Доклад: **Каким должен быть надежный поставщик данных?**

В этом докладе обсудим:

- Как можно экспортировать данные потребителям;
- Какие требования предьявляются к экспорту данным, на примере проекта ADF;
- Какие проблемы могут возникать при поставке данных (на стороне поставщика и потребителя);
- Каким образом организуется внутренний Heartbeat в экспортной системе;
- Что такое recovery процесс, когда и зачем его проводят. Как его можно реализовать и какие проблемы могут при этом возникнуть;
- Выводы

Начнем того, что такое экспорт данных и зачем это нужно.

Основной продукт компании Altenar — это инструмент, предоставляющий букмекерскую информацию о коэффициентах рынков и статистики матчей в прямом эфире
Одним из ключевых проектов внутри компании является проект Altenar Data Feed (ADF) — который как раз занимается интеграциями с поставщиками данных, ведет матчи, занимается рассчетом данных и их экспортом в другой внутренний продукт компании — SportsBook (SB2) — который уже отображает данные в рамках своего frontend
ADF экспортирует в SB2 информацию о всех рассчитанных коэффициентах на матч (после проведения итераций внутренних рассчетов, происходящих с некоторой регулярностью), расписании матчей и лайв статистики по матчам

Каким образом производится экспорт данных?
Экспорт данных производится с использованием брокера сообщений RabbitMQ и использования единого потребительского exchange — который имеет несколько routing keys (RK) для обеспечения логистики доставки данных потребителю в виде SB2 Import
То есть на деле ADF экспорт, получая какие то внутренние отчеты, выполняет некоторые конвертации над внутренними моделями данных, переводит их в формат потребителя (он описан в рамках JSON схемы для каждого типа экспортируемых данных) — и отправляет сообщение в exchange, указывая формат destination (целевой точки отправки)

Звучит неплохо, не правда ли?
Но у этого подхода есть и свои потенциальные затруднения. С одной стороны это гибкое асинхронное решение — экспорт данных создает сообщение и отправляет его в очередь — а потребитель сам обработает сообщение, когда будет к этому готов, сам может обработать на своей стороне возможные косяки с дублированием или перетиранием данных (обеспечить так называемую

идемподентность данных)
Но проблемы начинают появляться тогда, когда начинаются какие то проблемы на стороне поставщика или потребителя

В случае, если потребитель по какой то причине не мог читать очередь из сообщений какое то время — он, проснувшись, увидит, что у него скопилась огромная очередь (из сотен сообщений) — которую он не может быстро обработать — из-за этого более новые данные, сгенерированные ADF не могут своевременно появляться у потребителя на админке, где он мониторит их

В целом — это проблема частично может решаться использованием сдвига offsetов по очереди потребителя. В рамках потребляемой очереди можно видеть текущий лаг (кол-во непрочитанных сообщений, которые предстоит прочитать перед тем, как перейти к самому последнему) — в целом ручными и автоматизированными инструментами можно мониторить этот лаг и производить его уменьшение — сдвиг последнего сообщения.
Но в таком случае стоит понимать, что часть сообщений будет намеренно пропущено — и даже, если они не содержат самых актуальных интересуемых данных — эти пропущенные данные могут содержать потенциально важные данные для поддержания состояния потребителя

И надо думать, что с этим можно сделать...

Теперь перейдем к обзору ситуаций, когда у поставщика данных (ADF) случается что то плохое и:

- он либо не может слать какой то вид данных
- произошла ошибка в важном сервисе по импорту и просчету данных

Обращу внимание на момент того, что сейчас в системе обслуживается 5 видов спорта — и физически для них существуют различные подсистемы импорта, просчета данных, а также они физически экспортируется по разным каналам

В обоих случаях так или иначе поставщик не может слать актуальные данные, либо все целиком, либо по какому то спорту
Импорт SB может получать данные не только от внутреннего поставщика ADF – а и у крупных внешних поставщиков, например, BetRadar — являющегося одним из титанов на рынке
Если ADF не может нормально слать данные — SB Import хочет во время узнавать об этом и принимать меры на своей стороне

Как же обеспечить информацию о жизнеспособности поставщика?

На стороне ADF был создан инструмент Heartbeat — который занимается тем, что смотрит по Kubernetes, какие из сервисов сейчас доступны и работают — то есть те, которые сами отдают hc на REST-запросы внутри k8s namespace

Кроме этого он может смотреть на то, сколько времени нет этого самого hc
При этом у каждого сервиса есть некоторая спецификация — то есть инструкция того, на основании чего можно инструменту Heartbeat принимать решение о том, является ли сервис системы доступным или нет
Кроме того, каждый сервис помечен флагом, должен ли он влиять на общий heartbeat – критически важные сервисы должны сразу ломать общий heartbeat

Но у этого решения тоже есть свои минусы. Например, в текущем виде весь экспорт данных идет через неспортозависимые сервисы — то есть получая проблемы в ведении матчей по футболу мы не можем нормально вести экспорт данных по остальным спортам
И плюс к тому, бизнесу бы не хотелось, чтобы проблемы при ведении отдельных матчей (не слишком приоритетных) влияло на ведение и экспорт данных по более важным матчам (каких-то крупных лиг, потеря денег на которых будет достаточно ощутимой)

В данный момент как раз ведется разработки решений, призванных справиться с перечисленными проблемами.
Ключевое, что экспортер данных теперь будет способен сигнализировать о проблемах, произошедших в каких то отдельных матчах — и не останавливать поток сообщений о матчах остальных — что очень эффективно.

Вот, мы вроде бы частично порешали проблемы, которые могут возникнуть у нас при экспорте данных
Но что, если проблемы возникли у поставщика

Мы уже разобрали случаи того, когда потребитель не был какое то время активен и поэтому находится в ситуации, когда ему нужно обработать неподъемный набор данных — причем данных по всем спортам, и в рамках спорта по каждому матчу.
Но бизнес например хочет все равно получить актуальные данные на своей стороне хотя бы по каким то отдельным матчам.
Или хотя бы данные за какой то промежуток времени, чтобы хотя бы частично синхронизироваться по состоянию с поставщиком — а все остальные данные обработать уже позже, когда на это останутся ресурсы

Тут то мы и подходим к такому инструменту как recovery
По русски это звучит как операция восстановления. Это закрепленный термин в контексте поставщиком данных
В рамках разработки такого собственного инструмента мы вдохновлялись интерфейсом (использовали за основу REST API) который используется BetRadar

Что же предоставляет данный инструмент?
Он дает возможность в ручном режиме через HTTP API запрос послать запрос

на формирование восстановления данных по определенному матчу/всем матчам, изменения по которым произошли за некоторый промежуток времени
Далее, если говорить верхнеуровнево, recovery будет искать и формировать для каждого подходящего под запрос матча снэпшот (снимок) всех сформированных за это время изменений — и пошлет их потребителю по RabbitMQ — но в рамках другой очереди — чтобы он имел возможность обработать эти данные, не потеряв состояние данных из основной очереди, которые ему пришлось бы пропустить

Какие проблемы могут быть в данном подходе?

Одной из основных проблем, над которой мы сразу задумались — это то, не возникнут ли ситуации гонок между разными отправителями данных со стороны ADF у потребителя
То есть условно из обычной очереди может прийти изменение в тайминг А по матчу, и через 1 секунду придет изменение из очереди рекавери по тому же матчу — но при этом оно будет содержать более старые данные

Как такое возможно?
Физически, экспортом данных в основном потоке и экспортом рекавери данных занимаются разные сервисы — которые параллельно слушают изменения из внутренних Kafka топиков. Мы в целом никак не может синхронизировать на нашей стороне параллельную работу между ними
Но по итогу мы пришли к тому, что у каждого изменения, которое приходит к нам из домена рассчета данных, есть некоторый timestamp — метка производства данных — и мы экспортируем ее внутри сообщения
То есть разруливанием данных занимается сам потребитель, используя эти самые временные метки

Вроде бы все хорошо, подходы и архитектурные решения, спроектированные и примененные нами лишают нас практически всех проблемных кейсов, которые возникают как при ошибках и простоях и на стороне поставщика данных, и на стороне потребителя данных

Из минусов, про которые стоит упомянуть — это то, что сама экспорт система благодаря эти нововведениям очень сильно разрастается, появляется куча новых сервисов, хранилищ данных.
Изменения по бизнес логике и транспортным моделям теперь необходимо будет вносить в несколько разных мест, а не только в одном, как ранее

Частично решить эту проблему мы попытались с использованием подхода с вынесением общей кодовой базы в библиотеку — которая билдится как внутренний nuget пакет, который мы можем использовать внутри отдельных сервисов — и соответственно, проводить меньше копипаста
Но пока не получилось провести вынос всей общей кодовой базы, например,

транспортных моделей и связанных конвертаций — это связано во многом с тем, что в целом решили не использовать подход с переносом логики транспортного уровня в библиотеку

Какие выводы можно сделать на основании этого:

- Доставка данных в распределенных системах является сложным и затратным процессом
- В рамках процесса этой доставки может возникнуть множество редких, но больных ситуаций, которые придется как то решать
- Ситуации возникают либо на стороне поставщика, либо на стороне потребителя — они решаются по разному
- Обеспечение полной надежности доставки данных, создание мощных отказоустойчивых решений может обойтись достаточно дорого как по времени проектирования, реализации — так и сильно усложнит дальнейшую поддержку проекта
- В некоторых случаях действительно можно забить на какие то супер-редкие corner кейсы, решение которых может повлечь полный пересмотр реализации компонентов системы