**Проблемы и риски текущей архитектуры**

1. **Увеличение времени отклика**
   * Сервис ins-product-aggregator запрашивает информацию у страховых компаний в реальном времени, что приводит к задержкам в ответах из-за:
     + Медленных API страховых компаний.
     + Пиковых нагрузок, увеличивающих время отклика.
     + Возможных сбоев в API (например, временной недоступности).  
       **Риск:**
   * При добавлении ещё 5 страховых компаний время отклика сервиса ins-product-aggregator существенно увеличится.
2. **Ненадёжность взаимодействия через REST**
   * Сервисы используют синхронные REST-запросы для получения данных. Ошибки на любом из этапов (задержка ответа, недоступность API) приводят к сбоям в работе системы.  
     **Риск:**
   * Повышенная вероятность ошибок взаимодействия с ростом нагрузки и количества запросов.
3. **Дублирование данных**
   * Каждый сервис (например, core-app и ins-comp-settlement) поддерживает локальные реплики данных, обновляемые через отдельные запросы к ins-product-aggregator.  
     **Риски:**
   * Увеличение количества запросов для обновления данных при росте числа страховых компаний.
   * Расхождение данных между сервисами (например, из-за задержек в обновлении).
4. **Ненадёжность ночных операций**
   * ins-comp-settlement выполняет ночной запрос в core-app для получения оформленных страховок за день.  
     **Риски:**
   * Задержки или сбои в запросах могут нарушить формирование реестра оформленных страховок.
   * Ночная операция зависит от времени выполнения и доступности API.
5. **Сложность масштабирования**
   * Увеличение количества страховых компаний с текущей архитектурой приведёт к линейному увеличению нагрузки на сервисы.
   * REST API плохо масштабируется для больших объёмов данных и частых запросов.  
     **Риски:**
   * Сервисы могут стать узким местом системы.
   * Рост требований к ресурсам инфраструктуры.

**Нужен ли Transactional Outbox?**

**Да, внедрение Transactional Outbox необходимо, так как для нас критично потерять обновление данных.**  
Transactional Outbox позволит избежать проблем с потерей сообщений или несогласованностью данных между сервисами.

**Почему это важно:**

1. **Ценность данных:**
   * Потеря обновлений данных может привести к критическим проблемам, например, несоответствию информации о продуктах и тарифах между сервисами.
   * Расхождение данных приведёт к снижению качества обслуживания клиентов и репутационным потерям.
2. **Согласованность данных:**
   * Использование Transactional Outbox гарантирует, что события (например, обновление продуктов или оформление страховок) будут обработаны точно один раз (exactly-once).
   * Это устраняет риск частичной обработки событий, который особенно критичен при обновлении реплик данных.
3. **Технические преимущества:**
   * Transactional Outbox обеспечивает надежную передачу событий через механизм атомарной фиксации (ACID) между базой данных и брокером сообщений (Kafka).
   * Это упрощает архитектуру, исключая необходимость сложной обработки неуспешных транзакций (rollback).

**Пример сценария использования:**

* Сервис ins-product-aggregator при обновлении данных о продуктах записывает событие в таблицу Outbox.
* Специализированный процесс (например, Kafka Connect или Poller) передаёт событие из Outbox в Kafka.
* Это гарантирует, что даже при сбое системы обновления не будут потеряны.

**Рекомендации по внедрению Transactional Outbox:**

1. Реализовать Transactional Outbox для критичных событий, связанных с обновлением данных о продуктах и тарифах.
2. Настроить механизм обработки Outbox через Kafka или другой брокер сообщений.
3. Гарантировать атомарность операций между базой данных и Outbox, чтобы исключить расхождение данных.

Таким образом, Transactional Outbox решает проблему потери данных и обеспечивает надёжное асинхронное взаимодействие между сервисами, особенно при росте нагрузки.