Урок 10 > Дополнительные подсистемы

> Оглавление

> Ограничение нагрузки на систему

Важно предусмотреть механизм ограничения нагрузки, даже если система хорошо оптимизирована и развернута на хорошем железе.

Какие варианты ограничения нагрузки чаще всего применяют:

- Количество запросов от одного пользователя
- Количество запросов от одного устройства

- Количество запросов от одного IP-адреса

Некоторые механизмы ограничений служат и целям безопасности:

- Ограничение количество неудачных попыток предотвращает атаку грубой силы (brute-force), когда злоумышленники хотят получить доступ к данным путем перебора паролей.
- Ограничение RPS может предотвратить DDoS атаку.

Размещение модуля ограничения

- На клиентской части приложения. Вариант не надежный: запросы к серверу можно отсылать не только из приложения, а еще из других специальных приложений, например, с Postman. Такие запросы пройдут мимо ограничителя.
- На серверной части — вариант хороший, но дополнительно нагружает сервер основного приложения.
- Самым оптимальным вариантом остается вынести ограничитель в отдельный модуль между клиентом и сервером. Такой вариант снимает нагрузку с основного сервиса и одновременно служит буфером для любых запросов.

Требования к модулю ограничения нагрузки

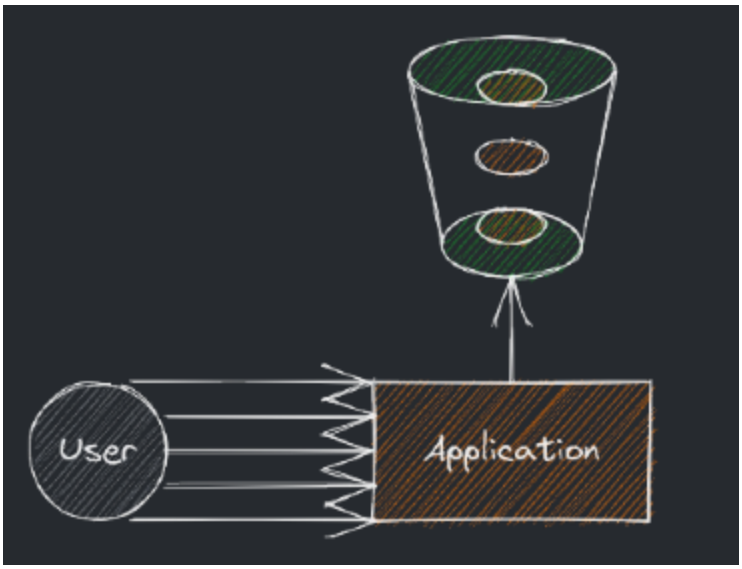
- Возможность корректно ограничить запросы к конкретному api в количестве N за время T
- Ограничитель должен быть доступен распределенно и учитывать нагрузку с разных серверов или процессов
- Ограничитель должен быть доступен всегда
- Применение ограничителя не должно сильно сказываться на быстродействии

> Алгоритмы ограничения нагрузки

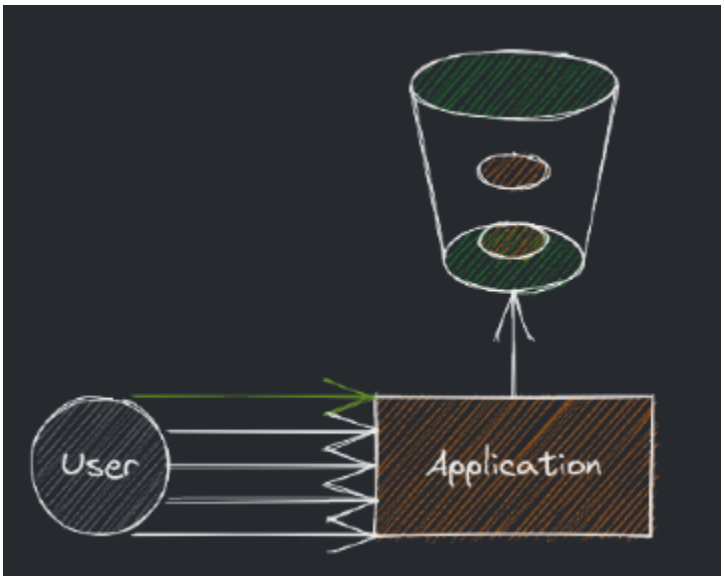
Алгоритм ведро с токенами

- Для контейнера с токенами используется

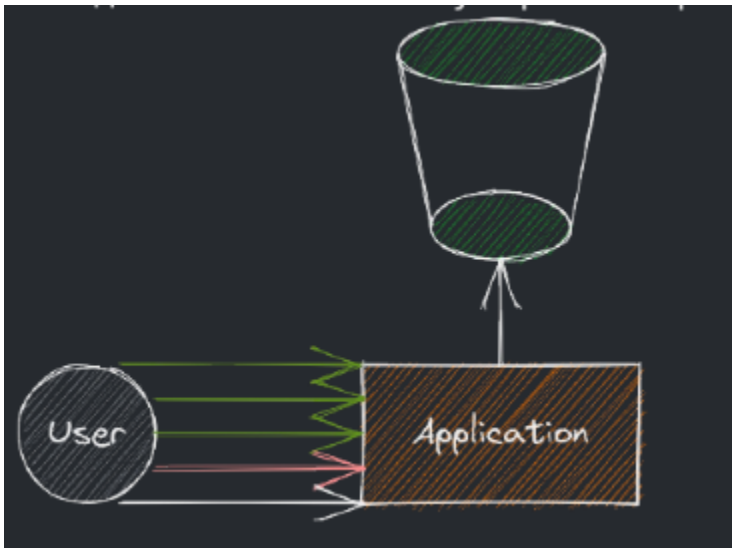
  стек
- Изначально стек заполнен
- Устанавливается периодичность добавления в стек токена
- При каждом запросе от пользователя из стека удаляется один токен
- При исчерпании токенов запросы от пользователя не принимаются до момента пополнения стека новым токеном



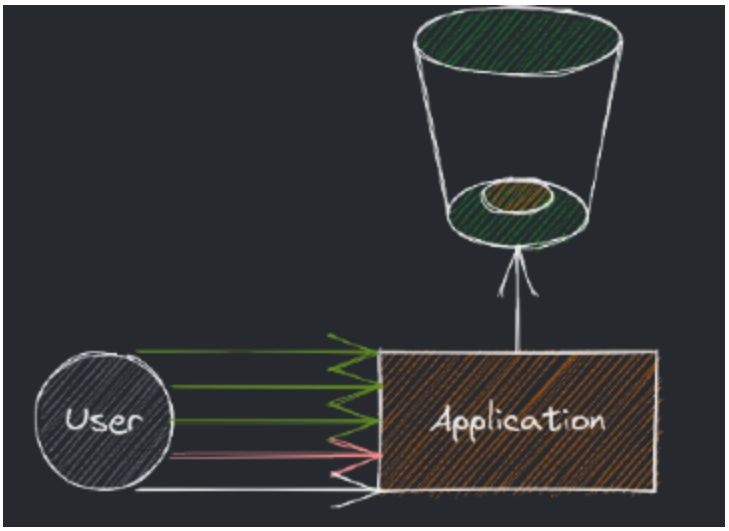
Стек заполнен



Пользователь сделал запрос и один токен сгорел



Когда свободные токены закончились, запрос блокируется

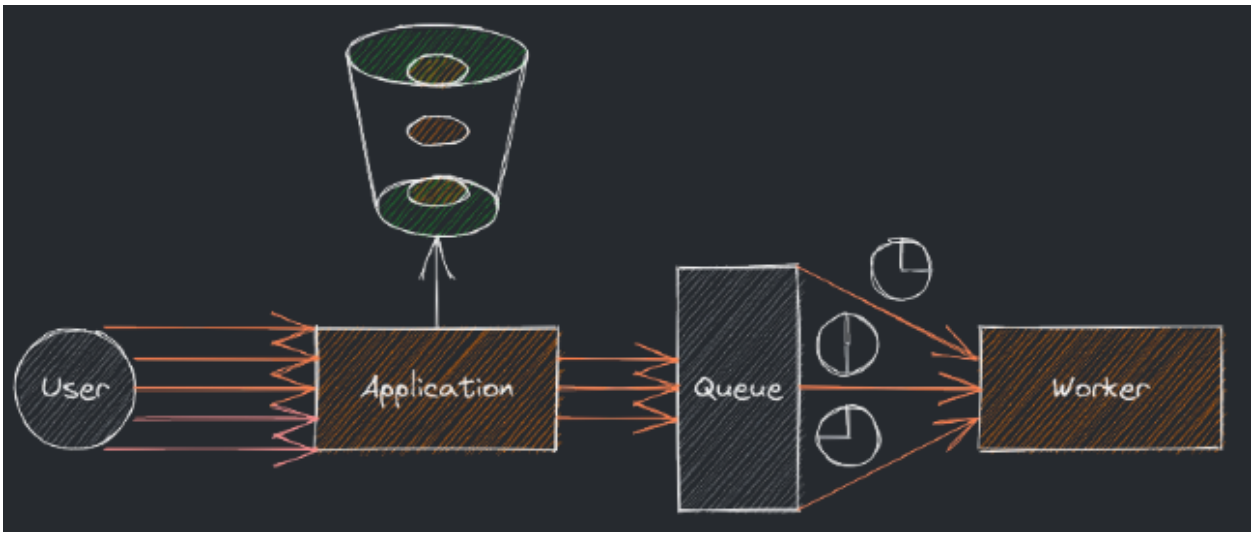


Через установленное время токены начинают восстанавливаться и пользователь сможет отправить запрос

К минусам алгоритма можно отнести тот факт, что если большое количество пользователей в конкретный момент времени начнут тратить свои токены, приложение получит максимальную нагрузку и может сломаться.

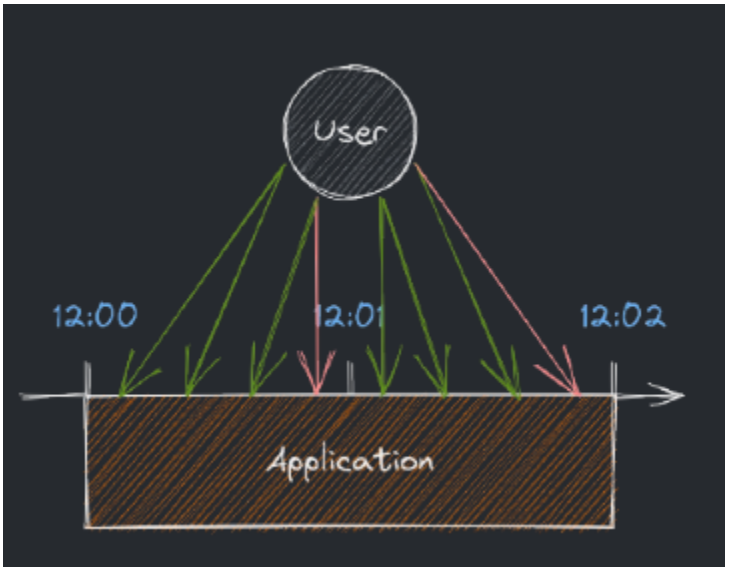
Алгоритм протекающего ведра

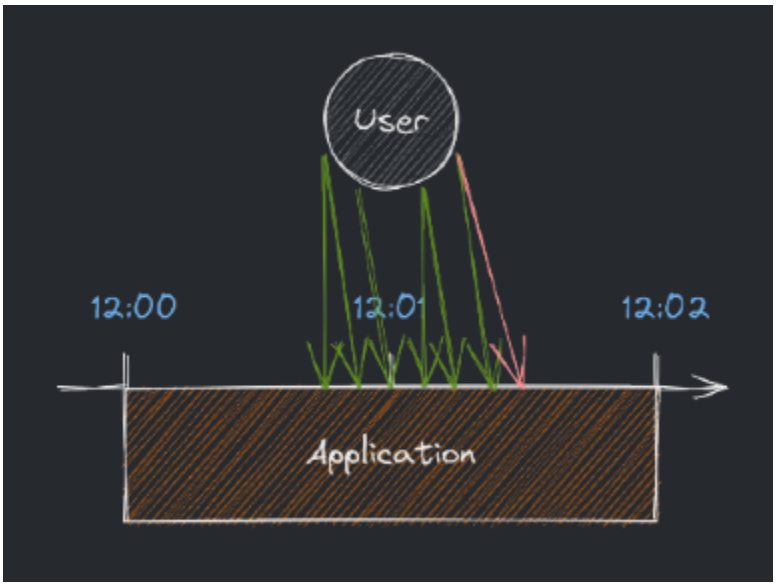
- В приложении также есть стек с токенами
- Дополнительно после стека установлена очередь исполнения. Т.е. запросы прошедшие через стек исполняются не сразу, а устанавливаются в общую очередь и исполняются по мере возможности сервиса



Алгоритм с фиксированным окном

Ограничиваем запросы в равные промежутки времени

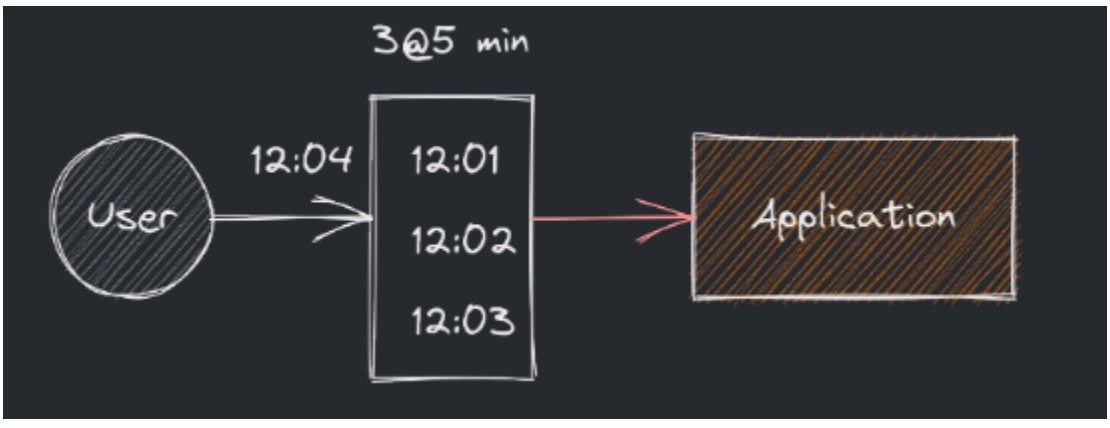




Все-таки предельная нагрузка может достигаться, если все запросы придут на период между двумя промежутками времени

Алгоритм с обновленным логом

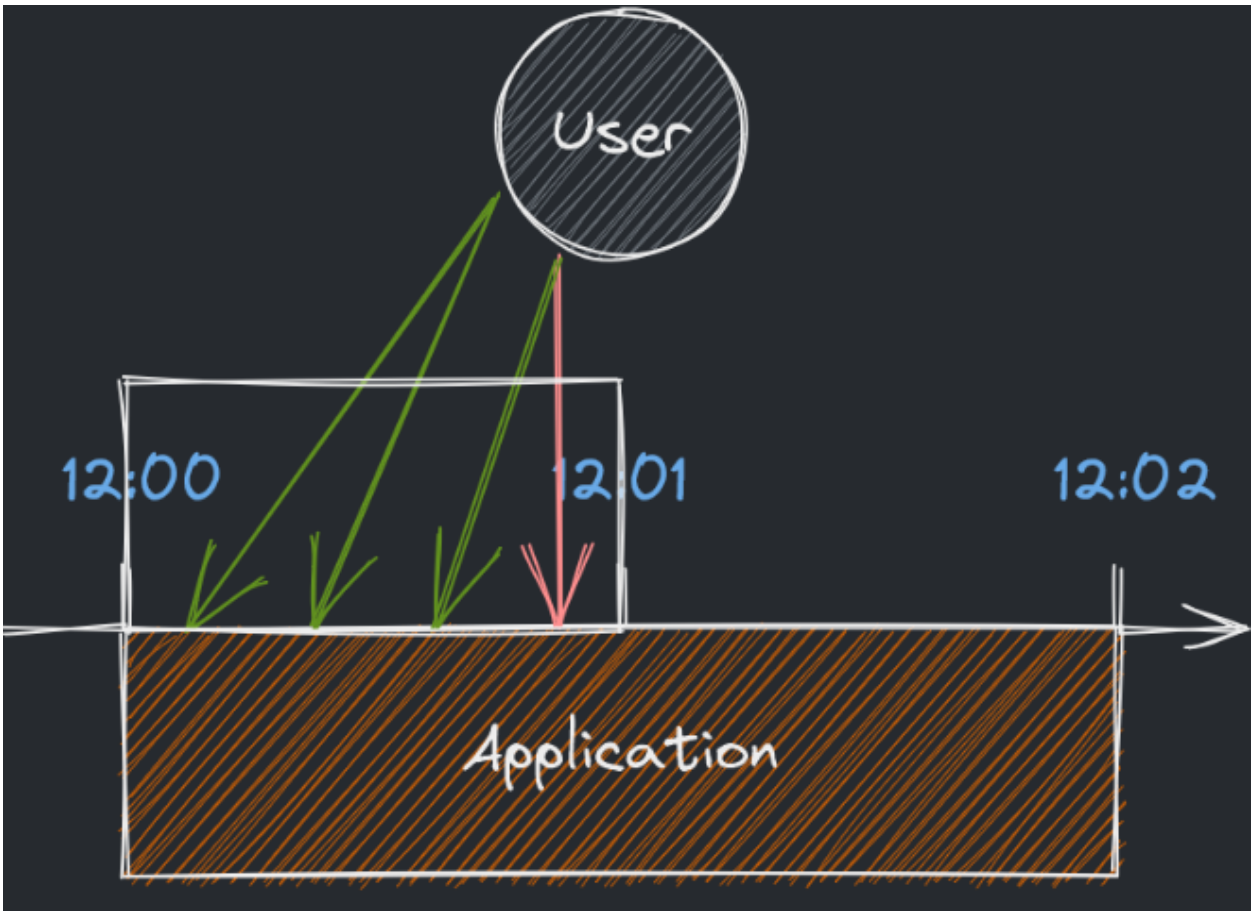
- Логируем каждый успешный запрос от пользователя
- При поступлении нового запроса чистим старые если такие есть
- Если в логе нет места для текущего запроса, пропускаем его

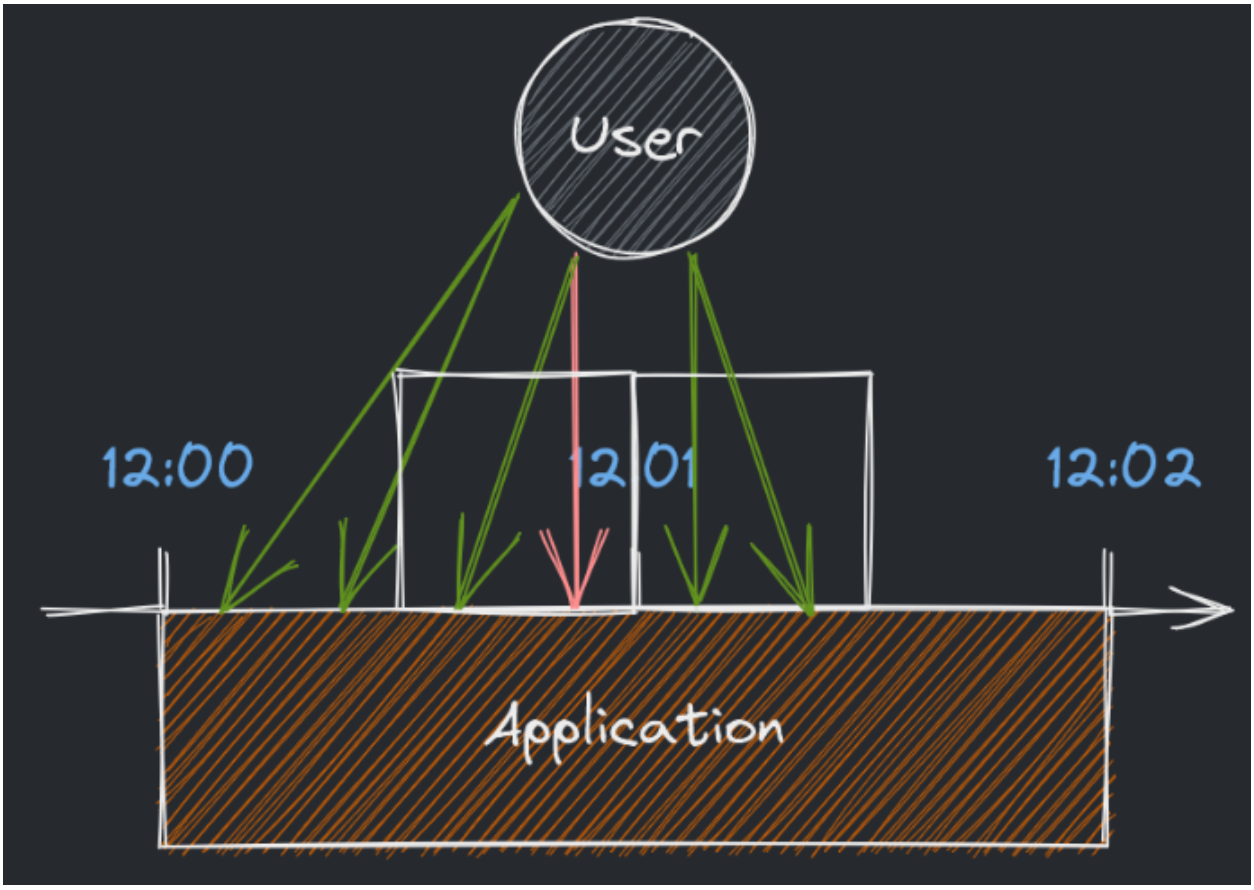


Алгоритм со скользящим окном

- Гибридный подход между фиксированным окном и обновленным логом

- Оценивает количество запросов во временном окне как взвешенную сумму статистик по периодам
- В окне, содержащем 1/3 первой минуты и 2/3 второй, оцениваем запросы как 1/3 * n1 + 2/3 * n2





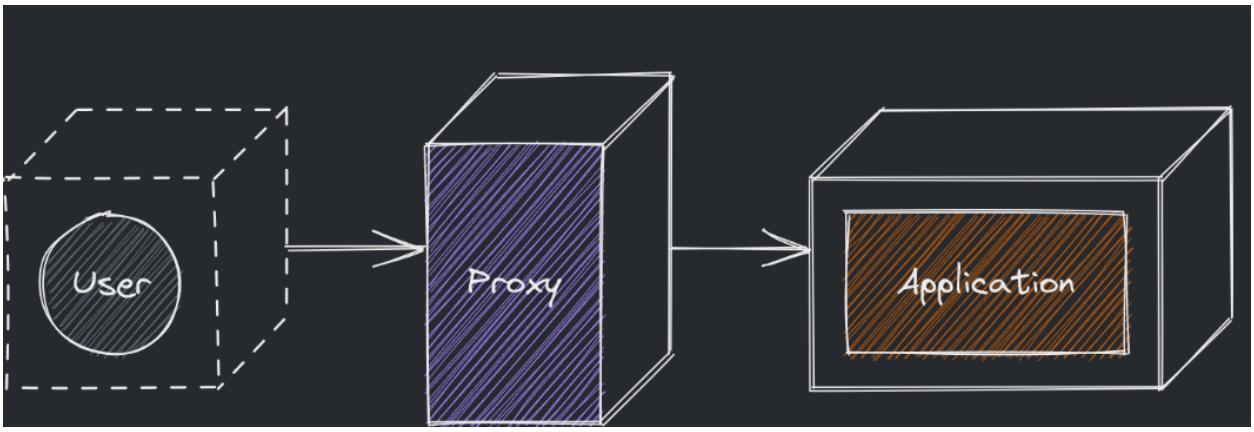
> Модули защиты системы

Способы защиты системы от злоумышленников

- Адреса реальных серверов скрывают за обратными прокси
- Файрвол для ограничения доступа извне

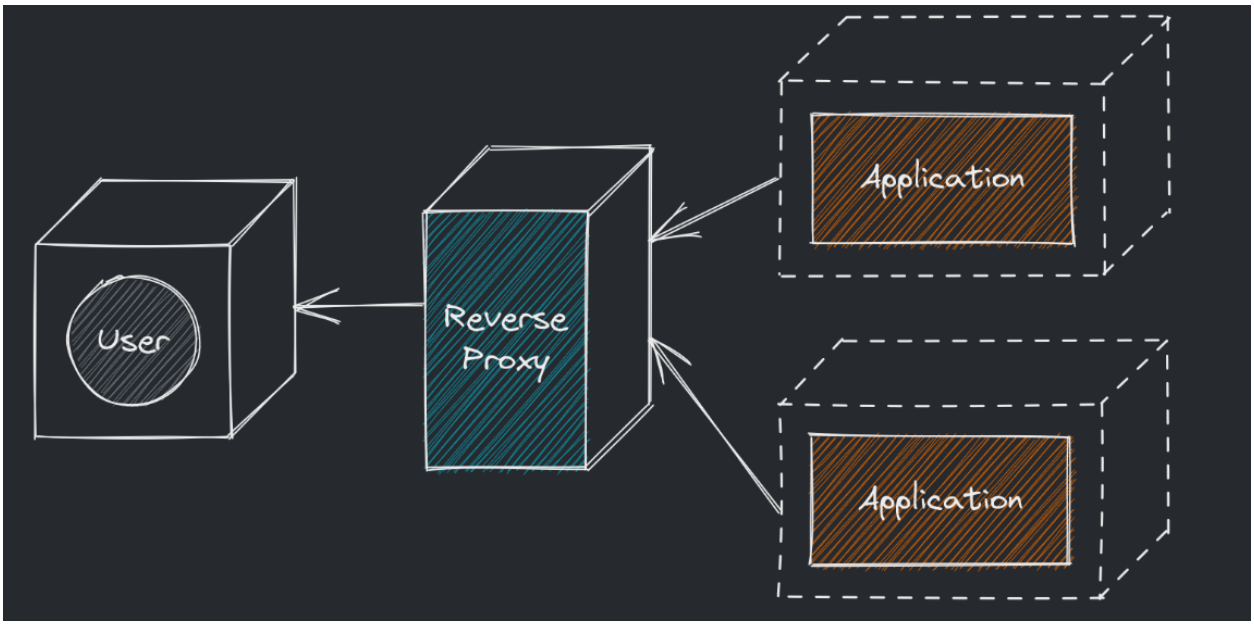
Работа прокси

Обычный прокси используют для сокрытия истинного источника обращения к приложению



Сервис не видит реальный IP пользователя

Обратный прокси-сервер используется для сокрытия реальных серверов, отвечающих на запросы пользователя

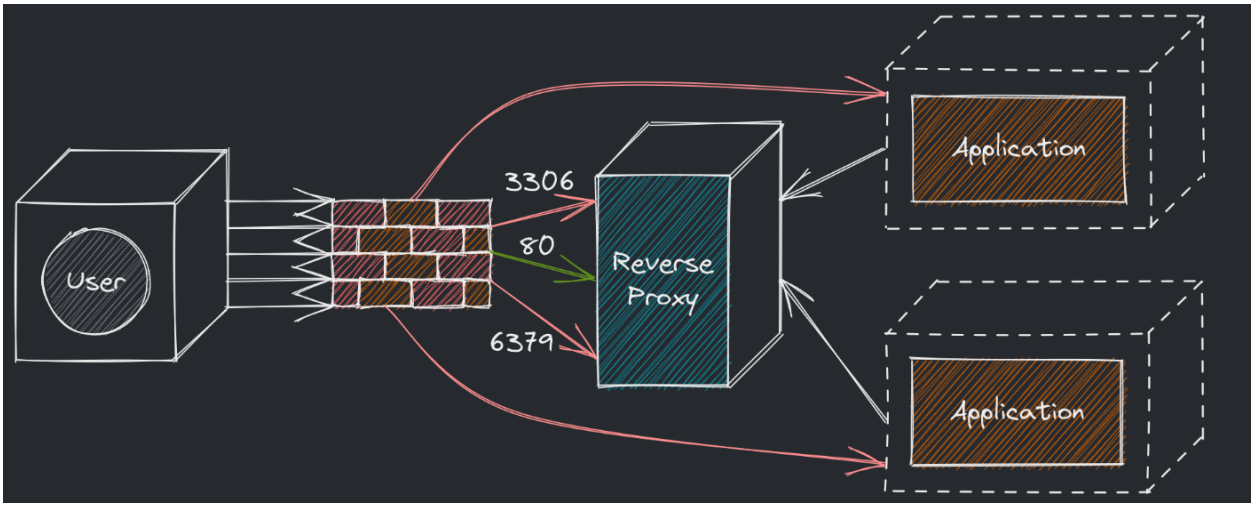


Таким образом, все запросы идут на IP прокси-сервера, при этом адреса настоящих серверов известны только прокси, который переадресует запрос к настоящему серверу, получает ответ и отправляет пользователю.

При этом подходе существует вероятность «уронить» прокси-сервер, но реальные сервера останутся в рабочем состоянии.

Файрвол

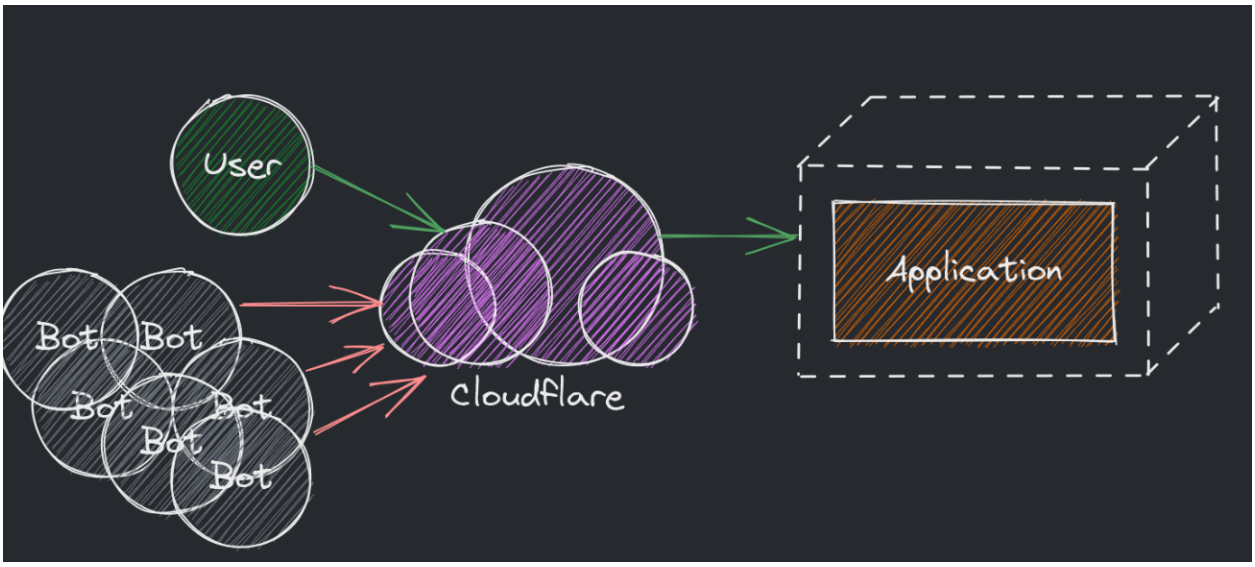
Файрволы ограничивают возможность обработки запросов. В модуле устанавливают конкретные IP-адреса, порты или конкретных пользователей, с которых можно обрабатывать запрос. Если запрос пришел с другого порта или адреса, он не пройдет к серверу.



Типы файрволов:

- Пакетные фильтры — самые простые, сверяют IP, порт, протокол и другие базовые свойства запроса
- Сеансового уровня — соединяются от имени внутренних сервисов, пропускают пакеты только для них
- Инспекторы состояний — контролирует пакеты, сессии и приложения на основе внутренних таблиц разрешения

Есть возможность использовать услуги внешних сервисов, например Cloudflare.



> Подсистемы мониторинга

Кроме балансировщиков нагрузки имеет смысл и самим отслеживать критичные показатели системы:

- Распределение времени ответа тех или иных подсервисов
- Распределение положительных ответов и различных ошибок

В случае наличия серьезных отклонений можно уведомить инженеров или администраторов.

Все показатели стоит сохранять в течении какого-то времени для ретроспективного анализа.

Основные парадигмы отслеживания показателей

1. Самостоятельное логгирование метрик приложения — Graphite
2. Сбор метрик внешними мониторами через API приложения — Prometheus

Grafana — система с удобным интерфейсом для мониторинга показателей. Сбор самих метрик можно осуществлять силами Graphite или Prometheus.

Для сбора метрик в большом приложении можно организовать подсистему логгирования с каждого из сервисов.