generic Structure with all required attributes and place them in Kafka topic Enriched Trades.
4. If the Trade is not relavent then the trades has to be moved to a Not Relevant topic with Source sytem, Payload & Comment about the Config
5. If the trade is relavent and in any case the trades didn’t get enriched then this information needs to be Alerted using central monitoring solution, and the trade needs to be retried as per the current configuration by placing them into Kafka topic Prisma.Re-processing
6. Each Flink job-transformers (Which are Trade type specific or Application specific) will fetch enriched trades and transforms them into application specific target structures and place them in each Trade type Kafka topics
7. Camel Jobs will read trades from different types of Trade type Kafka topics and pushes the trades to target specific topic and another camel job which reads all the trades and push them to target using circuit breaker resilience strategy.
8. Azure monitor will create required dashboard by using the Container insights, Application insights.

## ICE To-Be Architecture

The following diagram depicts the ICE To be Architecture

Graphical user interface, diagram, application

Description automatically generated

The changes needed to from Sprint 0 are :

1. Generic Enricher should work as a part of Flink job
2. If the Trade is not relavent then the trades has to be moved to a Not Relevant topic with Source sytem, Payload & Comment about the Config

## Endur Recon To-Be Architecture

The following diagram depicts the Endur Recon architecture which needed for the transformation.

Diagram

Description automatically generated

Diagram, schematic

Description automatically generated

Detailed Steps :

1. Kafka Connect Job will fetch Deals having external id in Endur GOE & Fenix
2. The Recon Flink Job has to compare the against the trades in Endur.GOE.Trades, Endur.Fenix.Trades topics with Endur.ExternalID topic and move the trades which are not in Endur.ExternalID to Recon Trades if the trade has not been retried for 5 times. If it is more then it has to move to EndurFenix/GOE.Recon.Notsuccessful topic.
3. The Camel job which is reading from Endur Goe , Endur Fenix specific trade types will also read the trades from Endur Fenix/GOE.Recon and push the trades to Endur specific application.
4. If a trade when they are moved to EndurFenix/GOE.Recon.Notsuccessful topic an email notification has to be sent to Support team.
5. The support team can manually push that trade to Endur specific Recon Trades to re process it.

## Central Monitoring Solution

|  |  |
| --- | --- |
| **PARAMETER** | **TASK** |
| Onboarding | * *The ICE interface being onboarded would be added in Azure Monitoring tool.* |
| Collection | * *The trade logs of the application would be captured by Azure Monitor.* |
| Analyze | * *The logs captured would then be analyzed with help of Azure Log Analytics.* |
| Visualization | * *The analyzed events would be onboarded on Azure Monitor Dashboard with advanced viewing visualization using Power BI/Graphana.* |
| Reconciliation | * *Comparing the messages at source and destination and sending alert on gaps identified.* |
| Notification | * *The alerts would be configured and notified over Email/SNOW to be resolved further by support team.* |
| ITSM | * *The alerts created and visualized in Dashboard would trigger a ticket in SNOW as per integration of SNOW with Azure monitor* |

## 3.2.4 Application Logging mapper

For logging requirements in the application code, we need to capture the following while defining application log files corresponding to all logging levels.

- Timestamp,

- Correlation id/TradeId

- Interface name

- Start time

- End time

- Success/ Failed

-Trade Type

-Enriched Attribute

-Enriched Value

-Trades Received

- Any other specific processing info related to the functionality/module.

**Benefits of the New Architecture –**

1. Scalability - Handle scalability in all the four dimensions, i.e. event producers, event processors, event consumers and event connectors. Ability to scale upto thousands of node in a cluster.
2. High Volume - It can work with the huge volume of data streams, easily.
3. Data Transformation - It offers provision for deriving new data streams using the data streams from producers
4. Fault Tolerance - The Kafka cluster can handle failures with the masters and databases.
5. Performance - For both publishing and subscribing messages, model has high throughput. Even if many TB of messages is stored, it maintains stable performance
6. Zero Downtime - It is very fast and guarantees zero downtime and zero data loss.
7. Streaming Processor - It has a streaming processor, which can run both batch and stream programs
8. High Throughput - Processes data in low latency (nanoseconds) and high throughput.
9. Reliability - The new Kafka implemented cluster can handle failures. .
10. Solid Debugging & Monitoring - It incorporates reading logs from applications and container logs and monitoring tools to make it is easier to track processing for trades and interfaces..
11. Security-Support for different styles of credentials and authentication and authorization leveraging a role-based access control model.
12. Microservices – Using Springboot microservices for Transforming the data which can be re-used for various integrations
13. Lightweight Application Camel is a lightweight application which transfers data from to target end points and heavy, presenting too little value for today’s applications