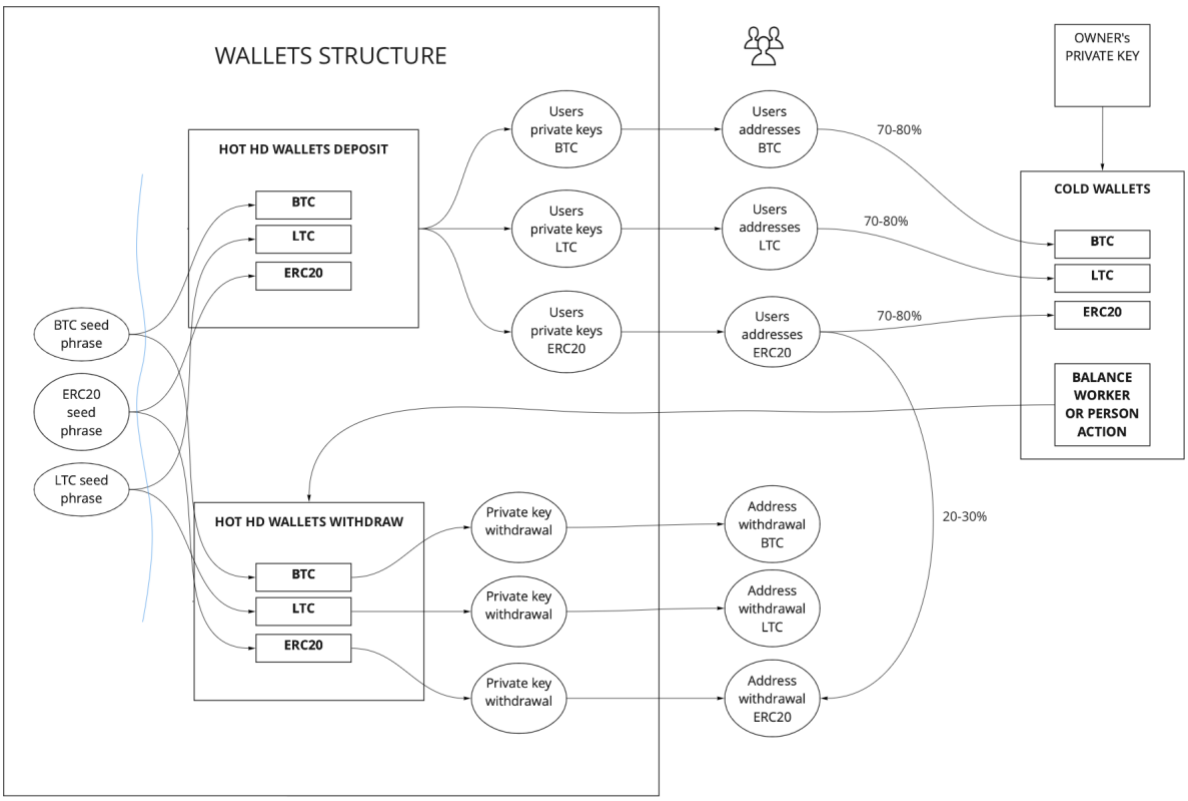
1. Wallet Service (WS)

1.1. Архитектура

Система кошельков

Система кошеьков основывается на технологии HD wallet, позволяющая генерировать приватные ключи из единой seed фразы, и подписывать inputs/outputs в единой транзакции.

В системе будут использоваться кошельки трех типов для ввода и вывода пользовательских токенов (BTC, LTC, ERC20) и 3N кошельков для пользователей, где N - количество пользователей.



Стек разработки:	**Python, Kafka**
Сборка и хранение логов:	Vector (альтернатива logstash) ElasticSearch, Kibana

1.2. Сервисы

В системе каждому пользователю будут присвоены 3 адреса (LTC, BTC, ERC20).
Для пополнения счета необходимо пополнить соответствующий адрес на желаемую сумму, отправив криптовалюту на соответствующий адрес. (Адрес выводится в личном кабинете пользователя.)

Мониторинг

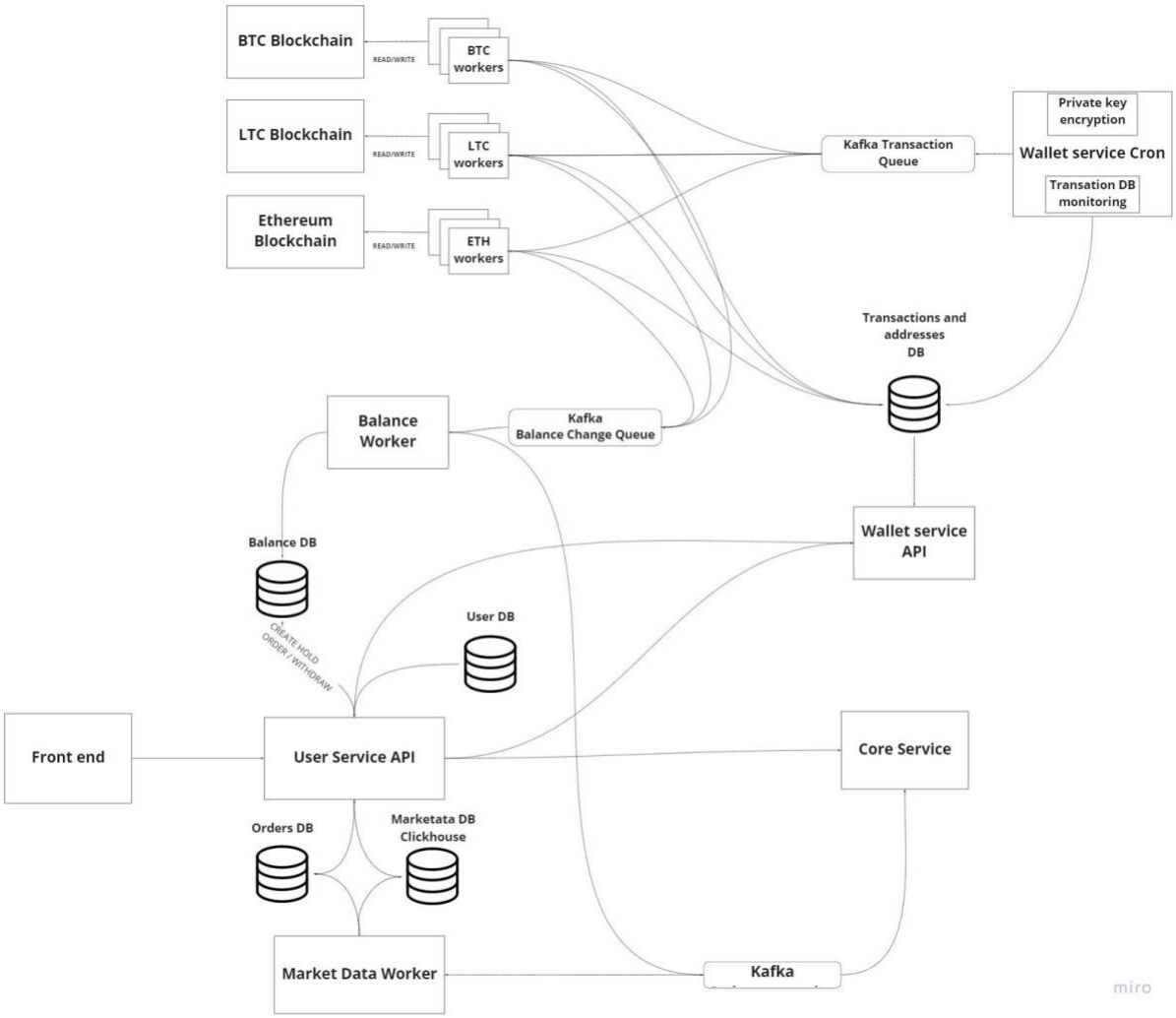
- В системе происходит постоянный мониторинг пользовательских адресов на предмет появления новых транзакций с помощью Cron скриптов и Kafka воркеров.
- При появлении новой транзакции в блокчейне (которой нет в базе данных) происходит создание этой транзакции в базе данных.
- Таблица с транзакциями также мониторится Cron скриптами и Kafka воркерами, при этом на каждый блокчейн выделен отдельный воркер.
- При смене статуса транзакции в блокчейне, происходит обновление соответствующей записи в базе данных и последующие действия.
- В случае депозита увеличивается баланс пользователя посредством соответствующего запроса на User Service API. После чего пользователь может распоряжаться своим балансов для торговли и вывода.

Блокчейн	Технология Вводы/Вывода
BTC	Unspent Transaction Outputs (UTXOs) Позволяет принимать активы на адреса пользователей и выполнять вывод с горячих биржевых кошельков без создания дополнительной транзакции на перевод с пользовательского адреса на адрес биржи.
LTC	
ERC20	Перевод с адреса пользователя (ERC20 баланса) на адрес горячего кошелька биржи для вывода и холодный кошелек владельца биржи.

Схема вывода крипто активов.

Для выполнения вывода пользователь должен создать запрос на вывод актива.

- Если у пользователя достаточный баланс, то создается запрос на вывод с холдированием баланса. После этого пользователь больше не может пользоваться холдированной частью баланса для торгов или для вывода.
- Далее из User Service происходит запрос на Wallet Service для создания транзакции на вывод. После формирования, транзакция создается в базе данных и происходит ее мониторинг (аналогично предыдущему пункту).
- Как только транзакция получает успешный статус, холд считается списанным, иначе, при неуспешной транзакции, холд возвращается пользователю и его заявка считается неуспешной.



1.3. Задачи

Раздел	Функционал	Технологии
Wallet service		
Структура БД	Создание структуры баз данных	ORM Tortoise
Интеграция с блокчейнами	Выбор и интеграция с нодой	BTC, LTC, ETH(ERC20)
	Механизм генерации кошельков для новых пользователей	
	Механизм хранения приватных ключей (Шифрование/ Расшифрование приватных ключей)	
	Управление кошельками биржи	
	Функции подписания и формирования транзакций и отправки в блокчейны	
Функции мониторинга выполнения транзакций	Обработка маленькой комиссии и пересоздание транзакции с тем же nonce	-
	Обработка ошибок сети	-
	Обработка частных кейсов для каждой криптовалюты и ERC20 токенов	-
Разработка Kafka воркеров	Воркеры трекинга созданных транзакций и изменение их статусов	-
	Интеграция с API для обновления балансов с непосредственной проверкой в блокчейне	-
	Обработка кейса при неуспешных транзакциях (возврат баланса)	-
Тестирование	Проработка механизма идемпотентности запросов	-
	Тестирование функционала в тестовых сетях	-
	Тестирование функционала в продакшен сетях	-
API	Разработка API для выдачи данных по транзакциям и кошелькам	Fast API + подключение Swagger

Мониторинг	Разработка административной панели для мониторинга внутренних процессов	Tortoise Admin
	Разработка инструментов аналитики и мониторинга системы??	Grafana,Prometheus, Sentry

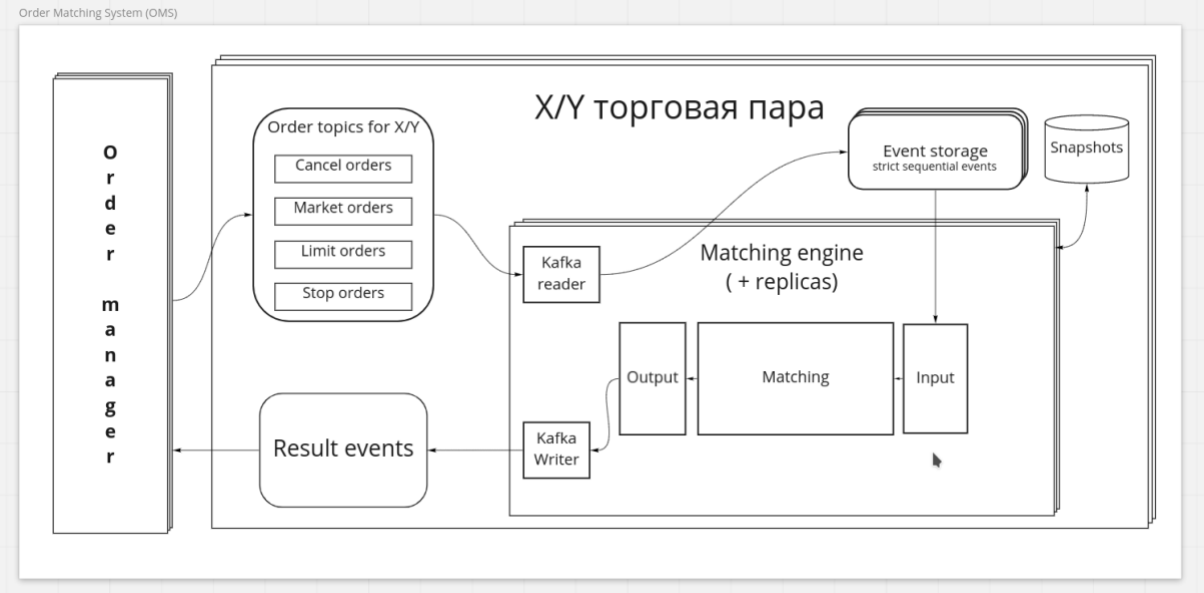
1.4. BitGo (версия)

Раздел	Функционал	Технологии
Wallet service		
Структура БД	Создание структуры баз данных	ORM Tortoise
Интеграция с блокчейнами	Интеграция с BitGo	BTC, LTC, ETH(ERC20)
	Механизм генерации кошельков для новых пользователей	
	Механизм хранения приватных ключей (Шифрование/ Расшифрование приватных ключей)	
	Управление кошельками биржи	
	Функции подписания и формирования транзакций и отправки в BitGo	
Функции мониторинга выполнения транзакций	Обработка маленькой комиссии и пересоздание транзакции с тем же nonce	-
	Обработка ошибок сети	-
Разработка Kafka воркеров	Воркеры трекинга созданных транзакций и изменение их статусов	-
	Интеграция с API для обновления балансов с непосредственной проверкой в BitGo	-
	Обработка кейса при неуспешных транзакциях (возврат баланса)	-
Тестирование	Проработка механизма идемпотентности запросов	-
	Тестирование функционала в тестовых сетях	-

	Тестирование функционала в продакшен сетях	-
API	Разработка API для выдачи данных по транзакциям и кошелькам	Fast API + подключение Swagger
Мониторинг	Разработка административной панели для мониторинга внутренних процессов	Tortoise Admin
	Разработка инструментов аналитики и мониторинга системы??	Grafana,Prometheus, Sentry

2. Order Matching System (OMS)

1.1. Архитектура



Стек разработки:	Rust, Tokio, Diesel, Tonic (Grpc), Protobuf, Flatbuffers / Cap'n proto, Rocket / Artix-web
Сборка и хранение логов:	Vector (альтернатива logstash) ElasticSearch, Kibana

1.2. Сервисы

Order manager (api-gateway)

- Знает про все инстансы Matching Engine и умеет общаться с их лидерами.
- gRPC (google remote procedure call)
- Горизонтально масштабируется.

Matching engine

- Детерминированная система: состояние определяется входными данными (event sourcing).
- Все состояние полностью в RAM для скорости.
- Матчит покупателя с продавцом.
- Шардирование по торговым парам: каждый инстанс - отдельная торговая пара.
- Умеет среди одной торговой пары выбирать лидера и в случае его смерти перевыбирать нового (failover + raft consensus).
- Поддержка снапшотов и восстановления состояния при падении.

Очередь сообщений Kafka

- Exactly once semantic
- Своя для OMS.

1.3. Задачи

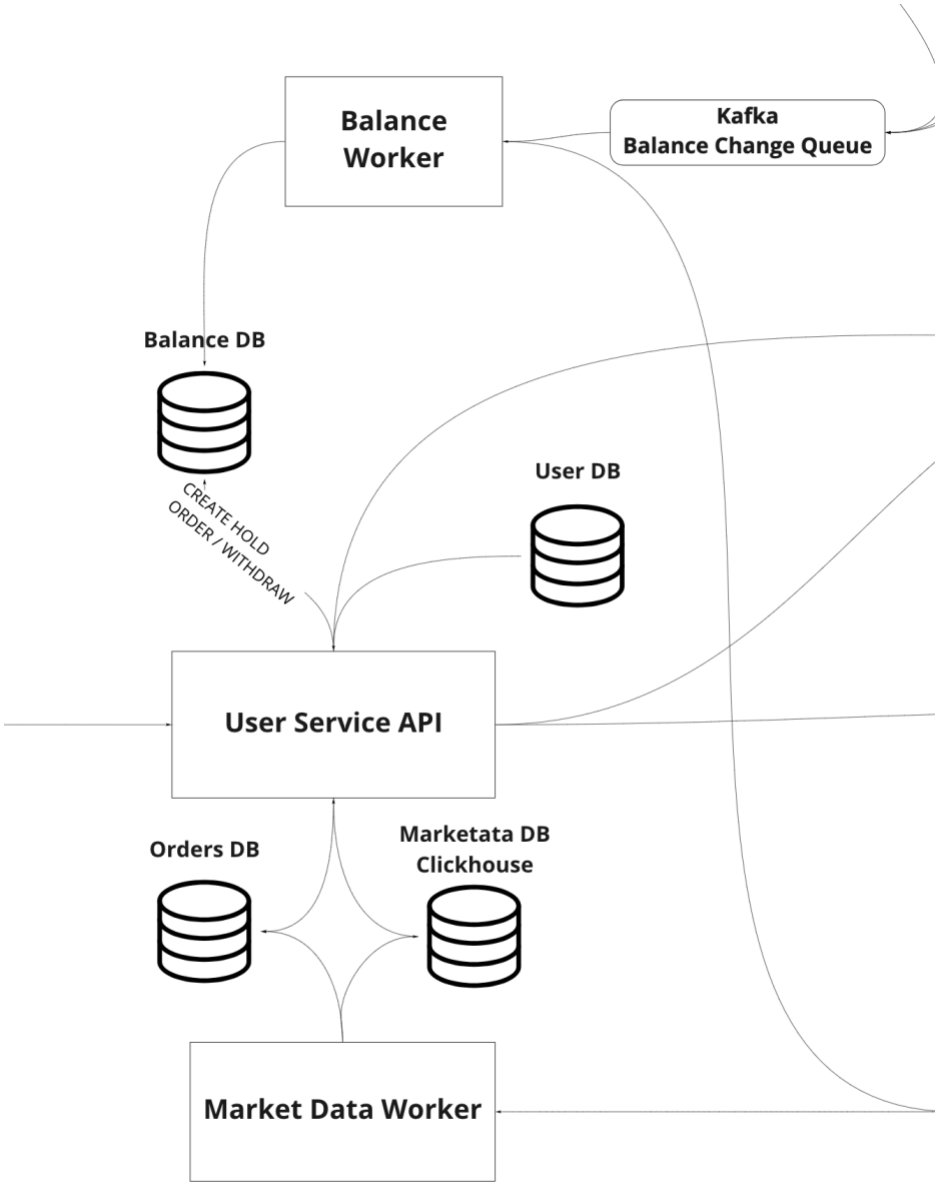
Раздел	Функционал	Технологии
Order Matching System (OMS)		
Механизм работы ордеров (Лимит и Стоп ордер)	Реализация механизма Limit Matching engine (Cancel, Limit, Stop)	-
	Работа с Kafka и Event Storage	Tokio, rdkafka
	Механизмы Fail Tolerance и failover внутри одной торговой пары	Raft
	Написание юнит тестов	-

	Формирование раздельных очередей с учетом приоритетов для разных типов заказов (Лимит, Стоп, Отмена)	Kafka
	Механизм коммутативности операций Cancel с остальными (Limit, Stop)	-
	Не матчить маркетмейкеры друг с другом и пользователя с самим собой	-
Механизм работы со стаканом	Реализация функциональности стакана	
	Получение информации по стакану через API.	-
Брокер сообщений	Формирование структуры данных для брокера сообщений	-
Результаты матчинга	Механизм информирования о результатах матчинга: преобразование внутренних эвентов OMS во внешние для остальных сервисов.	-
Механизм защиты от дублей	Защита от дубликатов сообщений из кафки (идемпотентность & exactly once delivery)	-
Механизм восстановления	Формирование базы для хранения снапшотов	RocksDB
	Механизм создания, сохранения и восстановления из снапшотов	-
Service discovery	Механизм обнаружения всех matching engine и их реплик.	-
Order manager (API gateway)	Работа с ордерами (создание / отмена)	-
	Получение стаканов	-
	Обнаружение matching engine	-
	Механизм отдачи внутренних событий о сделках во внешний мир	-
	Docker образы для сервисов	-

CI/CD	Сборка в Gitlab-ci	-
	Деплой в k8s	-

3. User Service (US)

3.1. Архитектура



3.2. Сервисы

User Service API

- Обработка запросов от фронта/мобильного приложения/API интеграций - входная точка интеграции с бэкендом биржи
- Контракты для внешних клиентов используя swagger

- Обработка авторизации и регистрации пользователя, логина, сброса пароля, двухфакторной аутентификации
- Возможность выставить/отменить ордер, холдирование балансов, ввод и вывод денежных средств
- Получение данных о текущих курсах, сделках, доходностях, биржевом стакане, графиков курсов валют
- Получение данных о текущих ордерах и их статусах исполнения
- Кеширование горячих данных
- Горизонтальное масштабирование
- Защита от double spend и корректная транзакционная обработка финансовых инструментов

Market Data Worker

- Обработка kafka исполненных заявок от торгового ядра
- Хранения данных о ценах финансового инструмента в момент времени
- Агрегация сырых данных для получения биржевых свечей в различных временных интервалах 5 мин/30 мин/4 часа/1 день/1 неделя.
- Кеширование популярных графиков для ускорения выдачи
- Сохранение ордеров пользователя со статусами и ценой их исполнения для их последующего отображения.
- Горизонтальное масштабирование
- Работа с финансовыми данными без потерь точности

Balance Worker

- Обработка kafka исполненных заявок от торгового ядра
- Списание, расхолдирование, зачисление денежных средств и активов
- Горизонтальное масштабирование
- Работа с финансовыми данными без потерь точности

3.3. Задачи

Раздел	Функционал	Технологии
User Service API (US)		
Личный кабинет	Авторизация и Регистрация, сброс пароля, доверенный браузер, двухфакторная аутентификация, хранение личных данных о пользователях	Сессии (Cookies/ Auth Token), Хэширование паролей, email (smtp/sendpulse) и

		push нотификации (firebase), google authenticator, sms API, PostgreSQL + PGBouncer
	Получение истории ордеров пользователя с актуальными статусами исполнения	Интеграция с БД ордеров
Работа с балансами и финансами	Информация о своем портфеле, балансы в различных валютах, ввод и вывод средств	API интеграции с wallet service API, транзакционное хранилище балансов на PostgreSQL, полинг и стриминг обновления портфеля на WebSocket/Centrifuge
Работа с финансовыми инструментами и графиками	Получение графических и числовых данных об исторической стоимости финансовых инструментов за выбранный период времени. Хранение информации об инструментах	Интеграция с Clickhouse для получения свечей. API интеграция с OMS для получения данных о текущем биржевом стакане.Полинг и стриминг обновления графиков на WebSocket/Centrifuge
Работа с историческими данными ордеров	Получение истории ордеров пользователя с актуальными статусами исполнения	Интеграция с БД ордеров
Выставление ордеров и работа с биржей	OpenAPI для выставления ордеров и получения статуса по ним. Предварительное холдирование балансов. Защита от двойных трат	Swagger, OpenAPI, API интеграция с OMS.
Оптимизация нагрузки и кэширование	Кэширование горячих запросов для уменьшения нагрузки на бд и увеличения пропускной способности	LRU Cache, redis
Market Data Worker		

История ордеров пользователя	Проектирование распределенного кластера БД ордеров с возможностью партиционирования, шардирования и репликации данных. Загрузка и обновление данных	PostgreSQL, TimescaleDB, pgbouncer, Kafka
Формирование биржевых свечей и графиков	Проектирование распределенного кластера БД свечей с возможностью партиционирования, шардирования и репликации данных. Загрузка и обновление данных. Агрегация сырых данных в свечи различных временных промежутков	PostgreSQL, TimescaleDB, Clickhouse, materialized view, chproxy, Kafka
Balance Worker		
Обработка операций ввода вывода средств с блокчейна	Обработка событий, произошедших на блокчейне (подтверждение/отмена транзакций), зачисление, расхолдирование и снятие средств с бд. Защита от двойных трат.	PostgreSQL, pgbouncer, Kafka
Обработка исполненных ордеров	Подтверждение или отмена холдирования средств. Зачисление средств при исполнении ордера. Защита от двойных трат.	PostgreSQL, pgbouncer, Kafka
Common		
Мониторинг	Мониторинг работы приложений с помощью метрик, логов.	Prometheus, Grafana, Sentry, ELK
CI/CD	Деплой приложений через gitlab CI в K8s или аналогичную среду	Gitlab CI, K8S, ansible
Масштабирование и отказоустойчивость	Сервисы масштабируются как вертикально, так и горизонтально, по количеству партиций в Kafka. БД реплицируются и шардируются.	-