# Оценка требований, нагрузки и объема хранилищ

#### Функциональные требования:

- учет финансовых расходов в 4 валютах RUB, USD, EUR, CNY по фиксированному набору категорий трат;
- статистика по расходам за периоды (неделя, месяц, год);
- возможность задать лимиты трат по категориям;
- сообщение о превышении трат за период во время добавления новой траты;
- статистика должна расчитываться по курсу валют на момент совершения самой траты.

#### Нефункциональные требования:

- высокая скорость работы
- высокая отказоустойчивость
- высокая доступность (sla 99%)

#### Дополнительные требования:

- сбор метрики здоровья приложения через prometheus экспортеры на подах, дашборды в grafana;
  - алертинг об инцидентах в slack/telegram/по email; логирование событий жц приложения;

#### Нагрузка

# 1000 пользователей:

0.2 (0.6) RPS на добавление трат, 0.03 (0.09) RPS на запрос статистики 100 000 пользователей:

20 (60) RPS на добавление трат, 3 (9) RPS на запрос статистики 1 000 000 пользователей:

200 (600) RPS на добавление трат, 30 (90) RPS на запрос статистики PS: берем за максимальную нагрузку x3 от стандартной

# Оценка хранилища:

- для 1000 пользователей будет достаточно порядка 250 мегабайт реляционного хранилища, для 100К и 1МЛН пользователей соответственно порядка 25 ГБ и 250 ГБ
- также потребуется х3 хранилища на реплики, чтобы обеспечить доступность базы данных в случае выхода из строя основного узла, либо для распределения нагрузки по репликам
- примерно x3 хранилища на бэкапы для обеспечения возможности восстановления потерянных данных в случае уничтожения базы данных из-за атаки, физического уничтожения основных хранилищ и тп

# Оценка размера оперативной памяти:

- на инстанс приложения потребуется кешировать порядка 50 MBs, 5 GBs, 50 GBs на 1000/100K/1KK пользователей соответственно
- при росте нагрузки, для поддержки высокой скорости обработки запросов и для обеспечения высокой отказоустойчивости приложения имеет смысл держать доступными несколько инстансов приложения

#### ПОЯСНЕНИЕ ПО РАСЧЕТУ НАГРУЗКИ:

- Будем считать, что обычный пользователь будет совершать примерно 10-20 трат в день и совершать примерное 2-3 запроса статистики в неделю
- исходя из этого, получаем, что для 1000 пользователей будем получать примерно 20К запросов на добавление валюты в день и 3К запросов на получение статистики в неделю

#### - Итого, для 1000 пользователей:

20K / 24 / 60 / 60 = 0.2 RPS на добавление трат 3K / 24 / 60 / 60 = 0.03 RPS на запрос статистики

# ПОЯСНЕНИЕ К ОЦЕНКЕ РАЗМЕРА ОПЕРАТИВНОЙ ПАМЯТИ:

- в кэше имеет смысл хранить текущие ставки валют для снижения количества походов за ними в базу и/ или во внешний сервис загрузки ставок
- имеет смысл ограничить объем кэша по одному или нескольким критериям: памяти, кол-ву дней, которые кэшируем
- так как в приложении можно получить статистику за неделю, месяц и год, то далее, чем на год кэшировать нет смысла; также большинство запросов будет приходится на ближайшие дни к текущему
- можно воспользоваться законом Парето и кэшировать порядка 20% от всех имеющихся данных о ставках за последние 365 \* 0.2 дней = 73 дня и реализовать Iru-алгоритм вытеснения кэша по этому принципу

# - итого,

получим необходимость в кэшировании для 1000 пользователей:

73 (days) \* 250/365 (MBs) ~ 50 MBs кэша

#### ПОЯСНЕНИЕ К ОЦЕНКЕ ХРАНИЛИЩА:

- в базе данных будем хранить базовые данные о пользователе, его транзакции, а также ставки валют

#### таблица пользователей:

id, дефолтная валюта (enum) 1000 (users) \* (4bytes + 4 bytes) ~ 8 KBs

#### таблица транзакций:

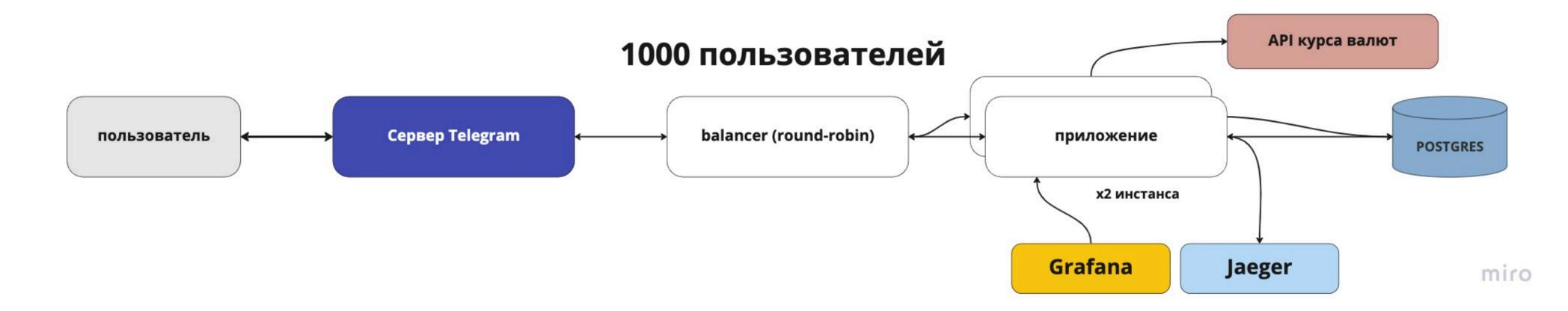
id, user\_id, category\_id, amount, created\_at 1000 (users) \* 20 (number of waste in day) \* (4bytes + 4bytes + 4 bytes + 12 bytes + 8 bytes) \* 365 (days) ~ 233.6 MBs

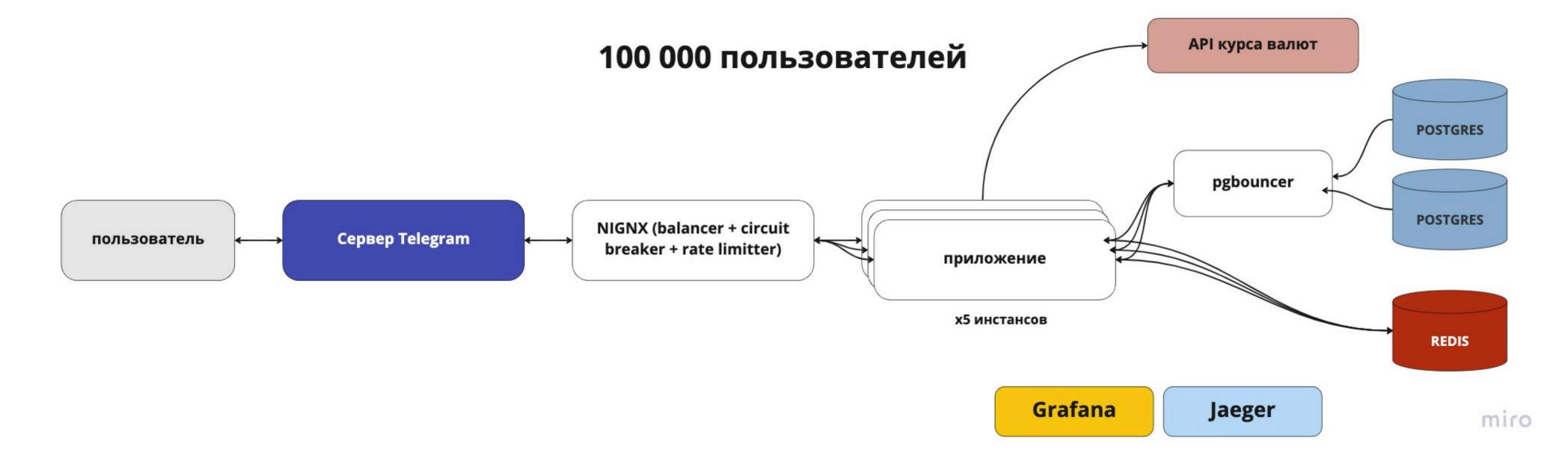
#### таблица ставок валют:

currency\_id, multiplier, on\_date
3 (types of currencies) \* (4bytes + 12bytes + 8 bytes) \* 365 (days)
~ 26 KBs

# Итого,

для 1000 пользователей в год потребуется порядка 250 мегабайт реляционного хранилища





# 1 000 000 пользователей POSTGRES user microservice POSTGRES (хранит данные о пользователе, лимиты, другие настройки) х5 инстансов POSTGRES pgbouncer transaction microservice (хранит данные о транзакциях, обрабатывает NIGNX (balancer + circuit breaker + rate запросы на построение отчетов) Сервер Telegram пользователь limitter) х20 инстансов POSTGRES rate microservice (оперирует ставками, обеспечивает подгрузку pgbouncer новых курсов, взаимодействует с кешом) POSTGRES х50 инстансов POSTGRES Grafana Jaeger REDIS АРІ курса валют miro