TRI Architecture and Services

TRI Sportsbook

Prepared for: Pragmatic Play
Prepared by: Fincore Product Development team
E igor.grdovic@fingaming.com
T +44 (0)20 7397 0620
P 47 Mark Lane, London EC3R 7QQ

Date: 25 March 2022 Version: 1.0 Classification: Restricted

> fingaming.com

Table of Contents

Version History

Version	Date	Author(s)	Status
0.1	15/12/2021	I. Vujovic, I. Grdovic	Introduced Betting Offer block chapter
0.2	30/12/2021	I. Grdovic, D. Rukavina	Introduced Bet Placement block chapter
0.3	14/01/2022	S. Gilezan, D. Rukavina, A. Mihajlov, Đ. Milovanovic, I. Vujovic, I. Grdovic	Corrections and additional embedded documents
0.4	24/02/2022	I. Grdovic	Introduced Bet Settlement block, Publishing and Backoffice Backend chapters
0.5	02/03/2022	C. Mullaparthi	Introduced ARC block chapter
0.6	08/03/2022	I. Grdovic	Introduced Custom Offer and Bet Transaction block chapters
0.7	10/03/2022	I. Grdovic	Introduced Alerts & Notifications block chapter
0.8	22/03/2022	C. Mullaparthi	Introduced Player Stats Corrections in ARC, Profitability and Alerts & Notifications block chapters
0.9	24/03/2022	C. Mullaparthi	Attached Profitability and Alerting Internals documents
1.0	25/03/2022	I. Grdovic	Embedded latest docs. Initial “release” version.

Version Numbering

Document versions numbered “0.x”, are draft status and therefore can be changed without formal change control. Once a document has been formally approved and issued it is versioned “1.x” and all subsequent releases will be consecutively numbered, following formal change control. The version number appears in the footer of every page.

Distribution, Reviewers and Approvers

The table below defines the distribution plan for this document and defines who reviewers and approvers of the document using the following RACI model:

R : **R**esponsible for the production / updating of the document
A : **A**ccountable and approves the document
C : **C**onsulted on and provides input into the document (takes part in reviews/workshops)
I : **P**rovided the document for Information (but has no direct input into it)

Name	RACI	Note
Zeljko Vracaric	C	technical owner
Milan Vitorovic	I	product manager
Ivana Vujovic	A	development team-lead
Aleksandar Mihajlov	C	developer
Djordje Milovanovic	C	developer
Srdjan Mitrovic	C	developer
Caslav Mijuskovic	C	developer
Igor Grdovic	A	business analyst team-lead
Sorin Gilezan	R	business analyst
Davor Rukavina	R	business analyst
Bojana Lazovic	R	business analyst
Chandru Mullaparthi	R	development tech-lead

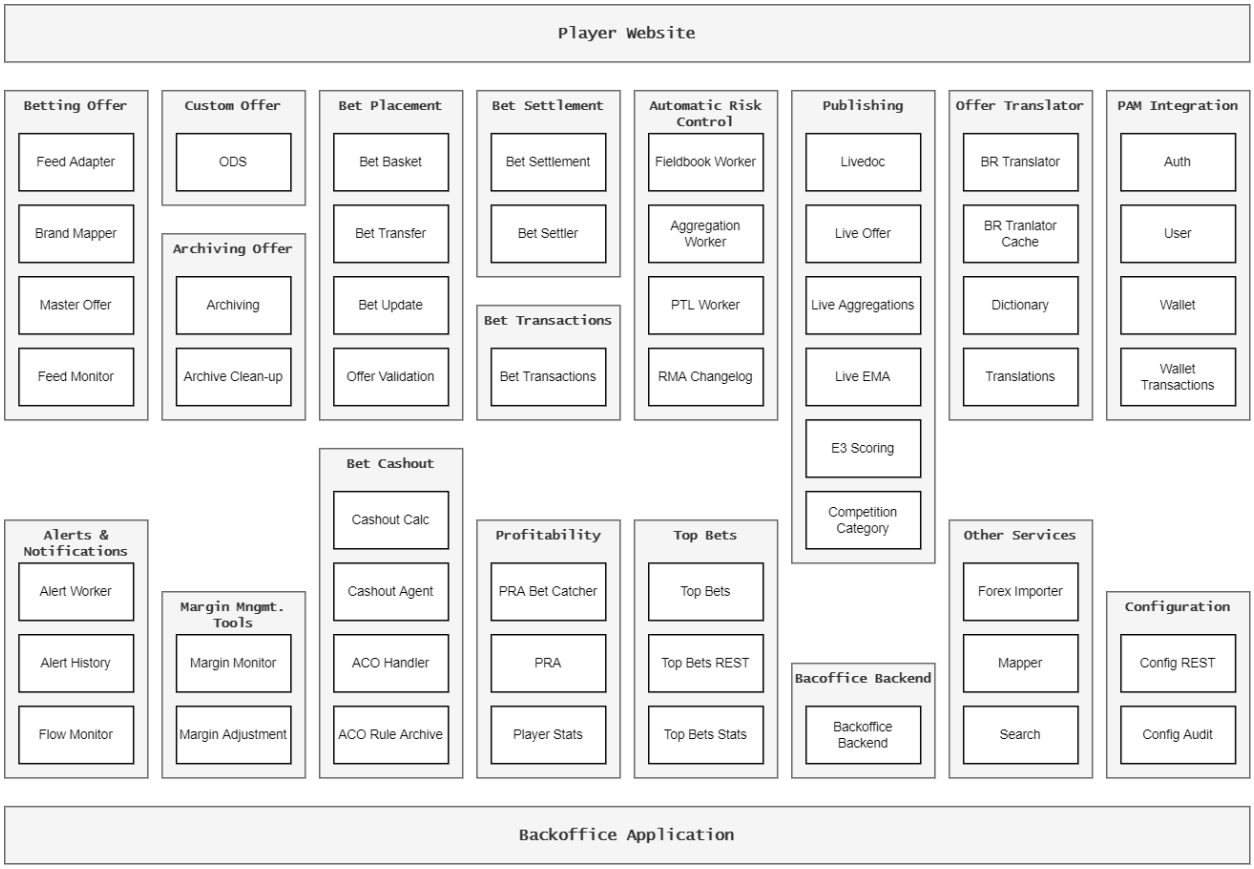
Overview

TRI Sportsbook is the latest and complete Fingaming solution for sports betting. Built on the Margin Maker heritage, TRI Sportsbook is capable of serving multiple brands having multiple channels and player websites. It covers the entire lifecycle of betting offer handling, provides player frontend, as well as the Backoffice Application to provide tracking and managing of both betting offers and bets.

Architecture Block Diagram

Following diagram presents major TRI Sportsbook building blocks. More detailed diagrams with description will be given in the further text.

`



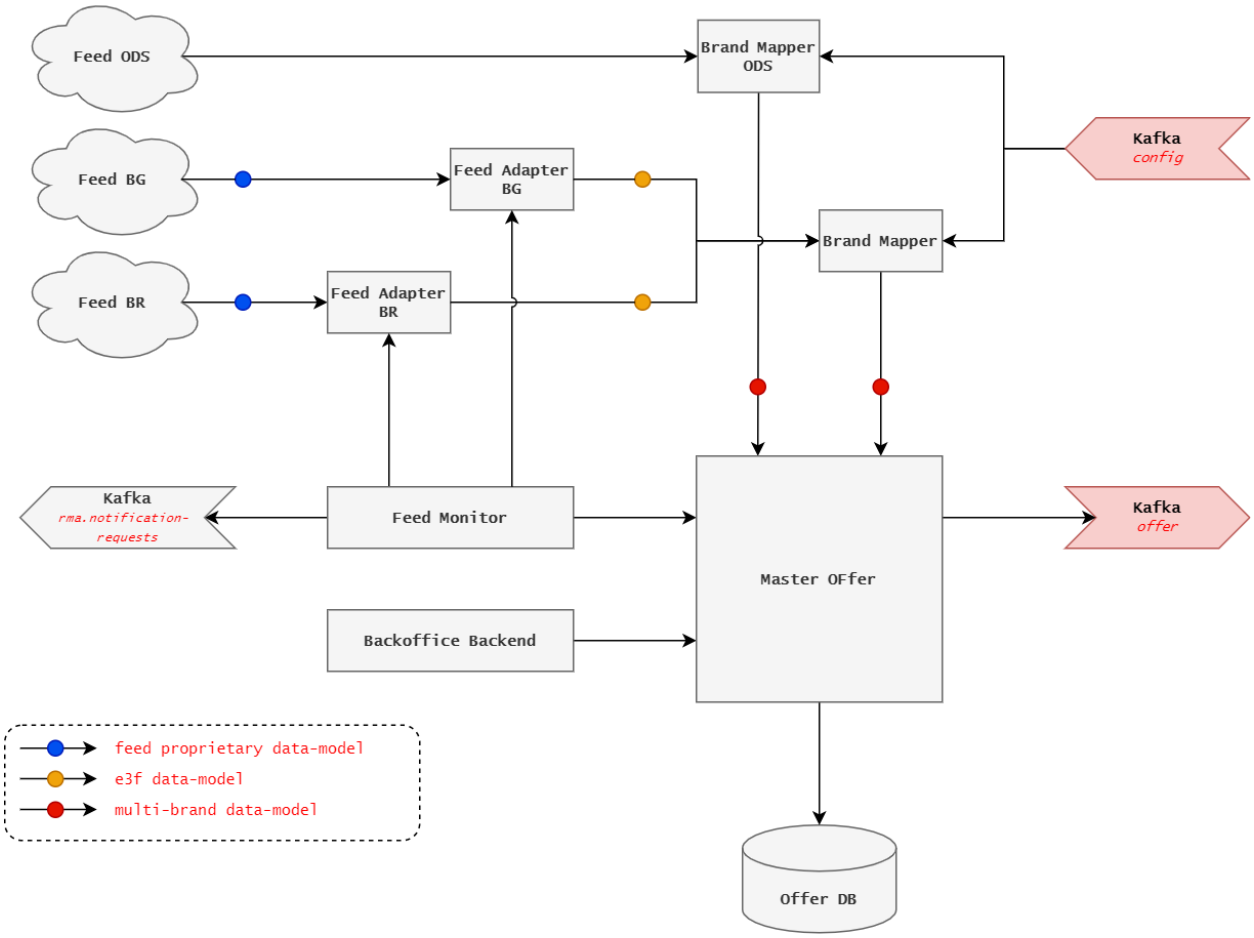
TRI Sportsbook core components block diagram

Betting Offer block

Betting Offer block contains components that are responsible for managing the lifecycle of betting offers objects: sports, competitions, venues, events, and markets. Starting from the input feeds that provide the source data, components in this block perform transformation of data into internal model, enriching the betting offer objects with additional properties and mapping them into betting programs of different brands.

Term “betting program”, as used here, relates to a full catalogue of betting offers that are made available for betting by bettors of a certain brand.

Terms “betting offer” or “betting offer object”, as used here, relate to an object of the betting program - be it a sport, a competition within a particular sport, an event within a particular competition, or a market within a particular event. In TRI Sportsbook market object encapsulates all dependant selection objects.



Betting Offer block logical diagram

Feed Providers

Terms “input feed”, “input data feed”, “feed provider” or just “feed” all relate to a concept of external third-party service that, for a fee, provides betting offer content and timely information updates (as things happen) through its proprietary integration point.

TRI Sportsbook is currently integrated with following input data feeds:

- Bet Radar (**BR**): This feed was the first one to be introduced. It’s integration point uses a combination of message queue (uses Advanced Messaging Queue Protocol) and REST API to provide betting offer content and its updates. Further documentation on Bet Radar feed can be found **here**.
- Bet Genius (**BG**): This feed uses an API to provide betting information. By using this API *Feed Adapter BG* opens a channel through which betting offer object messages are then pushed. Further documentation on Bet Genius feed can be found **here**.
- Own Data Source (**ODS**): This feed is TRI Sportsbook proprietary feed for serving custom-created betting offer objects and is a part of the in-house eco-system. Custom betting offer objects are maintained through the Backoffice Application UI by a custom events custodian user. More details related to ODS and its components will be given in further text.

Feed Adapter

Feed Adapter service is responsible for retrieving betting offer content from third-party data feeds and transforming it into an adopted internal data format that downstream component (Brand Mapper) understands called the “e3f data-model”.

Since all external third-party data feed providers have their own proprietary integration points through which they make betting offer available, a separate custom-made Feed Adapter service must be created for each of the processed input feeds. What this means is that for each external input feed there is a Feed Adapter instance that is tailored according to the interface used by that particular feed provider. Each Feed Adapter instance is, thus, able to “understand” the language of the dedicated data feed it processes, and to transform the incoming information into a common format which the downstream component can understand – the mentioned e3f data-model.

Once transformed to the e3f data-model, betting offer object is handed over to the Brand Mapper for further processing.

Feed Adapter service has a local database (*e3fdb*) in which it persists mapping data required to perform the transformation from the original data format into the e3f data-model format.

Feed Adapter service also implements an integrated monitoring mechanism that maintains an up-to-date information on whether the service manages to keep up with the feed, or how much it lags behind the feed provided data.

Brand mapper

Brand Mapper service is responsible for initial inserts of new betting offer objects into the Master Offer, and for all their subsequent updates due to information change provided by the originating feed. To perform these tasks, Brand Mapper converts the betting offer objects into the multi-brand data-model before handing it over to the Master Offer component.

When inserting a new betting offer object, Brand Mapper:

- receives a new betting offer object in e3f data-model format from the Feed Adapter
- converts it to a multi-brand model format by adding branded properties defined by the current brand mapper configuration into the betting object, and
- pushes the new betting offer object into the Master Offer component.

When updating an existing betting offer object based on feed-served updates, Brand Mapper:

- receives an updated betting offer object in e3f data-model format from the Feed Adapter
- retrieves the current betting object in multi-brand model format from the Master Offer component,
- applies the changes extracted from the updated betting offer object provided by the Feed Adapter,
- applies the change of branded properties defined by the current brand mapper configuration into the betting object; and
- pushes the updated betting offer object back into the Master Offer component.

By adding/applying the brand mapper configuration defined properties into the betting object, Brand Mapper effectively performs:

- distribution of betting offer objects across brands, thus forming and maintaining each brand's betting program; and
- addition/update of configuration defined branded properties on the betting offer objects.

Term "distribution of betting offer objects across brands" concerns a choice that can be made in the brand subscription configuration whether to include a particular betting offer object in the betting program of a particular brand when that object originates from a particular feed.

Term "branded properties" in general considers betting offer object properties that can be brand specific e.g., a status of the same event which may be different for two different brands. In the Brand Mapper context, branded properties which it adds/updates are the ones taken from the brand mapper configuration, like betting start time (according to the configured betting offset and announced event kick-off time) or in-play bet delay.

What adopting the principle of introducing branded properties means is that TRI Sportsbook does not perform the multiplication of entire betting offer objects for each of the brands. I.e., sport, competition, event, and market objects are not copied into separate versions of themselves, one for betting programs of each brand. Only branded properties are multiplicated within the betting offer objects instead.

Brand Mapper service retrieves up-to-date brand subscription configuration from the Kafka *configuration* topic. Brand subscription configuration is maintained through a Backoffice Application UI by a configurator user.

Master Offer

Master Offer is a REST service that is responsible for setting up and maintaining all betting offer content. It provides the following functionalities:

- receives betting offer objects and their updates extended with branded information based on brand subscription configuration from Brand Mappers
- receives global, per TRS or per brand property overrides that originate from the Backoffice Application via the Backoffice UI BE
- persists up-to-date original and overridden betting offer object information within the Master Offer DB
- maintains sports, competitions, venues, events, and markets in offer DB tables and produces messages to relevant Kafka topics (jointly termed "*offer*" topics group on diagrams) that carry up-to-date effective betting offer objects throughout the TRI Sportsbook; and
- provides up-to-date betting offer objects in different data-models to different consumers upon query.

Master Offer supports several data-models which are intended for different consumers throughout the TRI Sportsbook eco-system. These are:

- Multi-brand model: A data-model used by Brand Mapper services to push the betting offer objects. This model includes only feed-induced changes and does not hold any trader made overrides.
- Internal model (a.k.a. the database model): This model holds all data prepared for database persistence.
- Full model: A data-model holding all data used by various micro services.
- Trader model: A data-model holding data used by Backoffice Application.

Master Offer supports access to and management of betting offer objects by using each of this several data-models. When using the multi-brand data-model following main functions are supported:

- fetch, create or update a sport
- fetch, create or update a competition
- fetch, create or update an event
- fetch, create or update a market

Feed Monitor

As, from the business perspective, it is considered to be very important to constantly track system performance in regards with the speed of processing feed updates and react accordingly, Feed Monitor service has been created to fulfil two main purposes in cases when processing input feed data starts lagging behind:

- to raise appropriate alerts for Backoffice Application users; and
- to suspend betting on affected events

To achieve this, Feed Monitor service periodically queries Feed Adapter service on the feed processing status by using the REST exposed on the Feed Adapter side. In case that feed

processing lags behind, information provided in the Feed Adapter REST response describes how big this lag is when compared against the configured alerting threshold and suspension threshold values. Also, for Bet Radar feed, REST response separately reports processing statuses of in-play and pre-match betting program objects.

In case that REST response indicates that the configured alerting threshold value has been breached, an appropriate alert is raised by the Feed Monitor service by generating an appropriate notification request message in the *rma.notification-requests* Kafka topic.

In case that REST response indicates that the configured suspension threshold value has been breached, a suspension of the affected events is performed by the Feed Monitor service across brand betting programs.

To suspend betting on the affected events, Feed Monitor queries the Master Offer for the events that are open for betting and then overrides their status to performs the suspension. In order to be able to perform the unsuspension upon feed processing being restored, Feed Monitor persists identifiers of all events on which it previously performed the suspension in its local database (*dsmonitordb*).

By this mechanism traders are alerted, and bettors are prevented from taking the possible advantage of price changes lagging behind the feed provided data.

Technical implementation details

In actual technical implementation Feed Adapter and Brand Mapper services are realized through a single component that sports two-fold behaviour: it can operate as a Feed Adapter service for a configured target feed, and/or it can operate as a Brand Mapper service. The function and scope of the service instance is, thus, determined by its start-up flags.

Based on the input feed, two separate e3f components have been developed:

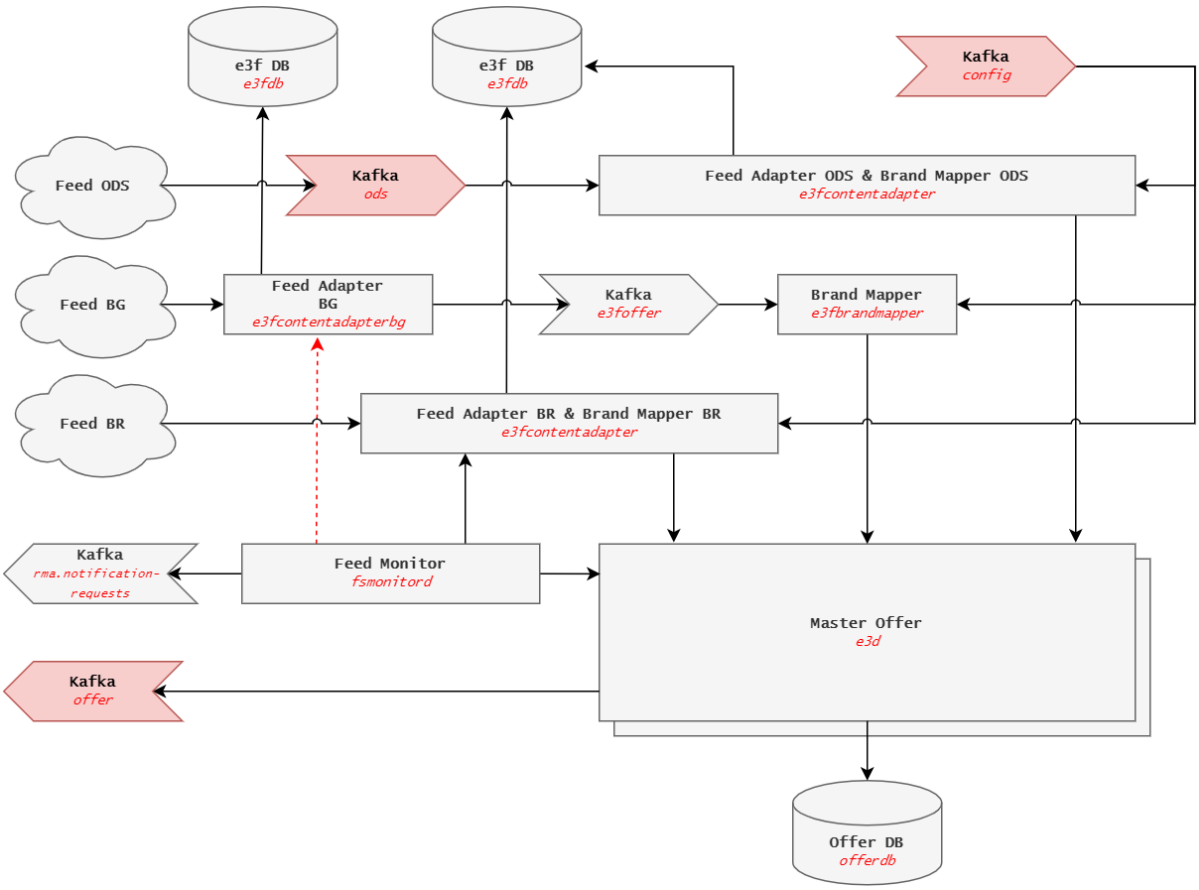
- one which can operate as a Feed Adapter BR for Bet Radar feed; and
- other which can operate as a Feed Adapter BG for Bet Genius feed.

As said, data format in which Feed Adapter service outputs betting offer objects is named according to the physical name of the Feed Adapter component itself (*e3f*) - the “e3f data-model”.

Having this in mind, physical implementation contains:

- Feed Adapter BR and Brand Mapper BR services which are realized through one single service instance named *e3fcontentadapter*.
- Feed Adapter BG and Brand Mapper BG which are realized as two separately configured service instances – one running in feed adapter mode against the Bet Genius feed named *e3fcontentadapterbg*, and the other running in brand mapper mode named *e3fbrandmapper*. These communicate via an interposed *e3offer* Kafka topic group (*e3fcompetitions*, *e3fvenues*, *e3fevents*, *e3fmarkets*) which carries betting offer objects in e3f data-model format.

- Feed Adapter ODS and Brand Mapper ODS service which are realized through one single service instance named *e3fcontentadapter* which uses Kafka topics from the *ods* group as its input source.



Betting Offer block physical diagram

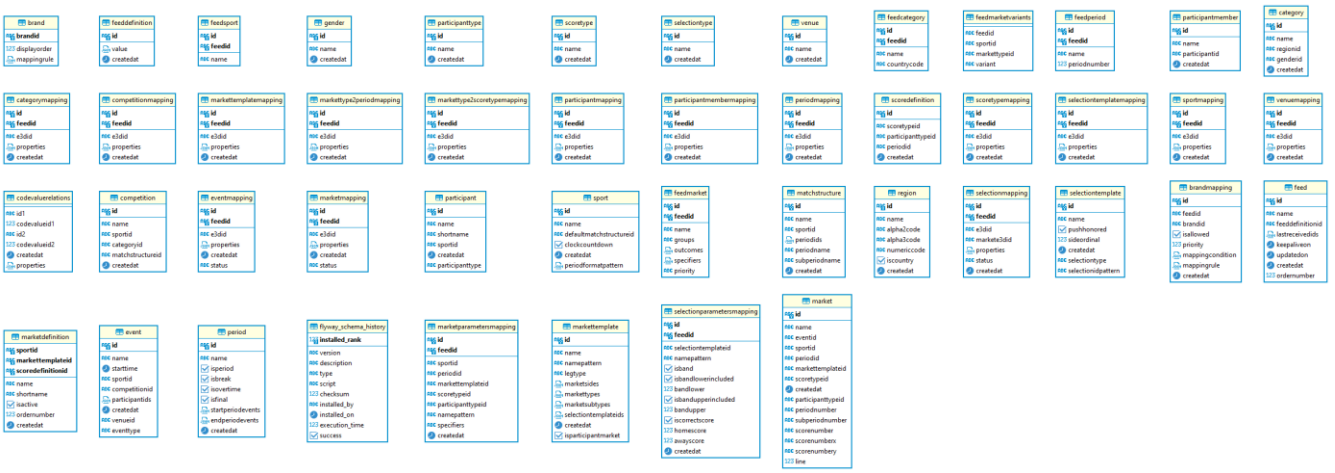
Master Offer component is named *e3d*. It hosts a set of REST endpoints that, amongst others, Brand Mapper service instances use to push feed provided betting offers and their updates.

As all operations against the Master Offer are executed over the same set of REST endpoints, it was adopted to use a separate content-type value in the REST requests in order to identify the data-model which is used by the initiating component. When Brand Mapper service uses these endpoints, it provides betting offer object data in the multi-brand data-model. Other system components use different, more suitable data-models when using these endpoints (more on that later).

Feed Monitor service (*fsmonitord*) currently performs the suspension of all bettable events based only on Feed Adapter BR service lagging behind the feed provided data which exposes this information to external consumers via a REST. Fincore intends to implement monitoring REST on the Feed Adapter BG variant and expand the Feed Monitor accordingly so that particular feed trouble leads to suspensions of events that also originated from that feed.

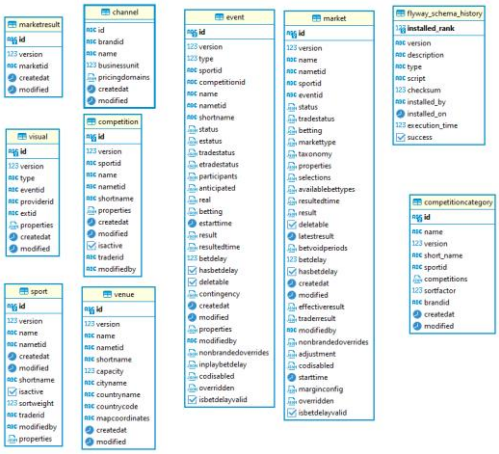
e3fdb schema

Overview of tables within the *e3fdb* schema is given on the embedded picture.



offerdb schema

Overview of tables within the *offerdb* schema is given on the embedded picture.



Further documentation

Following embedded documents provide additional info on Betting Offer block.

E3D Data Models

Feed Status Monitor Internals

E3F Content Adapter Internals

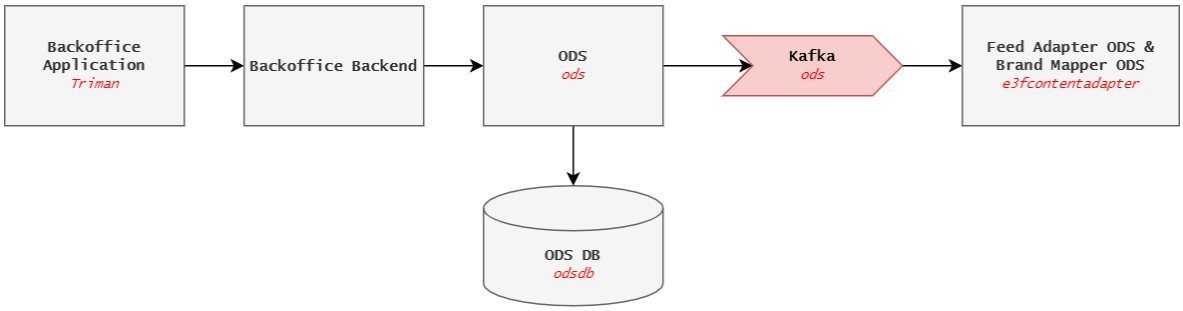
Custom Offer block

Custom Offer block contains components that are responsible for managing the lifecycle of custom offer objects. By using Backoffice Application, users can create, maintain, and result their own events and markets.

ODS

Main component in this block is the ODS service. ODS acronym, which is used here, is a short of Own Data Service and is given as the Custom Offer block presents itself to the rest of the TRI Sportsbook system as yet another data-feed.

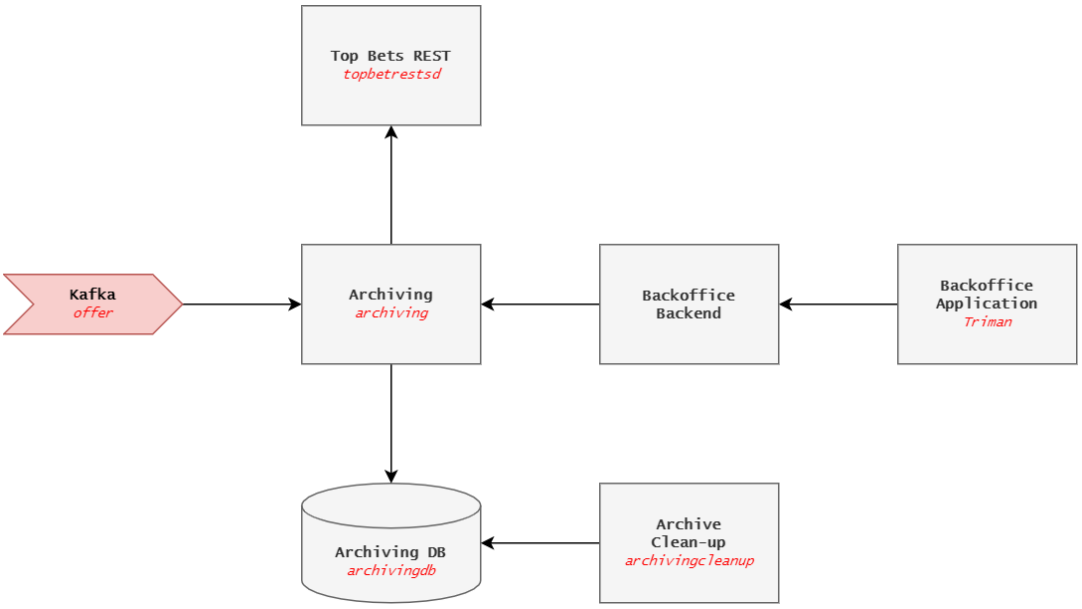
ODS service exposes RESTs that Backoffice Application uses over Backoffice Backend API layer to provide management functionalities over custom offer objects. Once status of custom betting program object changes from the initial one in which custom object is not being published (i.e., only visible through Custom Program screens of Backoffice Application), all further changes applied to it are being automatically pushed towards Feed Adapter/Brand Mapper via the interposed Kafka topics from the *ods* group.



Custom Offer block logical diagram

Archiving Offer block

Archiving Offer block contains components that are responsible for maintaining and clean-up of the betting offer archive containing latest versions of betting program objects and their changelogs.



Archiving Offer block logical diagram

Archiving

Main component in this block is the Archiving service. This service constantly tracks influx of new and changed betting offer objects that is coming through topics from the Kafka *offer* group. On each new betting offer object version that comes through this queue, Archiving service performs the following duties:

- Compares previous version of the target object (taken from the Archiving DB) against the new one having just arrived and creates branded changelog records in the database in case of detecting changes incurred over tracked parameters of the object (e.g., changes of object status under a brand, its overrides, its prices, etc.).
- Queries Top Bets component by using exposed Top Bets REST to establish for which brands placed bets exist which reference the target betting offer object.
- Establishes additional betting offer object information required for filtering (i.e., under which brand target object was overridden).
- Replaces (or inserts - in case of new objects) old object version with this new one within the Archiving DB so to preserve the object in its latest incarnation.

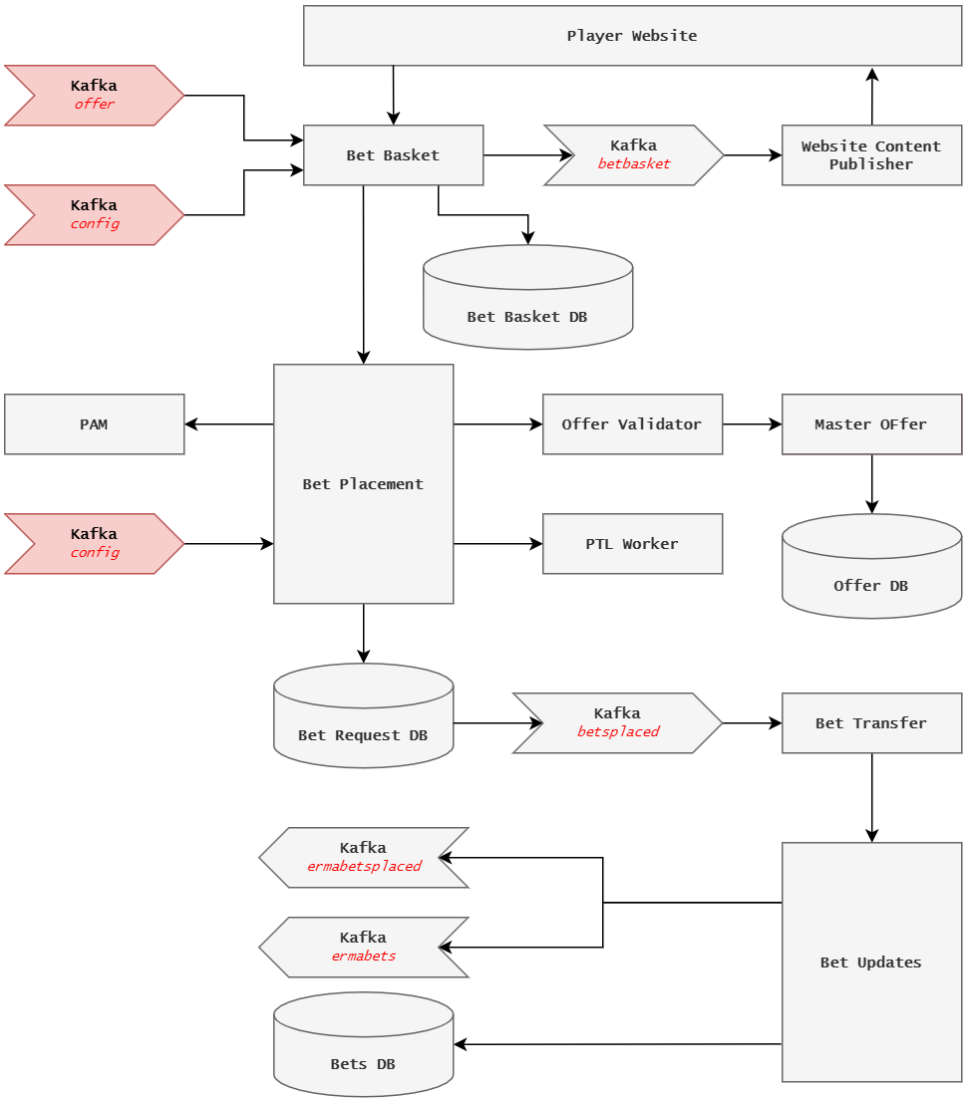
Archiving service additionally exposes RESTs which are being used by Live EMA service and Backoffice Application over the interposed Backoffice Backend in order to filter and fetch data for archived betting program objects and its related changelogs.

Archive Clean-up

Archive Clean-up service periodically sifts through the Archive DB and purges changelogs for all expired betting program objects having never been overridden under any of the brands and having no bets placed on them.

Bet Placement block

Bet Placement block contains components that are responsible for bet basket management and orchestration of bet placement related processes. Components from this block, thus, enable bet basket creation and maintenance, are responsible for enforcing various related rules (such as in-play bet delay or prevention of placement of bets having inter-related legs), enforce bet validation against up-to-date betting offer, configurations, and overrides and enable bet placement.



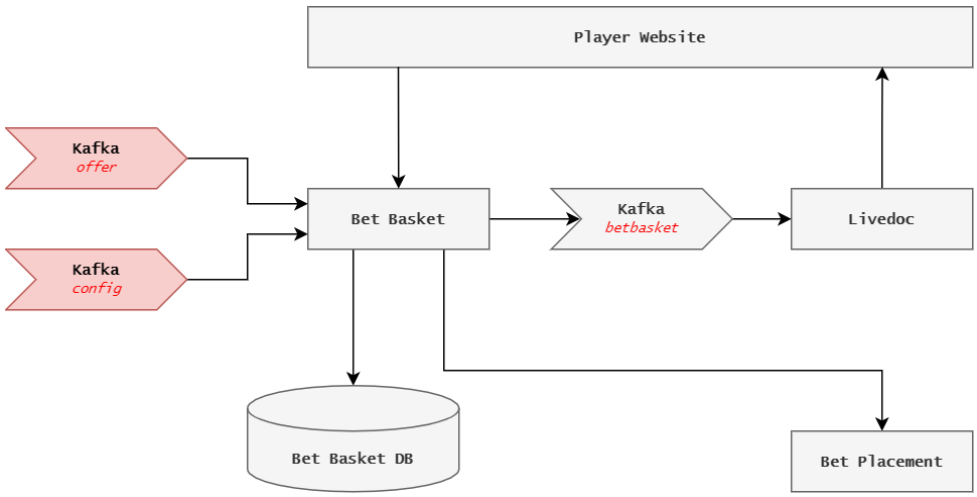
Logical diagram of Bet Placement block

Bet Basket

Bet Basket service is responsible for performing two main duties:

- creating and maintaining bet baskets; and
- initiating bet placements and reporting the outcome.

Bet basket can be defined as a virtual “container” for selections that the player intends to bet on.



Logical diagram of Bet Basket and associated components

Creating and maintaining bet baskets

There are two types of bet baskets that are being handled here:

- anonymous bet baskets, which are assigned to unknown players upon each Player Website visit; and
- personal bet baskets, which are assigned to known players once they log-in onto the Player Website.

What this means is that each player that navigates to the Player Website is being assigned with a new anonymous bet basket which he can use to pick betting offer selections he’s interested in betting on and review possible/allowed bet types. Once player logs-in, he is re-assigned to his personal bet basket which is being merged with the anonymous bet basket he used up to that point. One player can have only one personal bet basket at any given time.

Each action performed by a player against the bet basket on Player Website is communicated to the Bet Basket via exposed REST endpoints. This includes actions such as adding selections to bet basket, removing selections from bet basket, putting stake amounts on offered bet types, etc.

Additionally, Bet Basket service listens to real-time changes of related betting offer objects, configurations and overrides by following relevant Kafka topics.

Upon each change that can possibly affect bet basket content – be it a player action, betting offer object change or a configuration change - Bet Basket automatically computes a new bet basket version (i.e., new changed state of the bet basket) and pushes it towards Player Website through

Kafka and Website Content Publisher. These changes are then used to accordingly render the updated bet basket on-screen.

When computing a new bet basket version, Bet Basket performs the following:

- Disables or enables selections placed in the basket based on up-to-date betting offer statuses (e.g., disables a selection if betting gets suspended on the parent market/event).
- Establishes inter-relations between enabled selections based on up-to-date inter-relations configuration.
- Resolves possible and allowed bet types by taking into account enabled selections and established inter-relations between them based on up-to-date IR configuration.
- Calculates bet rates and numbers of units for each of the allowed bet types.

In addition to serving bet basket updates to the Player Website, Bet Basket also persists latest versions of baskets in the local database, both anonymous and personal.

Initiating bet placements and reporting the outcome

Second feature that Bet Basket service takes care of is initiating bet placement and reporting the outcome via a subsequent bet basket version update.

In this sense, when logged-in player places some stake onto one or more offered bet types listed in the basket and hits the appropriate Player Website button to request bet placement, Bet Basket service does the following:

- Resolves applicable in-play bet delay periods for each of the bet types backed by the entered stake amounts.
- Applies the resolved in-play bet delay by waiting for the longest resolved time period.
- Calls Bet Placement service to request separate bet placements for each of the bet types backed by the entered stake amounts.
- Waits for responses from the Bet Placement service to arrive for each of the tried bet types and then generates new bet basket version carrying placement outcomes for each of the tried bet types.

In case it is concluded from configuration and overrides that no bet delay should be applied to either of bet types backed by entered stake amounts (e.g., no event is currently live, or live-leg count exceeds the configured wavering limit), Bet Placement service will immediately proceed with placement.

Once a bet placement request is made towards the Bet Placement service for each bet type backed by an entered stake amount, Bet Basket service waits to receive responses for all tried bet types prior to reporting the total outcomes through a newly generated bet basket version.

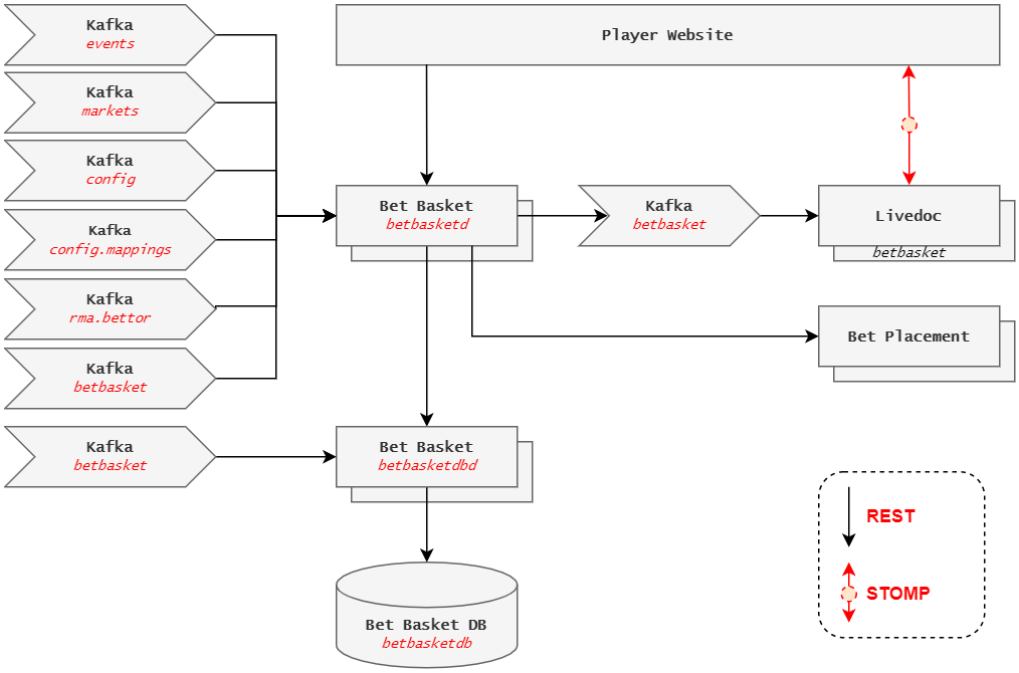
Technical implementation details

In actual technical implementation overall Bet Basket duties are delegated across two separate components:

- *betbasketd* service: This service is responsible for performing all other duties listed above. It exposes REST API providing methods for bet basket management and bet placement

initiation which are used by the Player Website. Uses *betbasketdbd* REST endpoint to fetch unknown bet baskets from the database and relies on Kafkas for remaining data.

- *betbasketdbd* service: This service is responsible for persisting latest bet basket versions in the database. To this end it exposes an appropriate REST API used by *betbasketd* and listens to bet basket updates passing through the Kafka *betbasket* topic so to perform database updates accordingly.



Physical diagram of Bet Basket and associated components

betbasketd service listens to following Kafka topics:

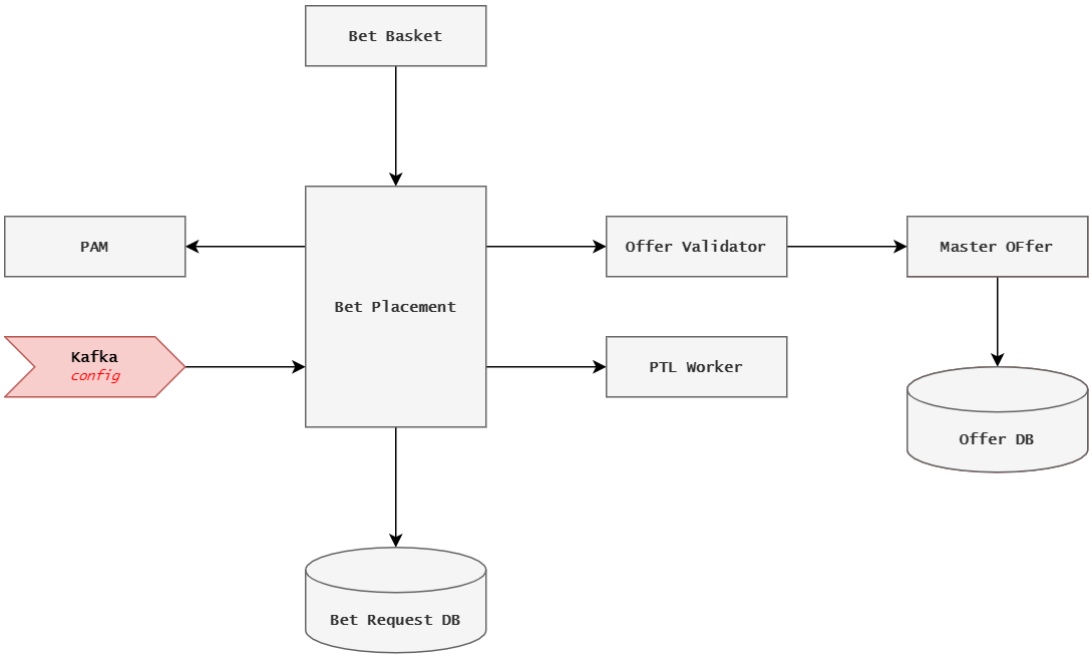
- *events* and *markets* to be up-to-date with changes happening on related betting offer objects,
- *config* and *config.mappings* to be up-to-date with changes happening on in-play bet delay configurations,
- *betbasket* for synchronization across other *betbasketd* instances, and
- *rma.bettor* to be up-to-date with changes happening to related player in-play bet delay overrides.

As said, upon each change that can possibly affect the bet basket content, *betbasketd* service computes a new bet basket version and pushes it via Kafka *betbasket* topic. This topic is read by the Website Content Publisher which pushes a new *betbasket* LiveDoc message carrying the updated bet basket to the Player Website.

Both *betbasketd* and *betbasketdbd* services can be horizontally scaled to cater for increased load of serving greater number of bet baskets simultaneously.

Bet Placement

Bet Placement service enables placing of the bet requests and is responsible for orchestrating the placement flow. It is made flexible so can be adapted to take different actions during placement, or to execute them in different order.



Logical diagram of components involved in bet request processing

Upon receiving a fresh bet placement request from the Bet Basket service, Bet Placement performs following actions:

- Performs validation of the bet request by using Offer Validator service. Based on the outcome of these validation Bet Placement either denies the bet placement request reporting this outcome back to the Bet Basket service or continues with the bet placement process.
- Performs PTL check for the bet request by using PTL Worker. Based on the outcome of the PTL check, bet request can be either approved, rejected, or reoffered. In case bet request has been rejected or reoffered, Bet Placement reports the appropriate outcome back to the Bet Basket service. In case of bet request being approved, Bet Placement continues with the bet placement process.
- Communicates with PAM to perform funds reservation and then confirmation or cancelation depending on the final outcome of the request.
- Performs other enforced validations of the bet request including check-up of inter-relations as carried by the bet against current configuration.
- Records received bet request in the Bet Request database and updates its state in database accordingly as bet request progresses through said placement stages.

In case the fresh bet placement request yields a reoffer outcome (offer validations yielded approvals, but PTL check reported bet being acceptable if having a diminished stake), this outcome together with acceptable reoffered stake amount and reoffer validity period duration is reported back to the Bet Basket service and then presented on the Player Website. Should the

player accept this reoffer within the said reoffer validity period, thus effectively lowering the stake placed on the tried bet to be acceptable for the PTL check, another repeated bet placement request with this diminished stake is sent by the Bet Basket service. This repeated bet request does not necessarily undergo all stages listed for the fresh bet requests above. For instance, and based on secondary validation configuration, repeated bet request may or may not undergo validations provided by the Offer Validation service. Also, in case that this repeated bet request arrived at the Bet Placement service within the defined reoffer validity period, secondary PTL check is generally not performed.

As said, which of the stated actions are to be enforced, and which are to be skipped can be configured. Additionally, the order in which Bet Placement service performs enforced actions can also be configured.

Offer Validator

When asked by the Bet Placement, Offer Validator service retrieves up-to-date relevant betting offer objects from the Master Offer and, by comparing them against betting offer objects carried by the bet request, performs following validations:

- Status validation: It checks whether all relevant betting offer objects are “bettable”, meaning all events, markets and selections referenced by the bet request have appropriate statuses and are currently within their defined betting periods.
- Price validation: It compares relevant betting offer object prices against those carried by the bet request to validate no prices have been changed.

In case of status offer validation failure, Bet Placement denies the request.

In case of price validation failure, up-to-date prices are reported back to the Bet Placement service which then compares them against the prices carried by the bet request. Based on the up-to-date price change tolerance configuration taken from the Kafka *config* topic, Bet Placement service then makes one of following decisions:

- To continue with placement taking better prices into account: This happens in case all price changes are within the configured tolerance.
- To deny the bet placement request: This happens in case at least one price change is outside the configured tolerance, in which case Bet Placement reports prices being changed back to the Bet Basket service.

PTL Worker

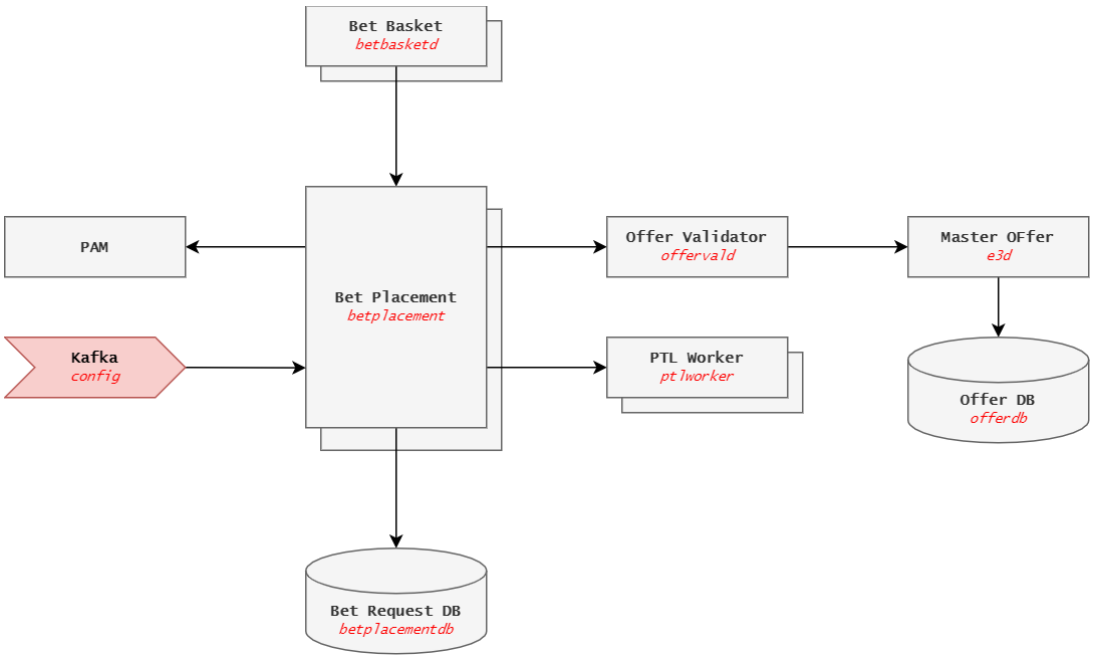
PTL Worker performs configured risk rule checks for the tried bet and reports back the outcome, being one of the following: accepted, rejected, or reoffered. It will be discussed in more details in further sections related to automatic risk control components.

Technical implementation details

In actual technical implementation Bet Placement service (*betplacement*) provides a REST endpoint which is used by the Bet Basket service (*betbasketd*) to submit fresh and retried bet requests and receive outcomes for these requests.

It uses REST endpoints exposed by the Offer Validator service (*offervald*) and PTL Worker (*ptlworker*) to perform check-actions against the tried bet request and reads Kafka *config* topics to have an up-to-date price tolerance configuration.

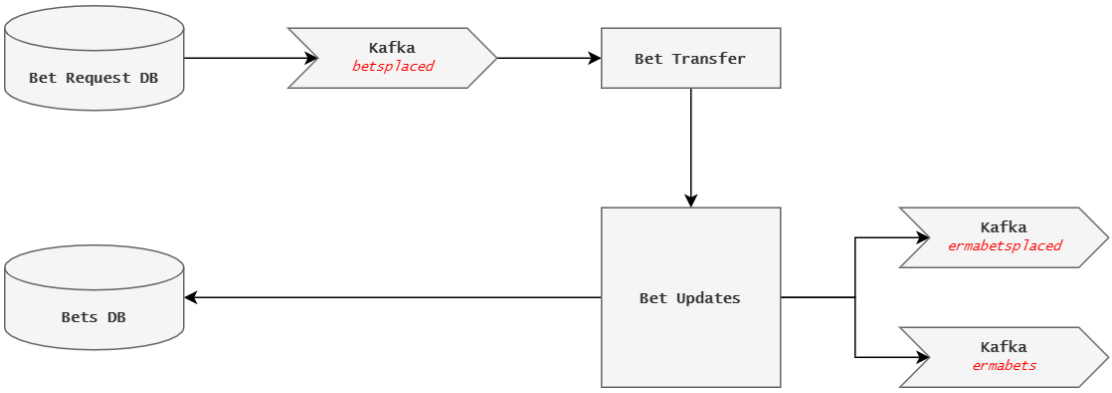
Bet Placement service can be scaled horizontally to cater for increased load of serving greater number of bet baskets simultaneously.



Physical diagram of components involved in bet request processing

Bet Updates

Bet Updates service is responsible for maintaining placed bets in the Bets DB and for providing bet data to interested consumer components.



Logical diagram of components involved in bet maintenance

From the business standpoint, functionalities that Bet Updates service provides are the following:

- Creation of new bets based on approved bet requests
- Updates of bets based on manual operations performed by Backoffice Application users
- Bet search features required by interested consumer components like Backoffice Application Backend to feed bet list and bet details page, etc.
- Providing bet data to interested consumer components via Kafka *ermabetsplaced* and *ermabets* topics.

Regarding said Kafka topics, Bet Updates service performs the following:

- Upon creation of a new bet, Bet Updates service pushes this new bet into the Kafka *ermabetsplaced* topic. This topic is then used by consumer components that require information on placed bets (e.g., Backoffice Content Publisher which uses this data to feed bet tickers in the Backoffice Application, Top Bets service which uses this data to maintain branded lists of selections having most placed bets, etc.).
- Upon creation of the bet and any further updates performed on it, Bet Updates service feeds Kafka *ermabets* topic with messages containing a dataset that describes bet change relative to its previous version (e.g., a set of totals, relative value changes from the previous bet version and various required flags). This topic is used by consumer components that require information on relative bet changes (e.g., Aggregation Worker service to update various aggregations needed for automatic risk control processing, Fieldbook Worker service to maintain market fieldbooks, Bet Transaction Generator service to generate transaction records, etc.).

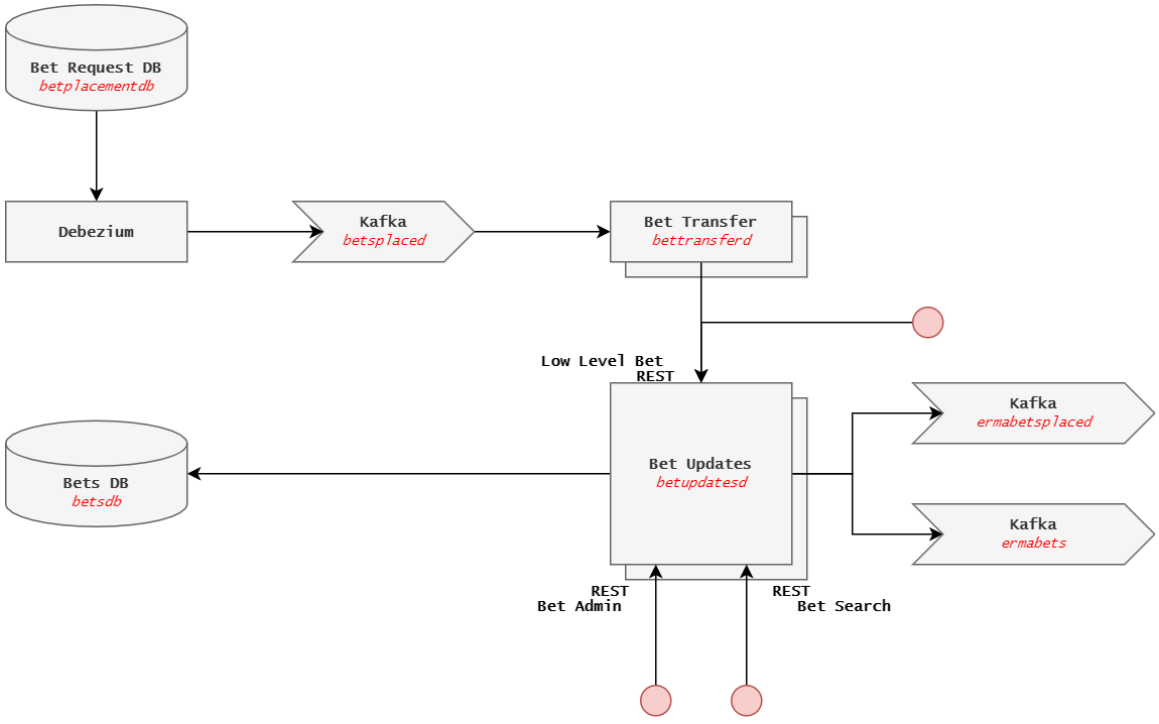
Bet Transfer

Creation of new bets in the Bets DB by the Bet Updates service is triggered by requests made by Bet Transfer service. Bet Transfer service picks up all bet requests that end up in the state indicating the request has been validated and approved for becoming a bet. This is done by Bet Transfer service listening to the Kafka *betsplaced* topic into which all validated and approved bet requests are being pushed by Debezium which does the actual monitoring of changes happening on requests in the Bet Request DB. Once a new message arrives in the said Kafka topic, Bet Transfer service requests Bet Updates service to create a new placed bet.

Technical implementation details

In actual technical implementation Bet Updates service (*betupdatesd*) provides three REST endpoints:

- Low Level Bet REST: This endpoint is used by the Bet Transfer Service (*bettransferd*) to request new bet to be created in the Bets DB. It also presents an integration point for connecting various third-party systems (e.g., to insert or migrate bets from other sportsbooks). Bet Updates service does not perform any validations/checks on bet placement request received through this REST.
- Bet Admin REST: This endpoint is used by the Backoffice Application Backend to perform bet updates based on manual Backoffice user actions.
- Bet Search REST: This endpoint is used by the Backoffice Application Backend to execute various searches and thus fetch target bet data.



Physical diagram of components involved in bet maintenance

Bet Updates and Bet Transfer services can be independently scaled horizontally to cater for increased load of serving a greater number of requests simultaneously.

Further documentation

Following embedded documents provide additional info on Bet Placement block.


Bet Basket Internals

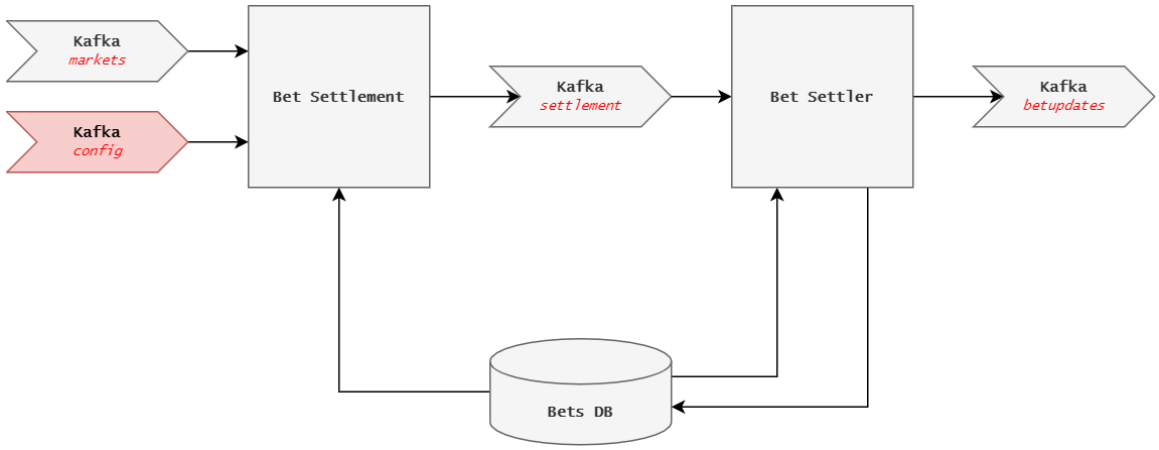

Bet Placement Internals


Offer Validator Internals


Bet Updates Kafka Data-model

Bet Settlement block

Bet Settlement block is tasked with performing settlement operations on bets based on results that arrive for markets backing those legs. Block consists of two main components being Bet Settlement service and Bet Settler service that are interconnected by Kafka queues. Such two-tiered approach is considered as important for scalability and achieving high bet settlement volumes as processing is distributed in such a way that Bet Settlement service processes market results and enforces settlement prioritization, while Bet Settler then processes actual bets.



Betting Settlement block logical diagram

Bet Settlement

Main responsibilities of the Bet Settlement service are the triage of market update messages and bet settlement scheduling.

Service performs the triage by listening to Kafka *markets* topic and filtering out market update messages that carry new effective market results for at least one of the brands configured in the system. Market update messages that provide no result changes are considered to be of no interest in settlement processes and are, thus, further ignored by the service.

As market results are brand specific, for messages that carry new market results Bet Settlement service first establishes the scope for each new market result by determining the brand(s) for which new results arrived.

It then starts with the settlement scheduling by fetching the identifiers of all bets having legs on the same market for one of the determined brands. This fetching is done according to the configured bet settlement strategy which is brand specific and can be set to prioritize between settling:

- potential winning bets having fewer open legs first,
- potential losing bets having fewer open legs first,
- potential losing multiple bets, or
- all bets equally

By default, equal settlement priority is assumed for all brands.

Once the IDs of all related bets for the market result under one brand have been fetched, Bet Settlement service pushes them together with the new market result into one of up to three settlement queues being Kafka *settlement* topics, namely highest, high, and normal priority queues. This is also done based on the configured settlement strategy for that brand.

Service then repeats the settlement scheduling process for all other brand results, their related bets and based on relevant configured settlement strategy.

As bet settlement strategy configuration can change at any time, Bet Settlement service additionally listens to Kafka *config* topic through which these configuration changes are distributed. Once message arrives carrying configuration change, service adapts by enforcing new strategy to upcoming market results.

Bet Settler

Bet Settler service performs the actual bet settlement by consuming data from said Kafka *settlement* topics and fetching, processing, and updating target bets in the Bets DB accordingly.

To enforce settlement prioritization, Bet Settler service will continuously consume unread messages from highest priority *settlement* topic until it depletes them all from the said queue. Only when no new messages remain in highest priority *settlement* topic, Bet Settler service will turn to consume messages from high priority *settlement* topic. Same principle is then applied for switching to consuming from high and normal priority *settlement* topics. In case any higher priority bets arrive, Bet Settler service will switch to consume those from the related queue.

By doing so, Bet Settler provides that all highest priority bets will get settled before any of high or normal priority ones.

In order to perform the actual settlement, Bet Settler first fetches the target bet from the Bet database. Service then changes the outcome of the relevant bet leg based on market results as carried by the *settlement* topic message, calculates new bet values like potential return and, in case all bet legs have been settled or potential return of the bet has dropped to zero, changes the status of the bet to “settled”. In similar fashion, and when new market result dictates so, it performs bet unsettlements and resettlements by changing leg outcomes and bet status accordingly.

Once done processing the bet, it finalizes the settlement by pushing the updated bet to the database and produces a bet update message in the relevant Kafka *betupdates* topic carrying newly calculated bet flags, totals and delta amounts.

Bet Settler service supports settling bets based of several market result formats called “result classes”, such as:

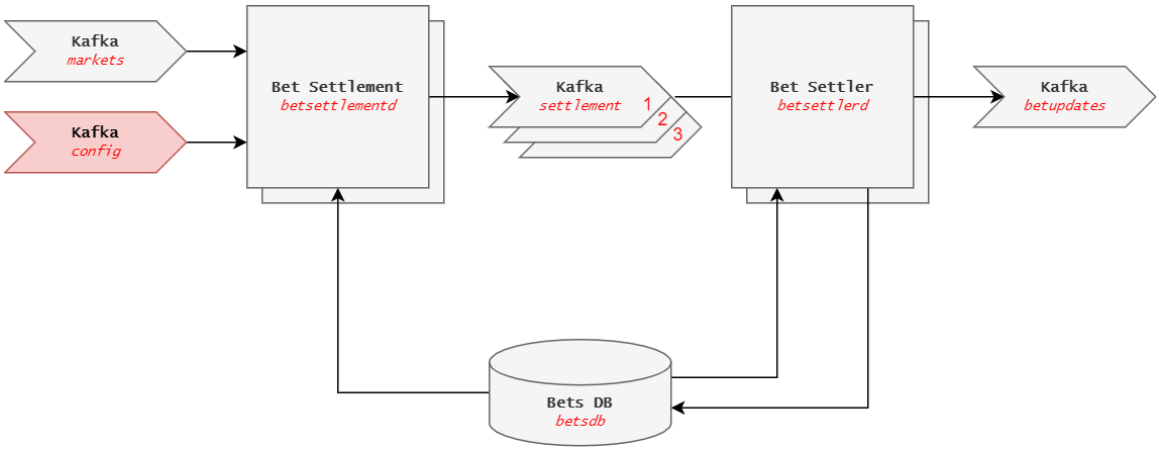
- winning selection, where an explicit outcome (e.g., “WON”, “LOST” etc.) is attached to a market selection in the market result,
- make-up value, where market result is defined as the difference or sum of scores of home and away event participants (e.g., “2” in football match, meaning home team scored two goals more than the away team), in this sense make-up value can represent home score - away score or home score + away score,
- finishing order, where market result is defined by the participants placement ranking (e.g., in racing events participant backing first selection placed third, participant backing second selection placed seventh, etc.),
- correct score, where market result is defined through separate scores for home and away event participants (e.g., “3:1” in football match, meaning three goals scored by home team and one goal scored by the away team).

In case of Bet Settler being encountering an error while trying to settle a bet (e.g., unable to settle it based on provided resulting information, or cannot save changes, etc.), it will save the bet identifier and the error message into the “failed” database table.

Technical implementation details

In actual technical implementation Bet Settlement service (*betsettlementd*) and Bet Settler service (*betsettlerd*) can be scaled horizontally through addition of new service instances and vertically through addition of new workers into existing service instance.

Bet Settlement service is constituted from two sets of workers: one set of workers performing reading of bets from the database, and the second set of workers producing Kafka topic messages. Number of workers of the former set can be scaled through command line, while number of workers in the latter set can be scaled via REST API calls.



Betting Settlement block physical diagram

All active Bet Settlement/Bet Settler service instances are tuned to automatically balance the bet processing load between them.

In case of all brands using equal settlement priority strategy, only one Kafka *settlement* topic is required. In case of using other settlement strategies, additional Kafka *settlement* topics are needed so to provide for bet segmentation according to configured settlement prioritization policy.

Further documentation

Following embedded documents provide additional info on Bet Settlement block.

Bet Settlement Internals

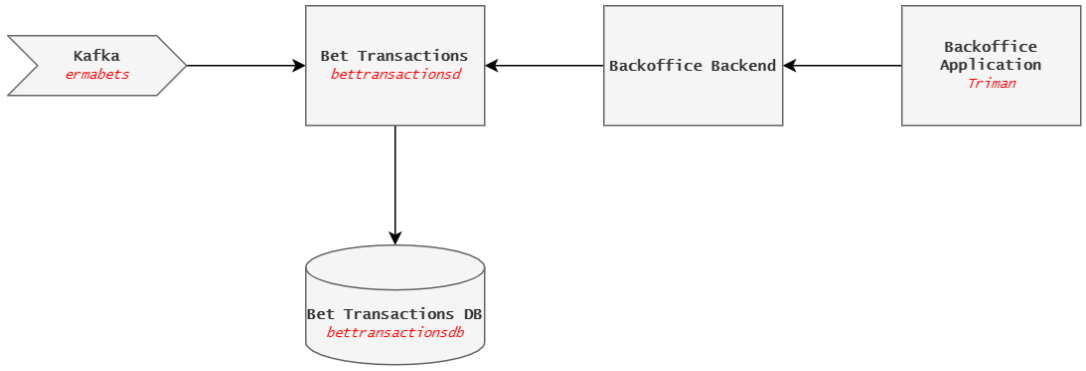
Bet Transactions block

Bet Transactions block contains components that are responsible for maintaining bet transaction records. According to adopted policy to reduce number of records generated, only bet changes that produce a monetary posting against player's wallet are ones being recorded as "bet transactions". In other words, no bet transactions will be recorded for any intermediary bet changes which are caused by partial bet settlement operations.

Bet Transactions

Bet Transaction component listens to changes happening on placed bets, coming through *ermabets* Kafka topic. Upon receiving a new message, Bet Transaction service performs the evaluation of the change and in case it concludes a monetary posting against player's wallet will happen, it determines the type of the transaction and inserts it in the Bet Transactions DB.

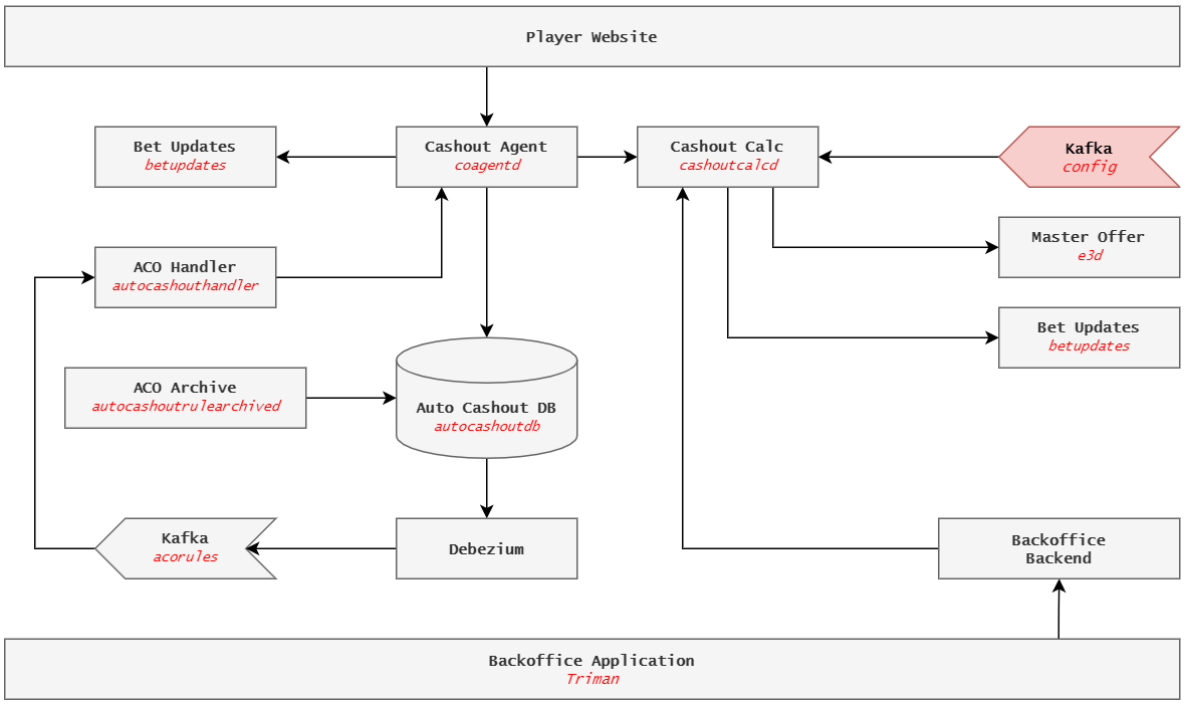
Bet transactions are made visible in Backoffice Application through summary or individual transaction views and, to this end, Backoffice Application over interposed Backoffice Backend APIs uses RESTs exposed by the Bet Transactions service to filter and fetch transaction data.



Bet Transactions block logical diagram

Bet Cashout block

Bet Cashout block contains services that provide on-demand bet value and bet cashout offer calculations intended for display on Backoffice Application and Player Website (former only), accepting and processing cashout requests and auto cashout rules. Through these inter-workings TRI Sportsbook provides full cashout, partial cashout and auto cashout functionalities.



Logical diagram of Bet Cashout block

Cashout Calc

This service provides REST endpoint for on-demand providing bet value and bet cashout offer amounts for the target bet.

Bet value is the amount of return that may be paid for the entire remainder of open bet stake taking into consideration current odds/prices of all bet related selections. Bet cashout offer, on the other hand, is calculated as bet value amount decreased for a certain commissioning percentage (i.e., bookie's keep) and is one actually offered to the player on website should he accept full cashout.

Upon being asked to by either Auto Cashout Agent or Backoffice Application (over interposed Backoffice Backend), Cashout Calc performs cashout eligibility checks over the current state of the bet and current state of all related betting offer objects, enforces current configuration that defines whether cashout is allowed for the player and betting program objects, and calculates and provides said required amounts.

In order to perform said task, Cashout Calc constantly tracks changes to cashout related configuration by listening to topics from Kafka *config* group, and, when asked for a certain bet,

collects current bet state from Bet Updates service and related betting offer states from Master Offer service.

Described mechanism is also employed within the Livedoc service that feeds Player Website with cashout offers for player's open bets, and within Auto Cashout Handler service to dynamically react on changes affecting bets targeted by active rules and their related betting offer objects in order to check whether rule conditions have been met.

Cashout Agent

Cashout Agent provides top-layer APIs which are used by Player Website to pass partial/full cashout requests and create/delete auto cashout rule requests and performs orchestration of actual cashout executions upon request.

Player logged-onto Player Website when observing his open bets has following abilities:

- to request partial and full cashout over bets which are considered eligible for cashout and bounded by limits of cashout offers for such bets (i.e., calculated amount of money that player can be reimbursed with should he accept to cashout the bet in full).
- to request creation or deletion of auto cashout rules for his open bets through which he defines for him acceptable cashout triggering threshold and amount of return he wishes to be reimbursed with.

Such requests are, upon player action done on website, propagated to the Cashout Agent via mentioned API calls.

Upon receiving full or partial cashout request, Cashout Agent performs auth check to verify request is related to actual player's bet, enforces in-play bet placement delays when needed (according to configuration), and validates bet eligibility and current cashout offer against actual conditions on bet and related betting program objects. If all pans out, Cashout Agent attempts to perform the actual cashout by requesting it from Bet Updates service. Eventual request outcome is reported back to Player Website to inform the player.

Upon receiving request to create or delete auto cashout rule for the bet, Cashout Agent performs auth check to verify request is related to actual player's bet and examines current statuses of rules that possibly exist for same bet. In case of a request to create auto cashout rule for the bet, Cashout Agent allows rule creation only if no other active rule exists in the database for that same bet. In case of a request to delete auto cashout rule, Cashout Agent allows target rule deletion only if active rule for same bet exists in database and has not yet reached cashout execution stage.

Verified, checked, and accepted auto cashout rules are inserted by Cashout Agent into the Auto Cashout DB from which they are pushed towards ACO Handler via Debezium/Kafka for processing.

ACO Handler

As said, Auto Cashout Handler (or ACO Handler for shorts) maintains in memory list of all active auto cashout rules, and processes those upon each change affecting related bets or any of backing betting program objects.

By this virtue, ACO Handler reacts to any said change by performing cashout eligibility check and bet cashout offer recalculation. It uses those to compare against values recorded on the processed rule in order to determine whether conditions carried by the rule have become met. In case it determines that the conditions are met (i.e., current bet cashout offer amount surpasses recorded threshold and bet is eligible for cashout), ACO Handler requests from Cashout Agent to orchestrate the actual cashout execution according to the rule upon first changing the rule status in the Auto Cashout DB so to prevent it getting deleted by the concurrent player's request coming from website.

Based on the outcome of the attempted cashout execution, ACO Handler properly changes rule status in the database so to either retire the rule or to mark it will continue processing it.

ACO Handler also reaches the decision on rules which conditions have become impossible to be met and performs their delayed discarding. Delayed, as it discards rules only after they remain in this-impossible-to-be-met state after a certain time period has lapsed, thus allowing for possible resulting or settlement errors to be corrected.

ACO Rule Archive

Upon bet auto cashout rule becoming discarded or retired and, thus, removed from the memory and vision of ACO Handler, ACO Rule Archive service moves in to transfer such rule into the archive table, thus performing housekeeping of the main tables of the Auto Cashout DB.

Further documentation

Following embedded documents provide additional info on Bet Cashout block.


Cashout Implementation


Cashout Internals


Cashout Frontend Integration Guide

Publishing block

Publishing block contains services that provide real-time push of ever-changing backend data to frontend components. To do so publishing services use pub/sub mechanism to distribute information. Using this mechanism negates the need for frontend components to periodically query backend components to refresh displayed data.

What frontend components subscribe to is called “documents” (e.g., a document in JSON format holding current data of a certain market). For these services to distribute documents to subscribed clients, service is required to collect, store, and maintain related object(s) information.

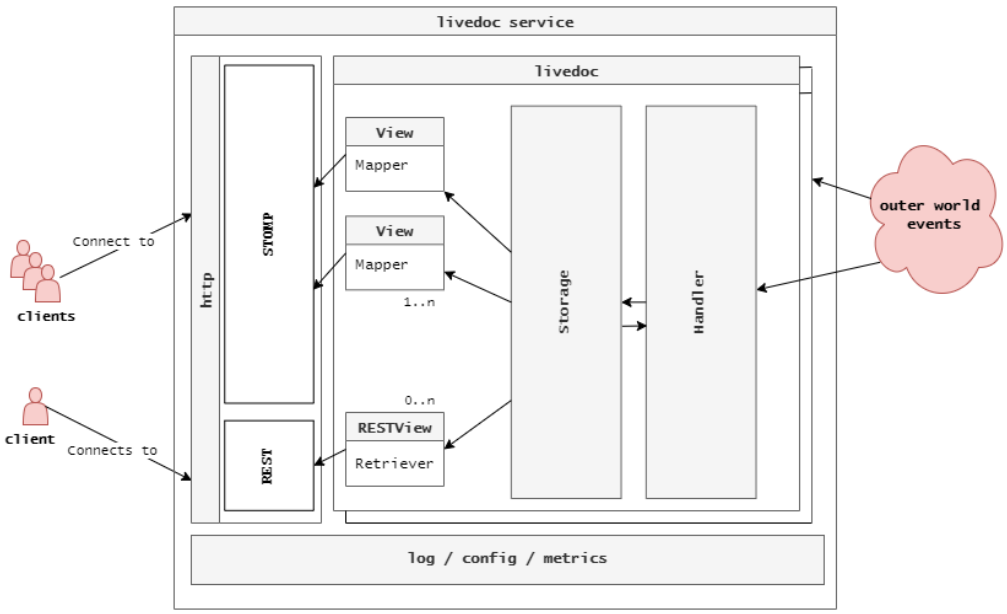
Clients connect to publishing services by using http protocol whereby connection, when established, is upgraded to websocket. Information among server and clients are then exchanged by using STOMP protocol over established websocket.

Two types of publishing services exist:

- public publishing services on which any interested consumer may subscribe to, and
- session publishing services where subscribing component must identify itself upon which documents intended only to her are being distributed by the service.

Examples of public publishing services are events, markets, competitions, sports, etc. Examples of session publishing services are betbasket, balance, cashout, etc.

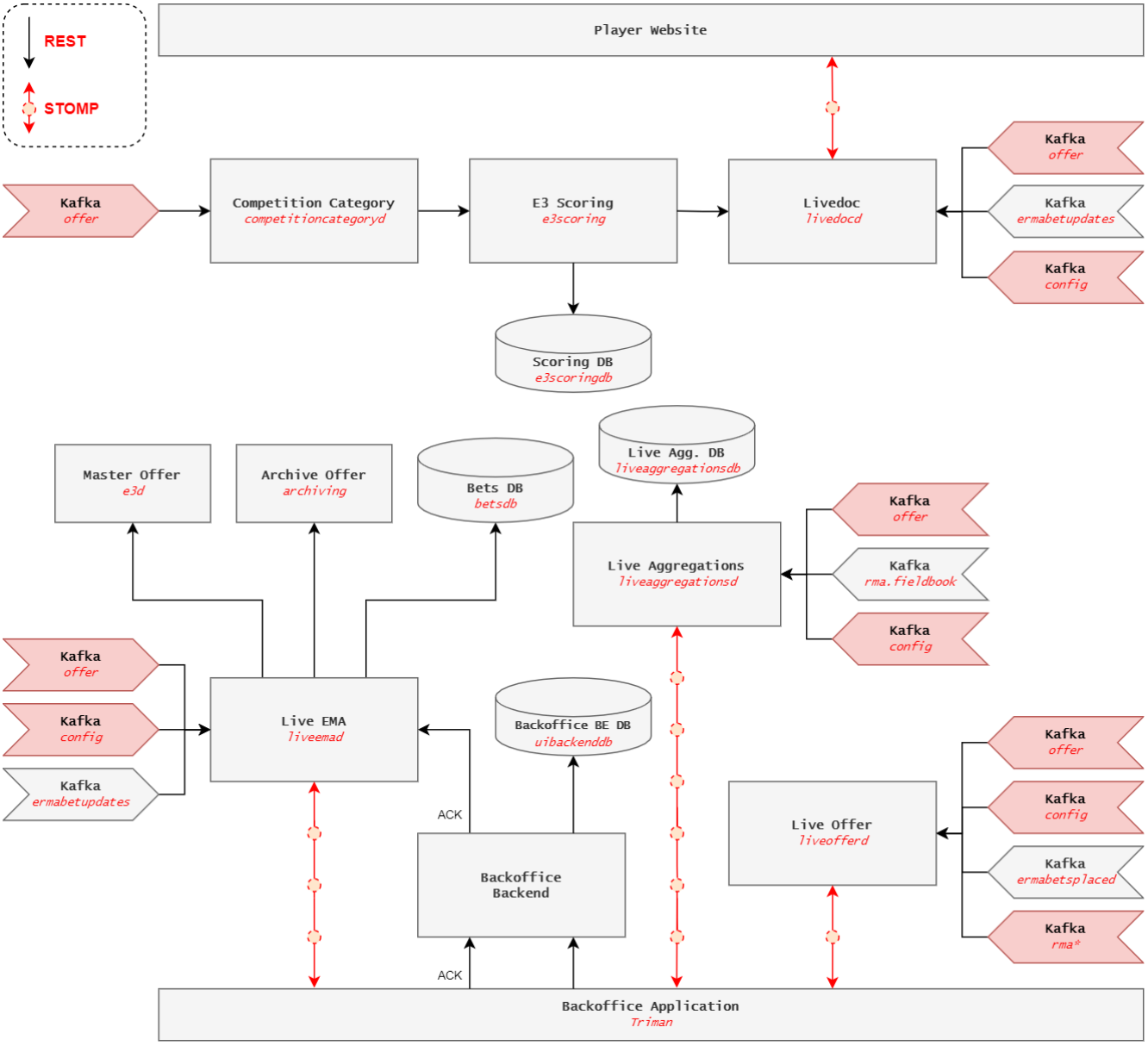
All existing instances of such publishing services are based on the architecture shown in the diagram below and are generally referred as “livedoc(s)” according to the name given to the publishing service that was implemented first.



Publishing (livedoc) service generalized architecture

"Livedoc(s)" moniker is used equally with the term "publishing service(s)" to refer to any of the actual different instances of existing services that utilize said principles.

TRI Sportsbook contains publishing services depicted in the following diagram providing data for either Player Website or Backoffice Application



Publishing services and associated components

Livedoc

This service was historically the first one to be developed employing said publishing mechanism and, thus, it donated its name to the proprietary publishing mechanism employed throughout TRI Sportsbook. This particular service is responsible with supplying data to Player Website.

What it primarily does is it collects, stores, maintains, and distributes information on betting program objects and bets that is required by the Player Website to present and dynamically refresh data. To do so for the betting program objects, it listens to following Kafka topics: *sports*, *competitions*, *events*, and *markets*, which are jointly termed the "*offer*" topic group. For bets and changes on those it listens to *ermabetupdates* Kafka topics.

Additionally, this service listens to Kafka topics from the *config* group for changes made to related configuration as they happen.

Physical name given to this service is *livedocd*.

Live Offer

Live Offer service is responsible with supplying data on betting offer objects, placed bets, PTLs and risk (like liveoffer) to Backoffice Application. To do so it collects, stores, maintains, and distributes information by listening to following Kafka topics:

- Kafka topics from the *offer* group (i.e., *sports*, *competitions*, *events*, and *markets*) – for information on betting program objects,
- *ermabetupdates* Kafka topic – for information on placed bets,
- Kafka topics from *rma** group – for information on PTLs and risk.

This service additionally listens to Kafka topics from the *config* group for changes made to their configuration.

Physical name given to this service is *liveofferd*.

Live aggregations

Liveaggregations service is responsible with supplying data required by Top Selections and Heatmap Backoffice Application widgets. To do so it collects, stores, maintains, and distributes information by listening to following Kafka topics:

- Kafka topics from the *offer* group (i.e., *sports*, *competitions*, *events*, and *markets*) – for information on betting program objects,
- *rma.fieldbook* Kafka topic – for information on market book.

To be able to resume its operation upon restart and retain data, service uses the database titled Live Aggregations DB. This database is used for service priming upon restart prior to resuming listening to Kafka.

This service additionally listens to Kafka topics from the *config* group for changes made to their configuration.

Physical name given to this service is *liveaggregationsd*. Physical name of the database used is *liveaggregationsdb*.

Live EMA

Live EMA service is responsible with supplying data required by Problem Multiples Monitor Backoffice Application widget (formerly known as Enhanced Multiples Awareness – thus the EMA moniker). To do so it collects, stores, maintains, and distributes information by listening to following Kafka topics:

- *events* and *markets* – for information on events and markets,
- *ermabetupdates* – for information on placed bets and changes happening on them.

Since data provided through said queues does not contain all information on betting program objects that is required to be displayed on the Backoffice Application widget, service further communicates via REST with Master Offer and Archive Offer to fetch remainder of the data.

To provide the acknowledgement functionality through which bets shown in the widget may be acknowledged by Backoffice Application users and, thus, removed from the view until changed beyond a configured amount, Live EMA service exposes a REST which is invoked by Backoffice Backend upon user performing bet acknowledgement in widget.

This service additionally listens to Kafka topics from the *config* group for changes made to its configuration.

To be able to resume its operation upon restart and retain data, service data within the Bets DB. This database is used for service priming upon restart prior to resuming listening to Kafka.

Physical name given to this service is *liveemad*.

E3 Scoring

Currently, E3 Scoring service stores information on branded competition categories (a.k.a. jurisdictions) which is collected from Competition Category service which, in turn, uses Kafka topics from *offer* group as it's data-source. Service is intended to, in the future, provide scores based on other collected various statistics, which scores could be used to determine betting program object popularity based on multiple statistical dimensions (e.g., to determine event popularity within the brand based on branded statistics, based on per player preferences, etc.).

In this sense, E3 Scoring service can be said to be responsible with collecting branded statistics related to betting program objects, calculating scores, and providing those to Player Website for data personalization.

To be retain scoring information, service uses Scoring DB

Physical name of E3 Scoring service is *e3scoring*. Physical name of Competition Category service is *competitioncategoryd*.

Further documentation

Following embedded documents provide additional info on implementation of publishing services in TRI Sportsbook.


Writing a Livedoc

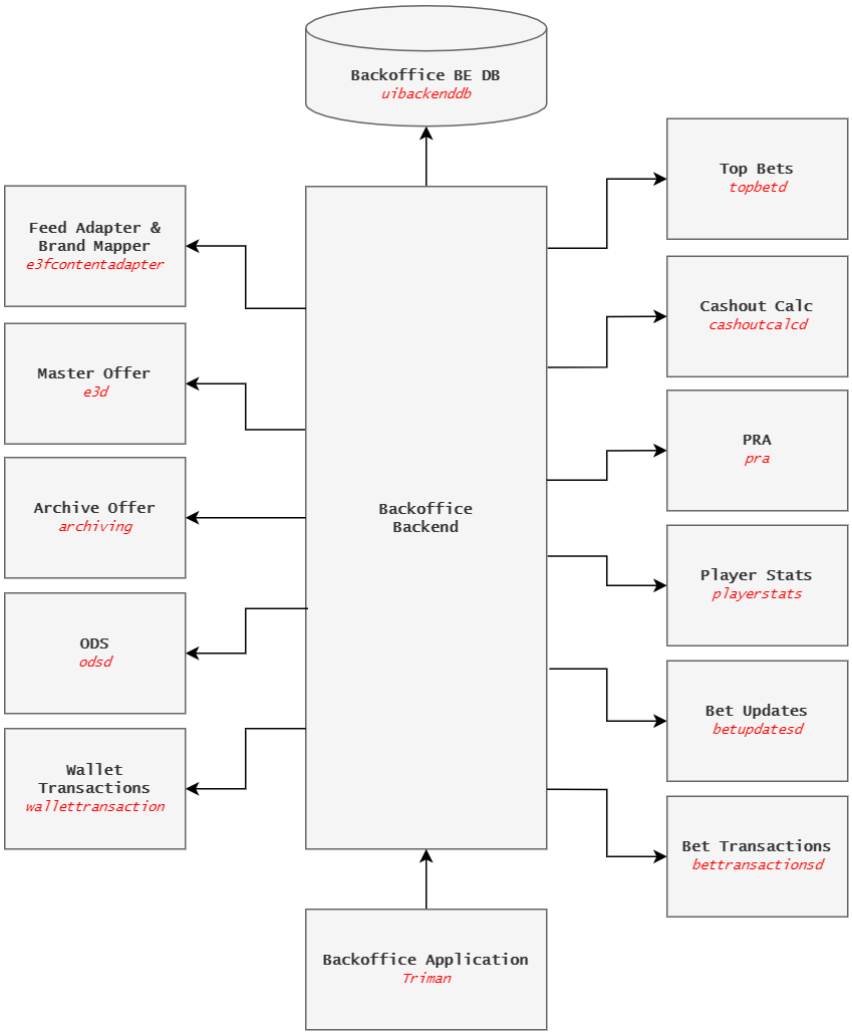

FE Livedoc Service Tech.Spec.

Backoffice Backend

Backoffice Backend is a collection of APIs used by Backoffice Application to provide and change data. In other words, Backoffice Backend presents a middle layer between the Backend Website and underlying core services that provide the actual data and perform operations on them.

Understanding the Backoffice Backend as the middleman, its APIs implement auth by which user is validated in sense of him being logged-in and assigned a valid token and him having required Backoffice Application roles to perform the required operation.

Core components that Backoffice Backend talks to are shown in the picture below.



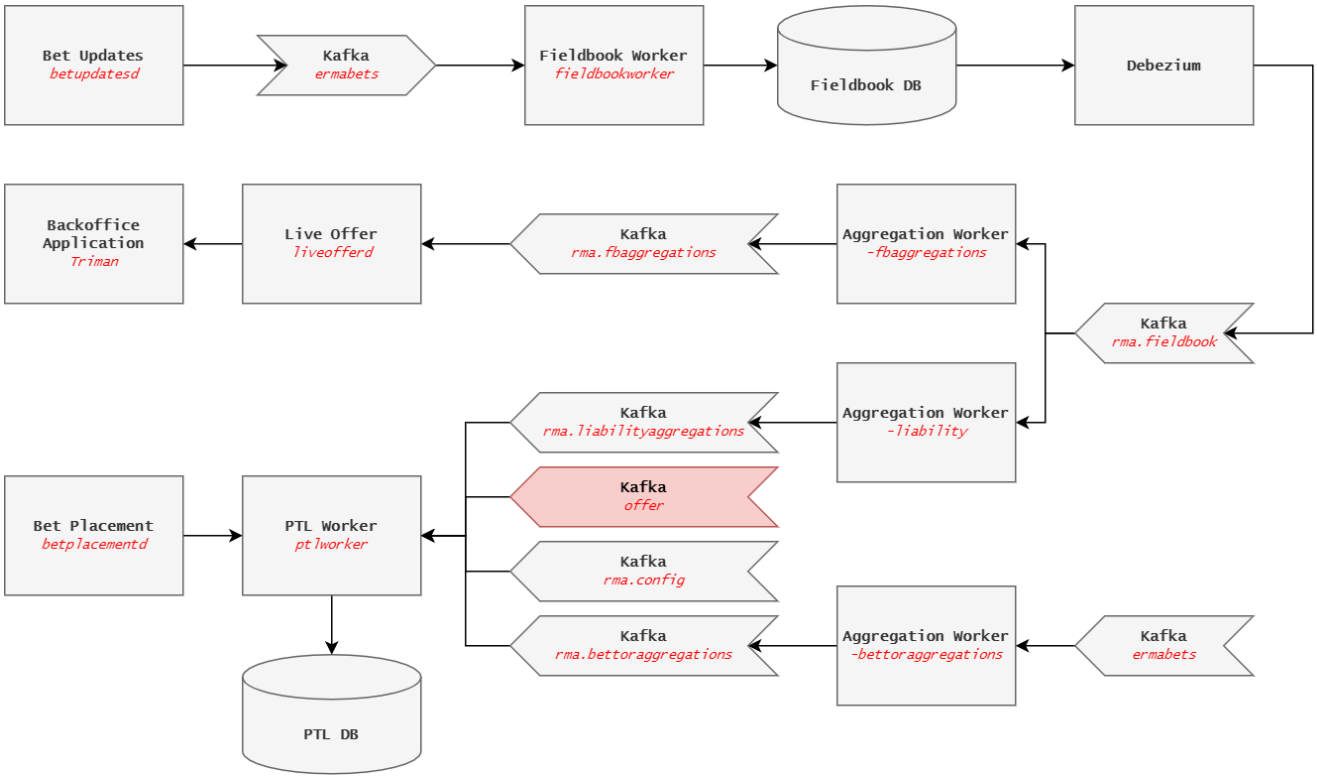
Backoffice Backend connections diagram

Automatic Risk Control block

TRI Sportsbook provides the capability to limit risk in a Sportsbook operation by specifying limits for various Risk rules. A few different services work together to ensure these risk rules are adhered to. These are:

- Fieldbook worker which maintains aggregated fieldbook data based on incoming bet deltas
- Aggregation worker which maintains bettor, liability and fieldbook aggregations to enable risk rule enforcement
- PTL worker that serves as the gatekeeper for bet placement by consuming aggregations produced by the AG-W and using that data to determine whether to allow, deny or reoffer diminished stake for a bet placement request.

The figure below shows their interworking.



Automatic Risk Control block logical diagram

Fieldbook Worker

Fieldbook Worker is responsible with maintaining market-level aggregations based on placed bets and their changes.

When a bet is placed and as it progresses through its lifecycle, the Bet Updates service produces so called “bet delta” messages (which contain a set of flags, current totals and differences from the previous bet version) into *ermabets* Kafka topic. Fieldbook Worker consumes these bet delta

messages, aggregates them and updates fieldbook data in the DB. In this sense, a fieldbook is essentially the aggregation of market level data, but at a more fine-grained level.

Aggregation Worker

Aggregation Worker is a component responsible for maintaining bettor, liability and fieldbook (a.k.a. event by market) aggregations. One worker instance may be configured to maintain current state of either bettor, liability or fieldbook aggregations in memory, so it does only one "type" of aggregations at a time.

At start-up Aggregation Worker configures itself by reading from the Backoffice Backend DB tables and then stays up to date by consuming updates happening in these tables pushed by Debezium through Kafka *config* topics group.

Once operational Aggregation Worker instance which is configured to maintain bettor aggregations starts consuming messages from Kafka *ermabets* topic to maintain these. It outputs data into *rma.bettoraggregations* Kafka topic for PTL worker to consume.

Similarly, Aggregation Worker instance which is configured to maintain liability or fieldbook aggregations starts consuming messages from Kafka *rma.fieldbook* topic to maintain its current aggregations snapshot. It outputs data into either *rma.liabilityaggregations* Kafka topic, or *rma.fbagregations* Kafka topic – based on the type of aggregations it maintains. Liability aggregations are required by PTL worker whereas fieldbook aggregations are required by Live Offer service to publish them to Backoffice Application for Linebook widget and Markets in Event Positions tab.

Multiple Aggregation Worker instances of the same type may be created to ensure scalability.

PTL Worker

PTL Worker provides actual Automatic Risk Control (ARC for short) functionalities by accepting PTL (Permission-To-Lay) requests from the Bet Placement service and returning a verdict being one of accepted, rejected, or reoffered.

At start-up PTL Worker configures itself by reading from the Backoffice Backend DB tables and then stays up to date by consuming updates happening in these tables pushed by Debezium through Kafka *config* topics group.

Once operational, PTL Worker starts consuming messages from following Kafka topics:

- *rma.liabilityaggregations*, *rma.bettoraggregations* for liability and bettor aggregations data respectively, and
- *events*, *competitions*, *sports* for betting program object data required to be stored in PTL records.

Based on current configuration, PTL worker validates attempted bet against any of the following risk rules (whatever is turned on by the config) and by using limits, factors and other values from the risk configuration:

- Always Lay Rule

- Stake Limiting Rules group:
  - Simple Max Bet Stake
  - Simple Max Bet Win
  - Max Accumulated Open Stake
  - Max Accumulated Open Win
  - Max Same Single Bet Win
  - Max Book Loss
  - Max Same Multiple Bet Win
- Frequency Limiting Rules group:
  - Max Repeated Bet Frequency

Frequency Limit rules are applied only if the request is rejected by the Always Lay Rule. Frequency Limit rules then may decide to reject the bet if violating the configured frequency. If compliant, bet is then passed through all of Stake Limiting rules where each one calculates its own max acceptable stake for the attempted bet. If all those values are above the tried bet stake, bet is accepted. If not, minimum of those is compared against the tried bet stake by the so called reoffer function to determine the ultimate fate of the bet. These rules are processed serially.

Multiple PTL Worker instances may be created to ensure scalability.

Further documentation

Following embedded documents provide additional info on Automatic Risk Control block.


TRI RMA ARC v1.2


Risk Rules Matrix

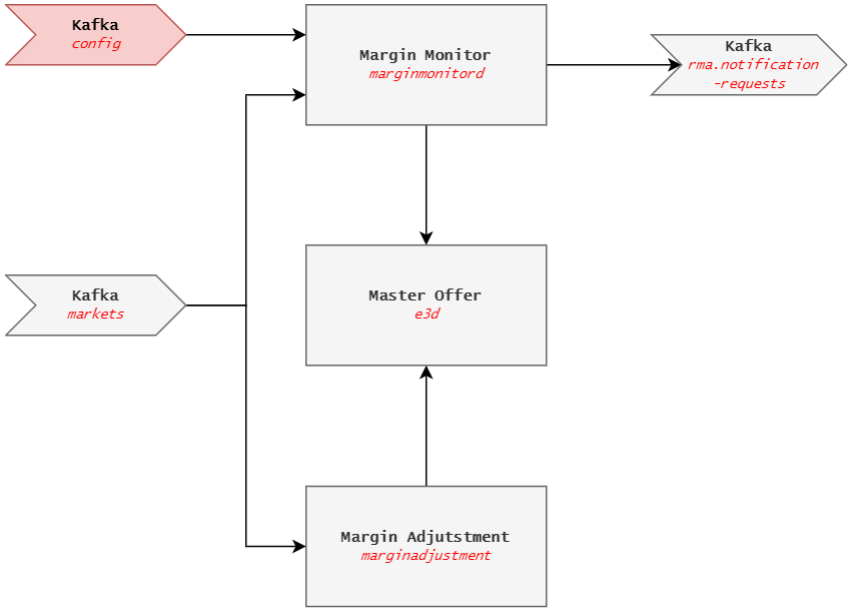

MSMW Calculation Formulas


Reoffer Function


ARC Internals

Margin Management Tools block

Margin Management Tools block contains services that provide automated operations regarding margins over betting program objects based on current configuration or overrides.



Margin Management Tools block logical diagram

Margin Monitor

Margin Monitor service is responsible for tracking current branded margins on markets and handling appropriate notifications or performing suspensions based on those margins crossing the configured alerting/suspension thresholds.

To this end, Margin Monitor listens to following Kafka queues:

- *markets* topic for new versions of markets carrying current selection prices under all brands where the market is offered so to calculate the margin for each such brand, and
- topics from *config* group to maintain current configuration of alerting and suspension thresholds

In case newly arriving market message carries such selection prices under a brand that yield margin which falls below the configured alerting threshold, Margin Monitor creates a notification request of appropriate type for the market under the endangered brand and pushes it into *rma.notification-requests* Kafka topic to be processed and propagated further by the Alerts Worker.

In case of calculated branded margin falling beneath the configured suspension threshold, Margin Monitor performs status override on the market so to automatically suspend betting under the endangered brand. It does so by calling the REST endpoint exposed by the Master Offer service. Since these overrides are temporary in nature, any subsequent selection price change

originating from the feed will automatically cancel this override, but in case calculated margin still remains beneath the configured suspension threshold, Margin Monitor will intercept the message carrying this and reapply the override.

Margin Adjustment

Margin Adjustment service is responsible for timely applying margin adjustment factor value on the market within a related brand. To this end, Margin Adjustment service listens for latest market versions (coming through Kafka *markets* topic) which carry effective margin adjustment factor values – "effective" as in being inherited from the configuration or overridden by Backoffice Application users. Regardless of the source (configuration/override), these effective margin adjustment factor values are basically a set of pairs of time-offset : factor values that dictate at which point in time preceding parent event kick-off time to apply the associated factor value.

So, for each market instance under each brand where the market is offered, Margin Adjustment service internally persists next margin adjustment factor change and related time when it's bound to be applied. Once this time comes, Margin Adjustment service performs a call to Master Offer via exposed REST and through it updates margin adjustment factor value of the market for the given brand.

Alerts & Notifications block

Components from the Alerts & Notifications block deal with handling system alerts and notifications, jointly termed as “alnots". Two-tiered approach has been adopted here where each component that is responsible for detecting the occurrence of an event (notification) or for tracking a metric (alert) first fires a notification request of appropriate type, which is then promoted to an actual alert/notification or it’s update if it fulfils certain conditions.

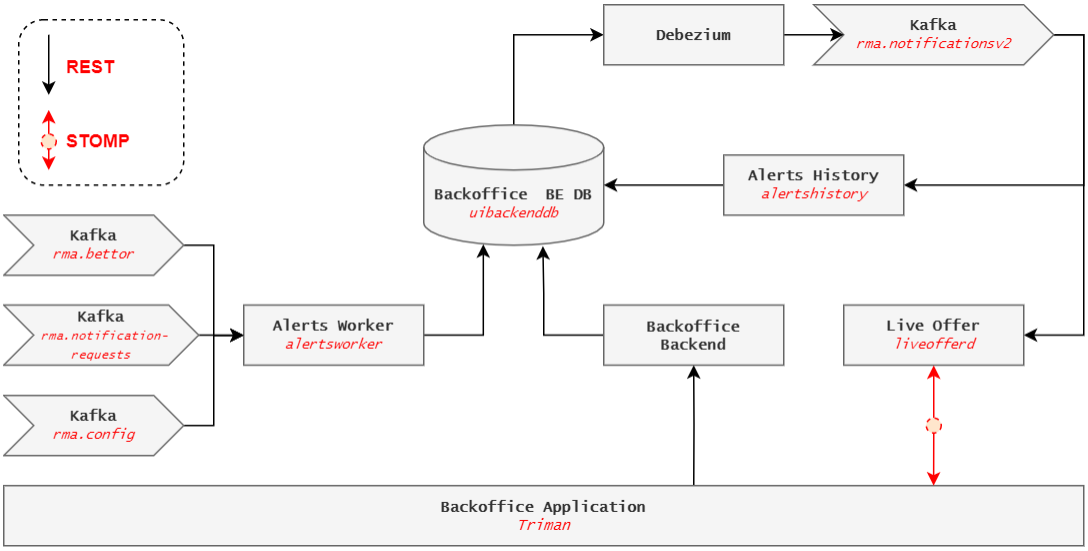
In this sense, each component that can fire an alert/notification request can be referred to as an “Emitter”. Emitters may fire one or more alert/notification types where, as a general rule, each:

- alert type is related to single metric being higher or lower than an emitting threshold,
- notification type is related to a single event type happening.

To this end, and to facilitate reusability, an emitter package has been created and is used by all components needing to act as generators of notification requests.

Whereas alert/notification requests carry only key value pairs, alert/notification type, main metric and several key values required to uniquely establish context (as to which object/entity alert relates to, e.g., TRS identifier and market identifier for MBL threshold alerts), actual alerts/notifications additionally carry further data required to display text, provide hotlinking, etc.

After an Emitter component fires an alert/notification request, it will continue to periodically renew it for as long the tracked metric is within the emitting range. Criteria for this renewal may be either time based (e.g., repeat alert request with up-to-date key values every x seconds), or delta based (e.g., repeat alert request with up-to-date key values once metric value changes for a certain amount).



Alerts & Notifications block logical diagram

Alerts Worker

Alerts Worker service is responsible for monitoring the *rma.notification-requests* Kafka topic and processing each request that comes in from responsible emitter component.

For notification requests, worker service performs data enrichment (e.g., constructing human-readable message and applying severity level) and insertion in the Backoffice Backend DB. For alert requests, worker performs the triage by comparing it against the alerting threshold and possibly previous alert version so to determine whether request is to be promoted to an alert or alert update.

Requests saved or updated in the database are then being picked up by Debezium and propagated through Kafka *rma.notificationsv2* topics for consuming components, one of which is the Live Offer service which uses this data to publish alerts and notification tickers.

Alerts Worker service uses alerts and notifications configuration, part of which can be changed through a related Backoffice Application page. These changes are pushed via Backoffice Backend APIs into the Backoffice Backend DB and then propagated towards Alerts Worker via Kafka *rma.config* topic messages pushed by Debezium.

Alerts Worker service can be scaled to cope with the increased notifications load.

Alerts History

The sole purpose of Alerts History service is to monitor the *rma.notificationsv2* Kafka topic for any raised or changed alert/notification and maintain the history in the Backoffice BE DB.

This history can be accessed at any time from the Backoffice Application where Backoffice Backend APIs are used to fetch required history data from the Backoffice BE DB.

Flow Monitor

Flow monitor is a microservice which monitors the rate of bets/PTLs being received by the system. If the number of bets/ PTLs being received falls below a certain threshold, it will raise an alert. It determines the volume of PTLs being processed by querying the PTL database for the number of PTL records inserted in a certain timeframe. It determines the volume of bets being processed by invoking the /count/bets API exposed by the BP-W. It uses the emitter package to raise alerts.

Further documentation

Following embedded documents provide additional info on Automatic Risk Control block.


TRI Alerts and Notifications Interna


Emiter Request Config

Profitability block

Profitability block contains services that provide profitability analysis features and player statistics.

PRA Bet Cacher

PRA Bet Cacher service (where PRA is short for Profitability Reporting Application) is tasked with following:

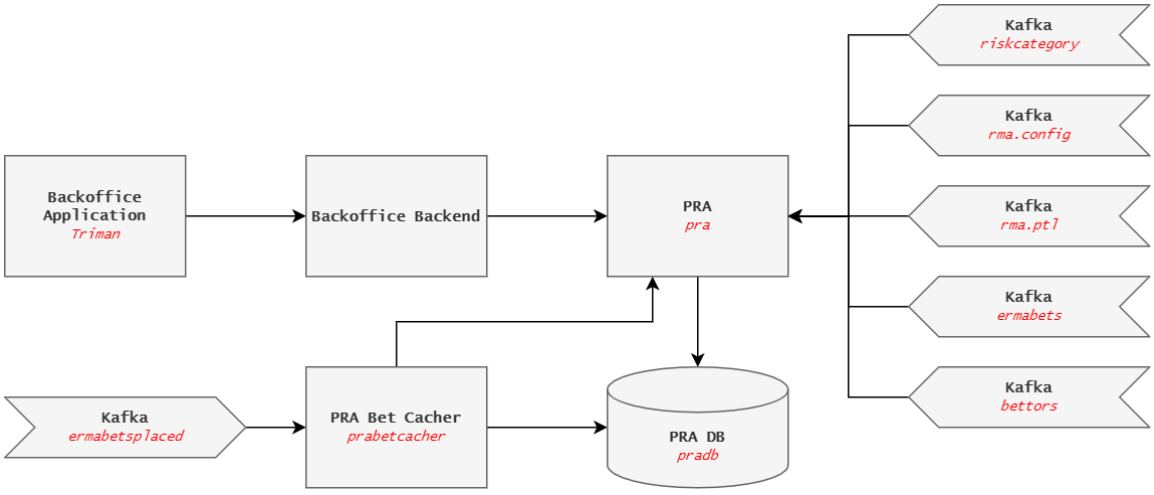
- receiving newly placed bets coming through the *ermabetsplaced* Kafka topic,
- performing lookups in the PRA config data so to establish current stake ranges, bet price ranges, player bet factor ranges, bet count ranges, etc.
- "categorizing" the bet according to current configuration, and
- storing data accordingly in the PRA DB.

PRA Backend

PRA Backend service collects related PTL info from *rma.ptl* Kafka topic to keep track of rejected stake metrics and uses additional data from *rma.config*, *riskcategory* and *bettors* Kafka topics to further enrich preserved data.

Also, upon each change received for the bet coming through the *ermabetsupdate* Kafka topic, PRA Backend service evaluates whether the change affects dimensions under which bet previously contributed the statistics and performs "recategorization" of the bet across dimensions when required.

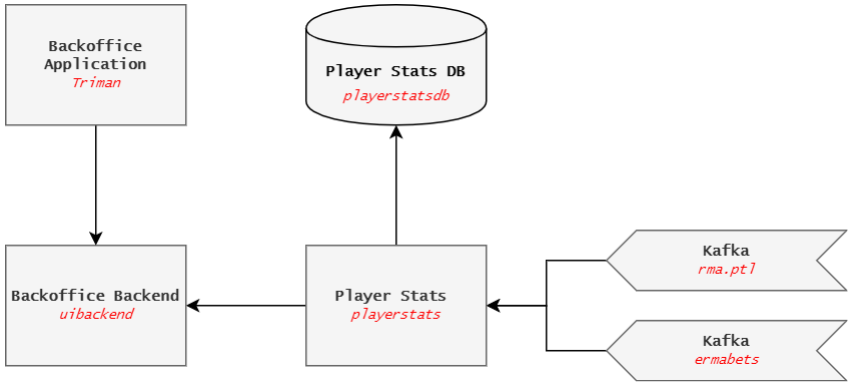
Finally, PRA Backend service sports REST endpoints that are used by Backoffice Backend APIs to filter and group profitability information of interest upon user action performed on the Backoffice Application PRA screen.



PRA part logical block diagram

Player Stats

Player Stats is a microservice which consumes data from Kafka to maintain statistics per player using data from the *ermabets* and *rma.ptl* topics. It exposes APIs for UI Backend to retrieve data for a Backoffice Application user. It maintains stats data in the Player Stats DB, as shown in the diagram below. Data are stored with a one-day granularity. This means that, in the worst-case scenario, DB will hold one entry per player, per day.



Player Stats part logical block diagram

Messages from the *rma.ptl* topic are consumed to keep track of rejected stake metrics for players and messages from *ermabets* are used to record all other metrics like bet count, total stake, and potential return. Kafka *ermabets* topic is consumed instead of *ermabetsplaced* because of the need to keep track of open stake and potential return for a player.

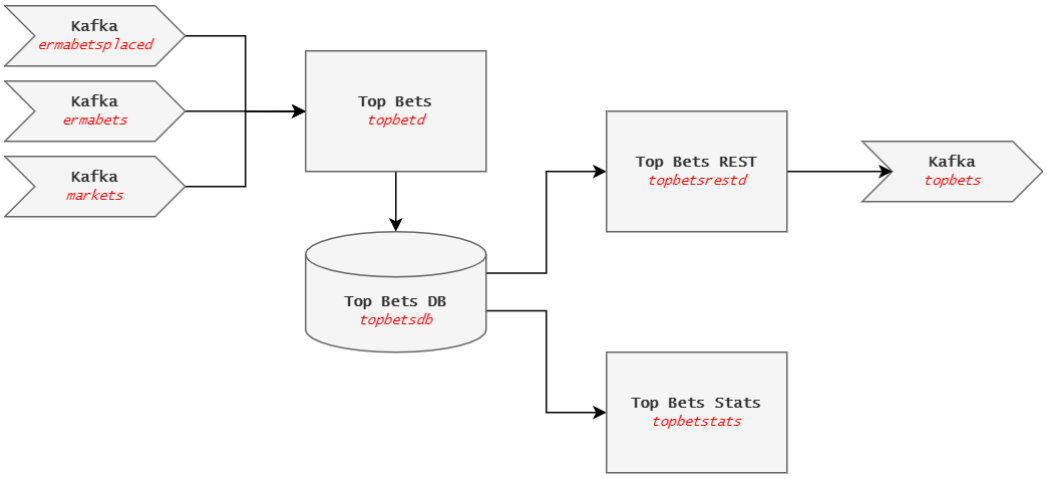
Further documentation

Following embedded documents provide additional info on Profitability block.

PRA & Player Stats Internals

Top Bets block

Top Bets block contains services that are responsible for providing Top Bets feature on Player Website and administering it via Backoffice Application. Top Bets feature provides a top-list of most popular bettable selections within a brand or overall, during a certain period in time.



Top Bets block logical diagram

Top Bets

Top Bets service tracks bet placements and changes on bets and related markets as they happen so to maintain branded statistics regarding number of open bets placed on selections. To this end, Top Bets service listens to *ermabetsplaced*, *ermabets* and *market* Kafka topics which are the main source of data on bets and related markets. Within Top Bets DB, service maintains daily branded rankings of selections according to number of open bets that they back. Once parent market gets closed for betting, Top Bets service purges all related rows from the Top Bets DB including information on overrides.

Top Bets REST

This service provides REST endpoint which is used by Backoffice Application (via Backoffice Backend) to perform branded overrides of actual number of open bets backing selections. This provides a tooling for back-office users having appropriate role to affect selection rankings on Player Website. These overrides are stored separately from actual bet numbers in the Top Bets DB and are purged together with the deletion of related selection data.

Additionally, Top Bets REST provides on-demand information on number of bets placed over the targeted market. This information is required by the Archiving Service so to enrich betting offer object data in the archive and enable advanced searching.

Top Bets Stats

This service periodically pushes top bets statistics to Kafka *topbets* topic which are then used by the Livedoc service to publish rankings on the.

Other Services block

Other Services block contains various components that provide required auxiliary functions required by one or more TRI Sportsbook services.

Forex Importer

Forex Importer service is responsible for periodically querying external forex provider on currency exchange lists and maintaining those internally. It also provides current or target-time exchange rates to other TRI Sportsbook services performing monetary calculations.

Mapper

Mapper service prepares events for insertion into elastic in order to enable Player Website search feature.