

Tinkoff Python

Лекция 10

Работа с памятью. Highload, high availability



Афонасьев Евгений

О чём пойдет речь?

- Как выделяется/освобождается память
 - Подсчет ссылок
 - GC
-
- Типичные проблемы масштабирования
 - Что такое высокая доступность
 - Нагрузочное тестирование



PyObject

```
typedef struct _object {
    Py_ssize_t ob_refcnt;
    struct _typeobject *ob_type;
} PyObject;
```

<https://github.com/python/cpython/blob/f1ec3cefad4639797c37eaa8c074830188fa0a44/Include/object.h#L109>

Так выглядит float object

```
typedef struct {
    PyObject ob_base;
    double ob_fval;
} PyFloatObject;
```

[https://github.com/python/cpython/blob/e42b705188271da108
de42b55d9344642170aa2b/Include/floatobject.h#L15](https://github.com/python/cpython/blob/e42b705188271da108de42b55d9344642170aa2b/Include/floatobject.h#L15)

A tak function object

```
typedef struct {
    PyObject ob_base;
    PyObject *func_code;           /* A code object, the __code__ attribute */
    PyObject *func_globals;        /* A dictionary (other mappings won't do) */
    PyObject *func_defaults;       /* NULL or a tuple */
    PyObject *func_kwdefaults;     /* NULL or a dict */
    PyObject *func_closure;        /* NULL or a tuple of cell objects */
    PyObject *func_doc;            /* The __doc__ attribute, can be anything */
    PyObject *func_name;           /* The __name__ attribute, a string object */
    PyObject *func_dict;           /* The __dict__ attribute, a dict or NULL */
    PyObject *func_weakreflist;    /* List of weak references */
    PyObject *func_module;          /* The __module__ attribute, can be anything */
    PyObject *func_annotations;    /* Annotations, a dict or NULL */
    PyObject *func_qualname;        /* The qualified name */
} PyFunctionObject;
```

[https://github.com/python/cpython/blob/e42b705188271da108
de42b55d9344642170aa2b/Include/funcobject.h#L21](https://github.com/python/cpython/blob/e42b705188271da108de42b55d9344642170aa2b/Include/funcobject.h#L21)

Все объекты обернуты в PyObject
структуру и могут хранить
дополнительные метаданные

Memory management

В python есть несколько схем выделения памяти, которые выбираются в зависимости от объема нужной памяти.

<https://rushter.com/blog/python-memory-managment/>

Запрос памяти у OS - относительно медленная
операция, особенно если запрашивать много
маленьких кусочков вместо меньшего кол-ва
больших

Для объектов, которым нужно
больше 512 байт, вызываются
напрямую `malloc/free` без
дополнительных оптимизаций

Остальные объекты хранятся в блоках

блок - это область памяти с
фиксированным размером кратным 8

Объект получает блок с
МИНИМАЛЬНО
необходимым размером

Например: если объекту нужно 13Б,
берем блок на 16Б

Request in bytes	Size of allocated block	size class idx
1-8	8	0
9-16	16	1
17-24	24	2
25-32	32	3
33-40	40	4
41-48	48	5
...
505-512	512	63

Блоки сгруппированы в пулы (pool) по 4кБ

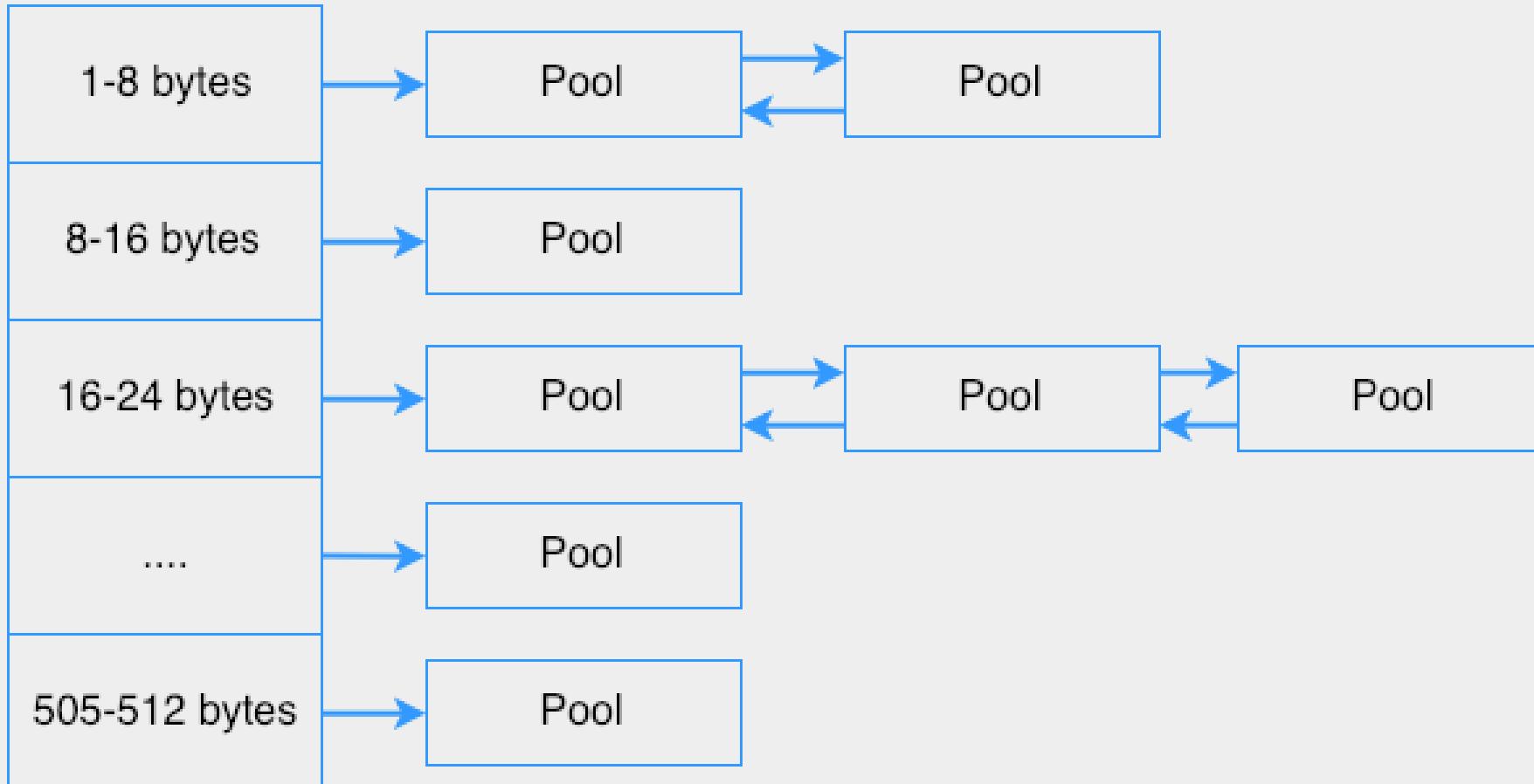
- Пул состоит из блоков одинакового размера (помогает избежать фрагментации)
- Пул помнит какие блоки в нем заняты, а какие нет.
- Если объект уничтожается, занимаемый им блок помечается как свободный и может быть использован заново.

pool_header

```
struct pool_header {
    union { block *_padding;
            uint count; } ref;
    block *freeblock;
    struct pool_header *nextpool;
    struct pool_header *prevpool;
    uint arenaindex;
    uint szidx;
    uint nextoffset;
    uint maxnextoffset;
};
```

/* number of allocated blocks */
/* pool's free list head */
/* next pool of this size class */
/* previous pool */
/* index into arenas of base adr */
/* block size class index */
/* bytes to virgin block */
/* largest valid nextoffset */

usedpools



Пулы группируются в арены (arena) по 256kB,
каждая аrena содержит в себе 64 пула

Arena		
Pool (4kB)	Pool (4kB)	Pool (4kB)
Pool (4kB)	Pool (4kB)	Pool (4kB)
Pool (4kB)	Pool (4kB)	Pool (4kB)
Pool (4kB)	Pool (4kB)	Free Pool (4kB)
Free Pool (4kB)	Free Pool (4kB)	Free Pool (4kB)
...

arena_object

```
struct arena_object {
    uintptr_t address;
    block* pool_address;
    uint nfreepools;
    uint ntotalpools;
    struct pool_header* freepools;
    struct arena_object* nextarena;
    struct arena_object* prevarena;
};
```

Python запрашивает память у
ОС только целыми аренами,
чтобы уменьшить кол-во
выделений памяти

Освобождается ли
память обратно?

Только если все блоки
всех пулов арены пусты



sys._debugmallocstats()

class	size	num pools	blocks in use	avail blocks
0	8	2	801	211
1	16	2	394	112
2	24	4	509	163
3	32	48	6024	24
4	40	99	9944	55
5	48	69	5686	110
...				
# arenas allocated	total		=	170
...				
76 arenas * 262144 bytes/arena			=	19,922,944
...				
Total			=	19,922,944
14 free PyCFunctionObjects * 48 bytes each	=	672		
72 free PyDictObjects * 48 bytes each	=	3,456		
3 free PyFloatObjects * 24 bytes each	=	72		
8 free PyFrameObjects * 368 bytes each	=	2,944		
...				

sys.getsizeof

```
In [1]: import sys
```

```
In [2]: sys.getsizeof([])  
Out[2]: 64
```

```
In [3]: sys.getsizeof([1,2,3])  
Out[3]: 88
```

```
In [4]: sys.getsizeof({})  
Out[4]: 240
```

<https://docs.python.org/3/library/sys.html#sys.getsizeof>

____dict____

```
In [15]: class X:  
...:     pass  
...:  
...:
```

```
In [16]: X().__dict__  
Out[16]:  
mappingproxy({'_module_': '__main__',  
             '_dict_': <attribute '__dict__' of 'X' objects>,  
             '_weakref_': <attribute '__weakref__' of 'X' objects>,  
             '_doc_': None})
```

```
In [17]: X().__dict__  
Out[17]: {}
```

В памяти питона словари
повсюду!

`__slots__` позволяет сэкономить немного памяти

```
In [1]: class X:  
...:     __slots__ = ('a', )  
...:     def __init__(self, a):  
...:         self.a = a  
...:  
...:  
In [2]: X(1).__dict__  
...  
AttributeError: 'X' object has no attribute '__dict__'
```

<https://docs.python.org/3/reference/datamodel.html#slots>

ИТОГО

- Питон использует сложную систему выделения памяти, чтобы оптимизировать кол-во аллокаций для небольших объектов, при этом память не всегда освобождается обратно.
- Больше объекты честно запрашивают и отдают память обратно.
- Иногда мы можем влиять на объем памяти, занимаемый нашими объектами.

Сборка мусора



<https://rushter.com/blog/python-garbage-collector/>

Reference counting

Подсчёт ссылок также известен как один из алгоритмов сборки мусора, где каждый объект содержит счётчик количества ссылок на него, используемых другими объектами. Когда этот счётчик уменьшается до нуля, это означает, что объект стал недоступным, и он помещается в список объектов на уничтожение.

PyObject

```
typedef struct _object {
    Py_ssize_t ob_refcnt; # счетчик ссылок
    struct _typeobject *ob_type;
} PyObject;
```

Счетчик увеличивается

- Создается переменная, указывающая на объект
- Объект записывается в атрибут
- Объект передается как аргумент в функцию
- Объект кладется в коллекцию
- etc.

Счетчик уменьшается

- Удаляется переменная/атрибут
- Заканчивается выполнение функции
- Удаляется коллекция
- etc.

Объект удаляется из памяти, когда счетчик
ссылок опустится до 0

Удаление одного объекта может
каскадно привести к удалению других!

sys.getrefcount

```
foo = [ ]  
  
# 2 references, 1 from the foo var and 1 from getrefcount  
print(sys.getrefcount(foo))  
  
def bar(a):  
    # 4 references  
    # from the foo var, function argument,  
    # getrefcount and Python's function stack  
    print(sys.getrefcount(a))  
  
bar(foo)  
# 2 references, the function scope is destroyed  
print(sys.getrefcount(foo))
```

<https://docs.python.org/3/library/sys.html#sys.getrefcount>

Итого

- Каждый объект имеет счетчик ссылок
- Если счетчик достигает нуля, объект удаляется из памяти

Но есть случаи, в которых
недостаточно подсчета ссылок

Циклические ссылки

```
object_1 = {}
object_2 = {}
object_1['obj2'] = object_2
object_2['obj1'] = object_1
del object_1, object_2
```

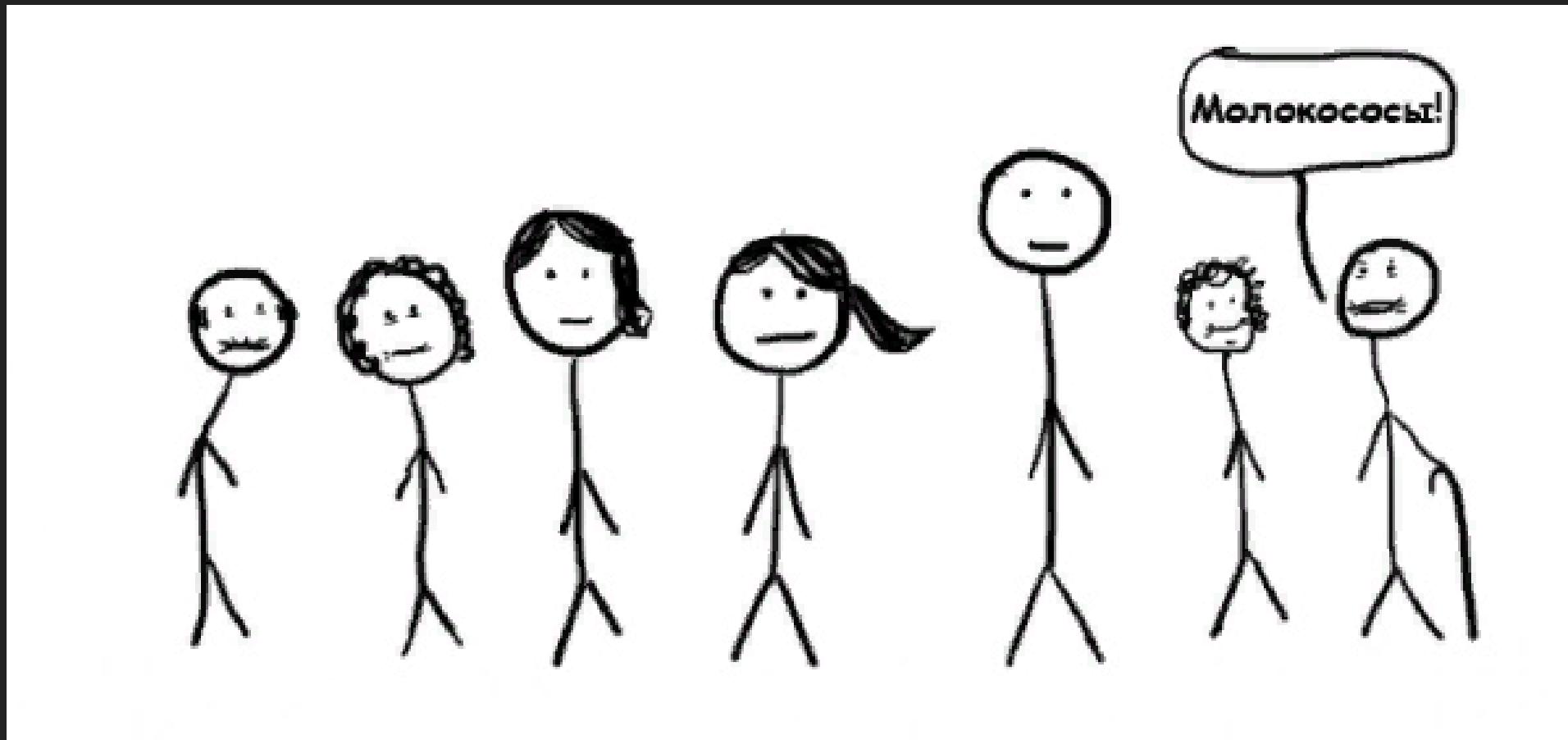
gc

Специальный процесс, называемый сборщиком мусора (англ. garbage collector), периодически освобождает память, удаляя объекты, которые уже не будут востребованы приложениями.

<https://docs.python.org/3/library/gc.html>

Нужен для обработки циклических ссылок,
поэтому отслеживает ТОЛЬКО объекты, в
которых могут быть ссылки (dict, object, etc.),
но не примитивные типы (int, str, etc.)

Все объекты делятся на три поколения



Новые объекты попадают в
первое

У каждого поколения есть **счетчик**,
который увеличивается при добавлении
объекта в поколение и уменьшается при
удалении

После создания любого контейнерного объекта проверяется, достиг ли **счетчик** некоторого порогового значения (**700, 10, 10** по умолчанию), и запускается сборка для соответствующего поколения

Пороги можно менять

но в очень редких случаях

`gc.get_threshold`

`gc.set_threshold`

Объекты, пережившие сборку мусора, переходят в следующее поколение. После чего счетчик для поколения обнуляется.

Переход объектов в следующее поколение может сразу же вызвать сборку и в нем!

Как gc находит циклические ссылки?

Обходит граф объектов в памяти и помечает все объекты, которые могут быть достигнуты от некоторого узла, остальные считает недостижимыми и удаляет

<http://arctrix.com/nas/python/gc/>

`__del__`

```
In [1]: class X:  
...:     def __del__(self):  
...:         print('del x')  
...:
```

```
In [2]: x = X()
```

```
In [3]: del x  
del x
```

https://docs.python.org/3/reference/datamodel.html#object.__del__

`__del__`

Может быть вызван при сборке мусора,
но на него не стоит полагаться как на
надежный механизм закрытия
ресурсов. Лучше использовать
контекстные менеджеры.

weakref (не учитываются как ссылки)

```
In [10]: x = X()
```

```
In [11]: ref = weakref.ref(x)
```

```
In [12]: print(ref())
<__main__.X object at 0x102bdd668>
```

```
In [13]: del x
del x
```

```
In [14]: print(ref())
None
```

Можно вручную запустить gc
через `gc.collect()`

Отладка gc

```
import gc

gc.set_debug(gc.DEBUG_SAVEALL)

print(gc.get_count())
lst = []
lst.append(lst)
list_id = id(lst)
del lst
gc.collect()
for item in gc.garbage:
    print(item)
    assert list_id == id(item)
```

Подсчет ссылок не
отключаем

Для всего остального
есть `gc.disable()`

memory_profiler

Line #	Mem usage	Increment	Line Contents
=====			
3			@profile
4	5.97 MB	0.00 MB	def my_func():
5	13.61 MB	7.64 MB	a = [1] * (10 ** 6)
6	166.20 MB	152.59 MB	b = [2] * (2 * 10 ** 7)
7	13.61 MB	-152.59 MB	del b
8	13.61 MB	0.00 MB	return a

<https://pypi.org/project/memory-profiler/>

Перерыв?

Какие метрики описывают
состояние нашей системы?

availability

RPS - requests per second

Errors per second (400, 500,
etc.)

Latency

Processing time

Для асинхронных запросов тоже важно
знать время их обработки

CPU/memory/IO etc.

Что такое Highload?



Что такое Highload?

У каждого он свой. Если ваша система перестает показывать удовлетворительные показатели при достижении некоторого объема нагрузки, считайте, что у вас уже маленький highload =)

Bottleneck

Бутылочное горлышко. Конкретная часть системы, которая не справляется с нагрузкой, и тормозит все остальное.

Любые оптимизации должны начинаться именно с поиска bottleneck.



Что такое high availability?

Способность системы продолжать корректную работу при отказе части компонентов. Обычно достигается множественным резервированием рабочих систем (failover).

Важно представлять возможные
точки отказа и вероятность их
падения

В реальности невозможно
построить системы надежные и
быстрые в 100% случаев

SLA - Service Level Agreement

Формальный договор между заказчиком услуги и её поставщиком, содержащий описание услуги, права и обязанности сторон и, самое главное, согласованный уровень качества предоставления данной услуги.

Например мы можем сказать, что наш сервис отдаст ответ за 200мс в 99% случаев, и что он будет доступен 99.9% времени.

https://ru.wikipedia.org/wiki/Соглашение_об_уровне_услуг

Gracefull degradation (для бэка)

Способность системы продолжать работать при полном или частичном отказе части системы, пусть даже с уменьшением показателей.

Например, мы можем просто не показывать рекламу, если упал сервис рекламы, вместо полной недоступности системы.

Каждый компонент системы можно рассматривать как отдельную систему со своими гарантиями. Мы можем масштабировать и повышать устойчивость наших бд, очередей событий, файловых хранилищ etc. , а не только непосредственно прикладных сервисов

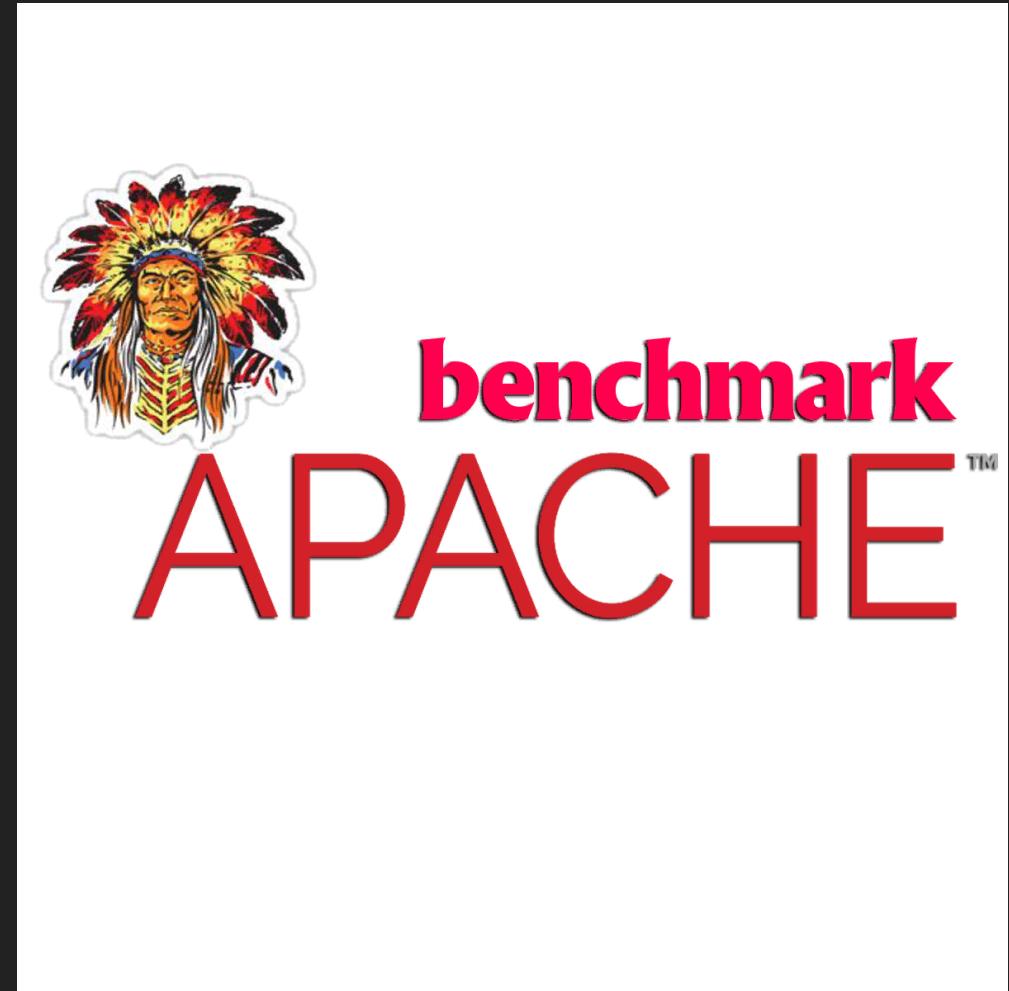
Нагрузочное тестирование



Нагрузочное тестирование

Подвид тестирования производительности, сбор показателей и определение производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе

Для простых
запросов



Шлем 2000 запросов в 200
параллельных соединений

```
ab -n 2000 -c 200 https://service/api/users
```

Concurrency Level: 200
Time taken for tests: 35.902 seconds
Complete requests: 2000
Failed requests: 7
 (Connect: 0, Receive: 0, Length: 7, Exceptions: 0)
Non-2xx responses: 1994
Total transferred: 642068 bytes
HTML transferred: 327016 bytes
Requests per second: 55.71 [#/sec] (mean)
Time per request: 3590.243 [ms] (mean)
Time per request: 17.951 [ms] (mean, across all concurrent requests)
Transfer rate: 17.46 [Kbytes/sec] received

Connection Times (ms)

	min	mean	[+/-sd]	median	max
Connect:	0	1412	1519.1	1134	21320
Processing:	53	448	721.2	294	9093
Waiting:	0	389	615.5	226	9093
Total:	217	1861	1700.0	1498	21508

Percentage of the requests served within a certain time (ms)

50%	1498
66%	1928
75%	2328
80%	2659
90%	3743
95%	5223
98%	6842
99%	8987
100%	21508 (longest request)

Те самые перцентили для SLA



LOCUST

Для сложных
сценариев

```
from locust import HttpLocust, TaskSet, task, between

class WebsiteTasks(TaskSet):
    def on_start(self):
        self.client.post("/login", {
            "username": "test_user",
            "password": ""
        })

    @task
    def index(self):
        self.client.get("/")

    @task
    def about(self):
        self.client.get("/about/")

class WebsiteUser(HttpLocust):
    task_set = WebsiteTasks
    wait_time = between(5, 15)
```

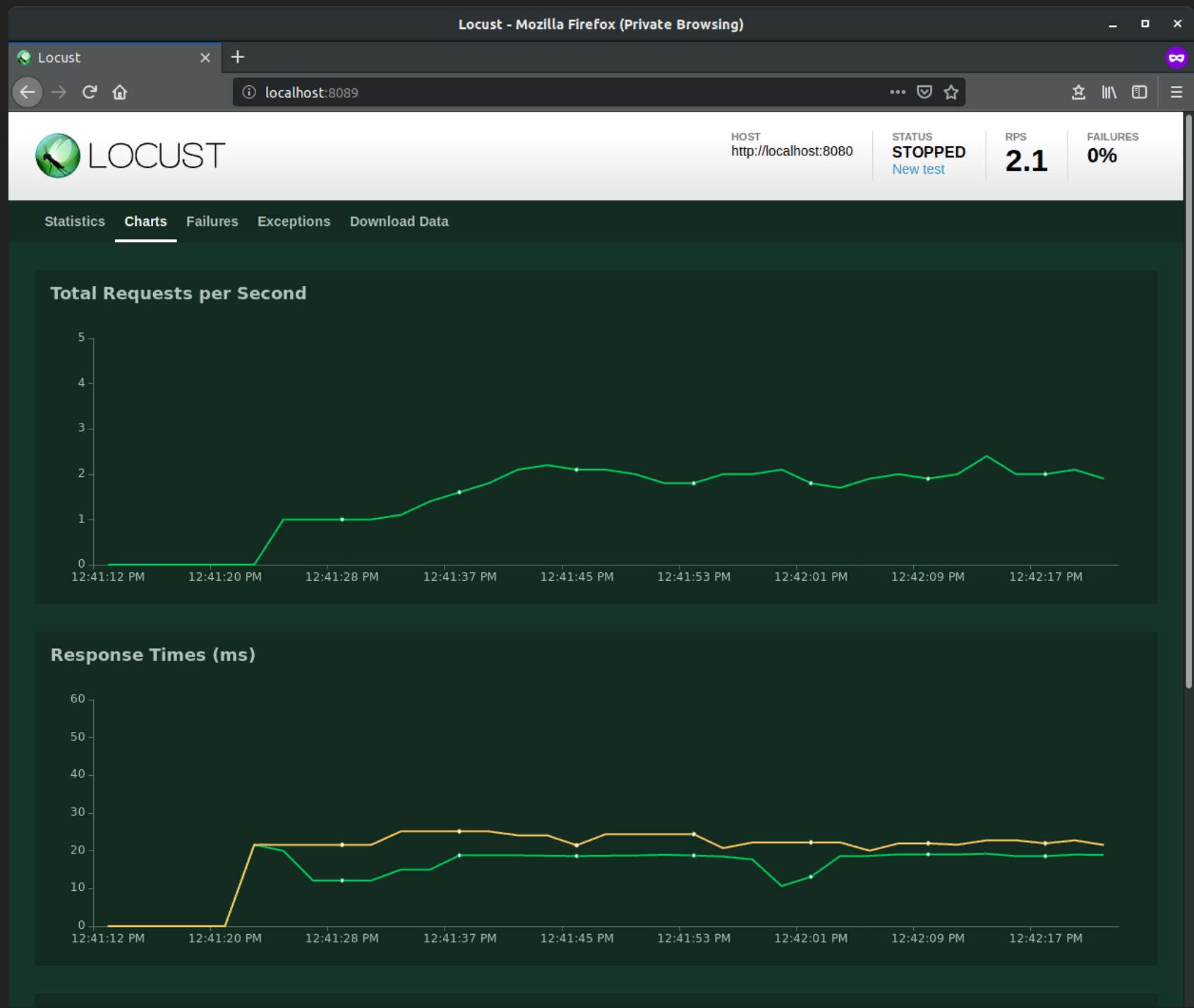
```
$ locust -f locustfile.py
```



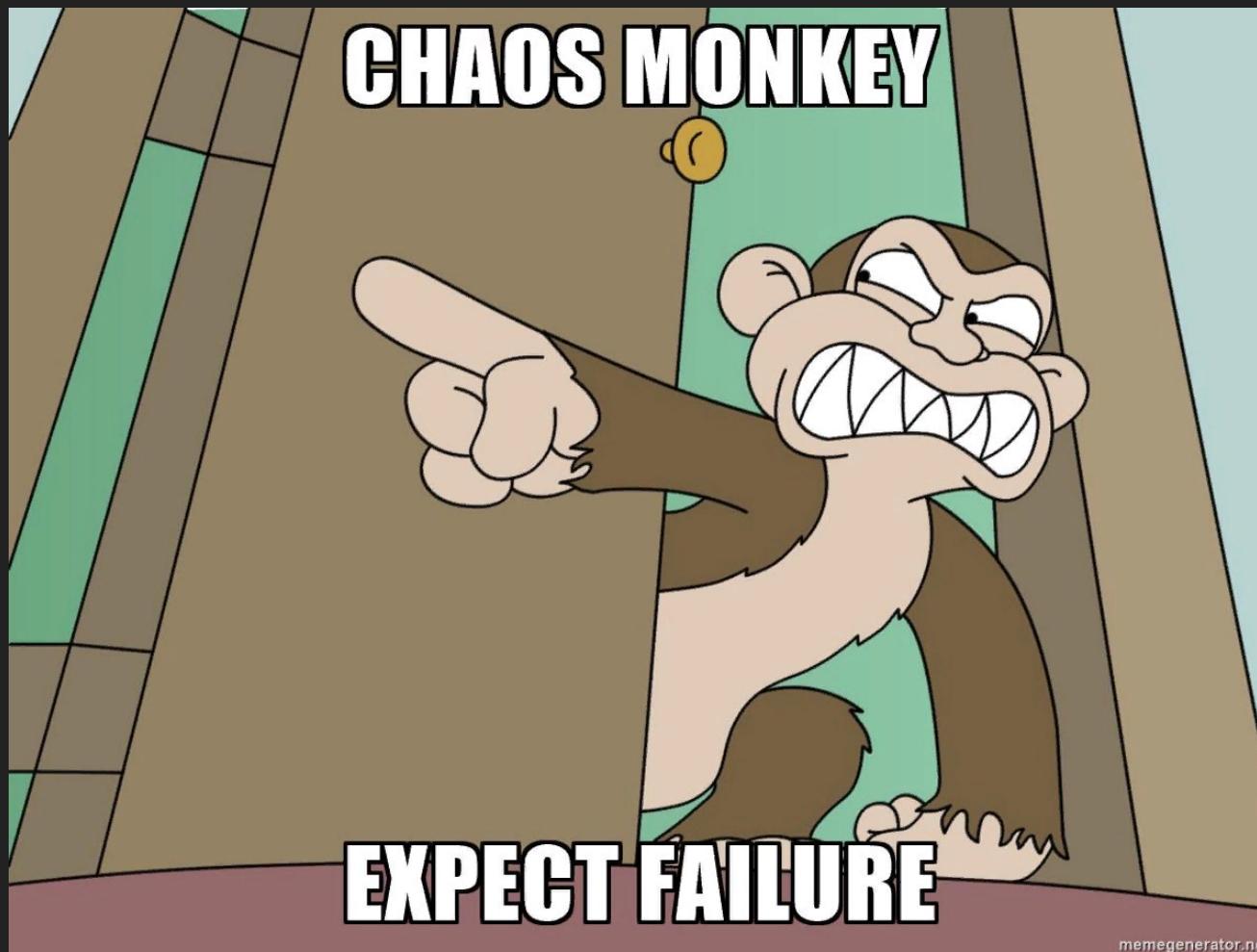
LOCUST

HOST
http://api.initech.comSTATUS
RUNNING
21400 users
[Edit](#)SLAVES
6RPS
240FAILURES
0%**STOP****Reset Stats**[Statistics](#) [Charts](#) [Failures](#) [Exceptions](#) [Download Data](#) [Slaves](#)

Type	Name	# Requests	# Fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
GET	/	5416	0	21	21	4	38	20336	44.1
GET	/blog	1745	0	27	26	3	49	20370	13.7
GET	/blog/[post-slug]	1824	0	15	15	2	27	19943	15.9
POST	/groups/create	185	0	57	55	5	108	3273	1.9
GET	/signin	10266	0	26	26	3	49	19949	66.6
POST	/signin	10266	0	82	82	45	120	20030	66.6
GET	/users/[username]	1802	0	31	31	6	55	20194	15



Тестирование на отказоустойчивость



memegenerator.net

Тестирование на отказоустойчивость

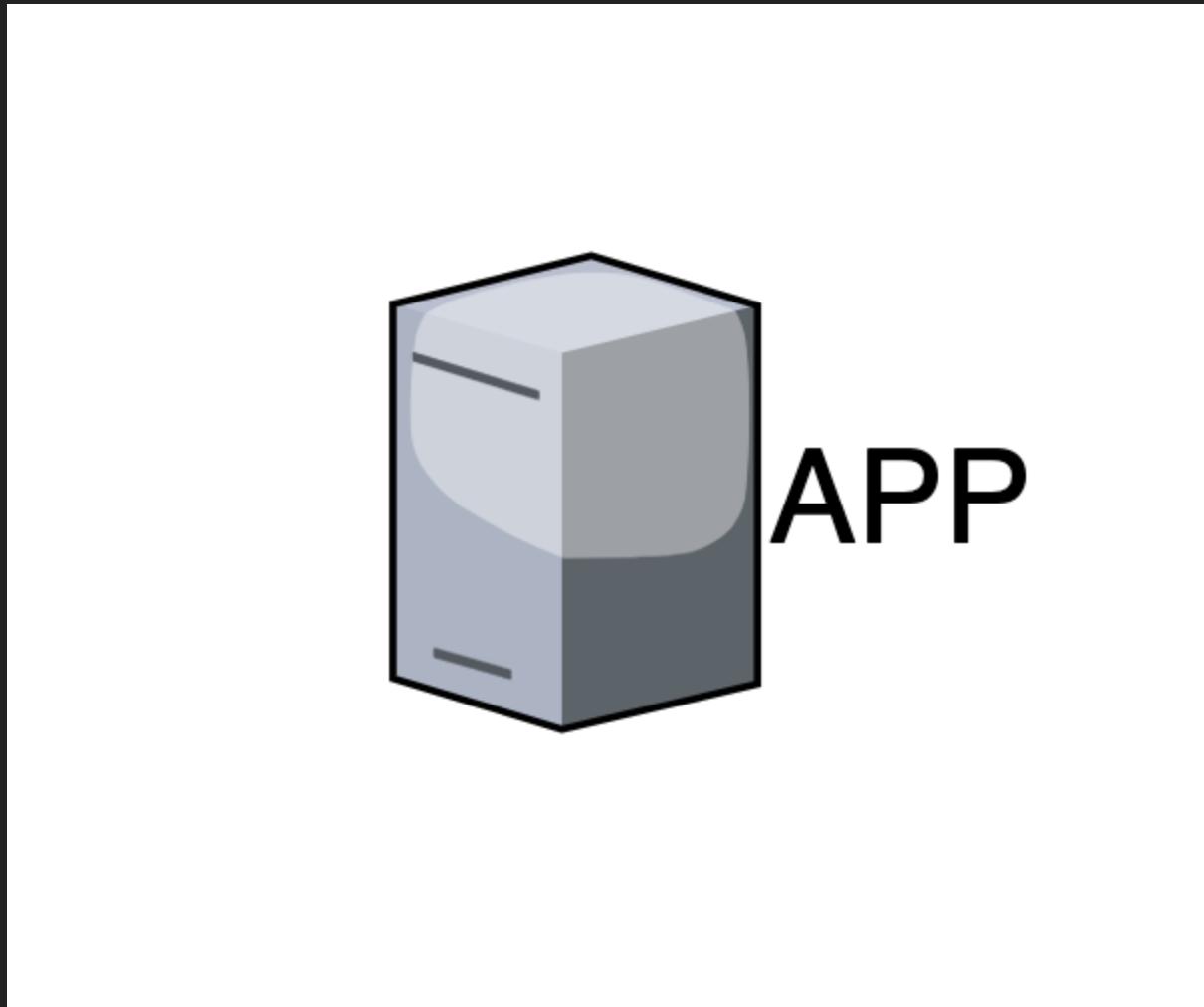
Выключаем те или иные компоненты системы (реплики бд, брокеры сообщений, контейнеры с сервисами, сеть между сервисами etc.) и следим, как изменятся наши показатели, останется ли система работоспособной.

<https://habr.com/ru/company/flant/blog/460367/>

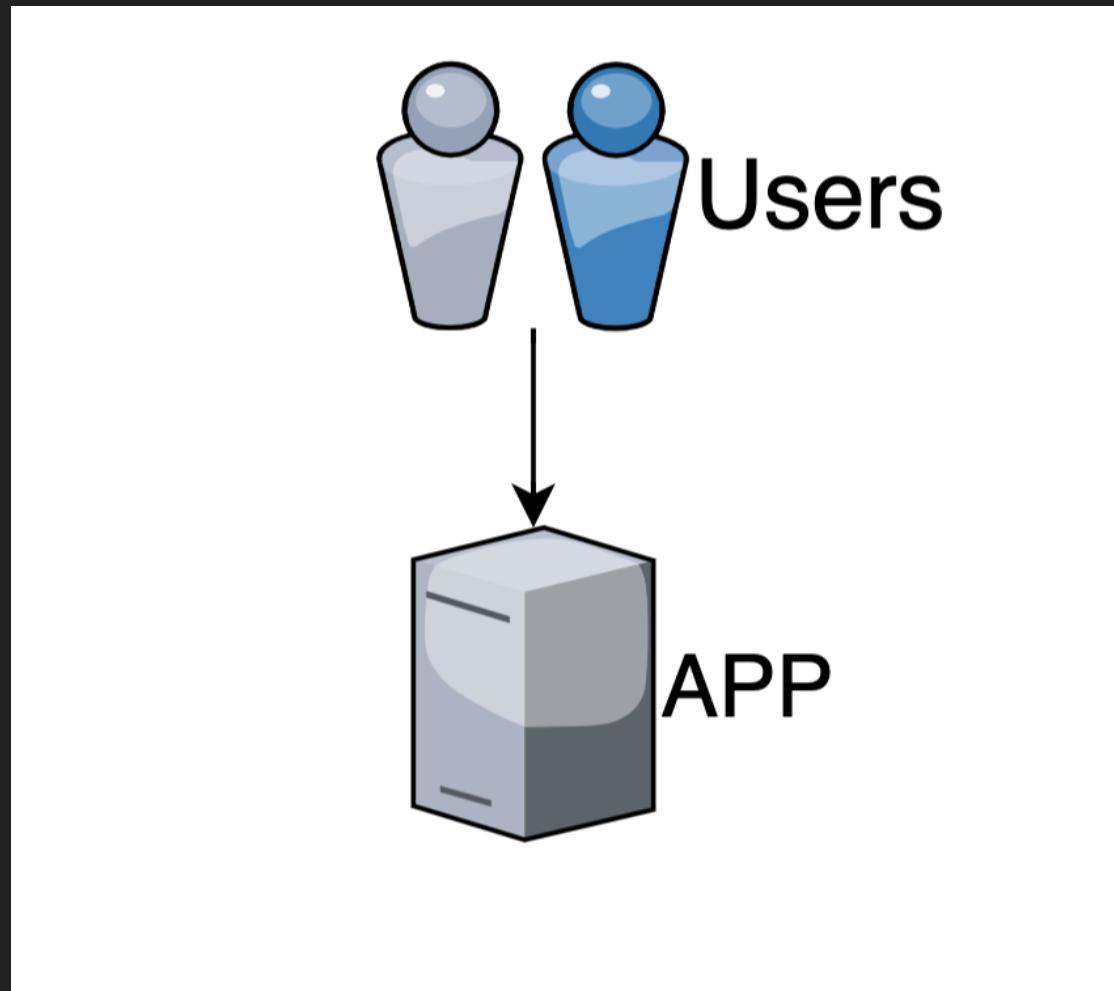
Иногда такие "учения" бывают в рамках всей компании. Например отключение на время одного дц, чтобы увидеть, какие системы это не пережили

Как растут сервисы по
мере развития?

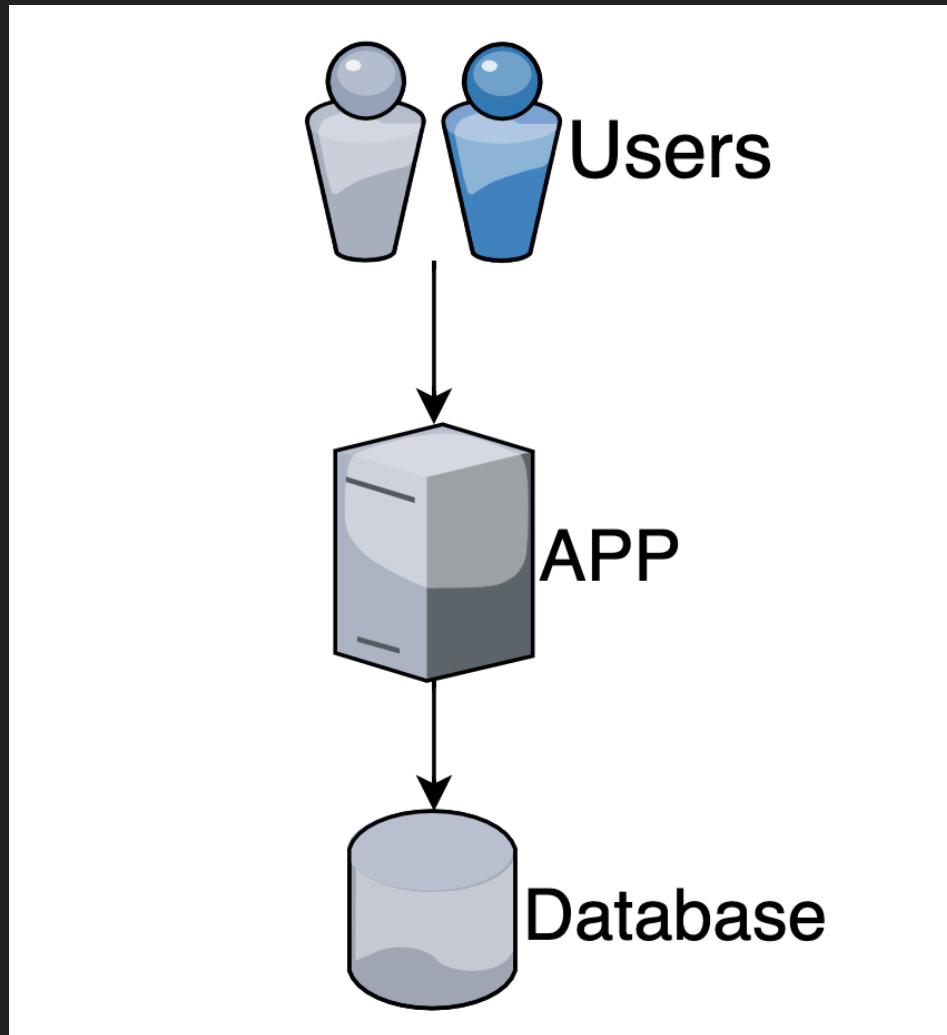
У нас есть приложение



В которое ходят клиенты



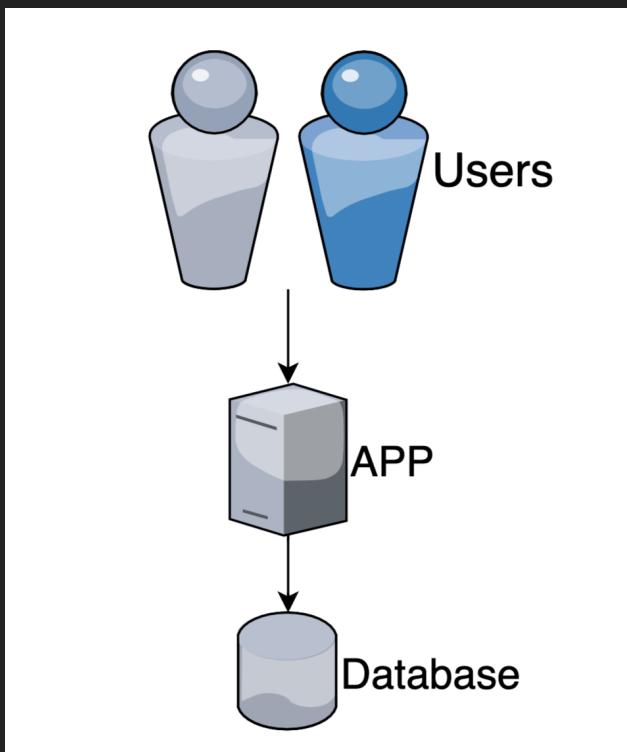
Данные читаем из базы (РСУБД)



Число одновременных
запросов растет по мере
роста популярности системы

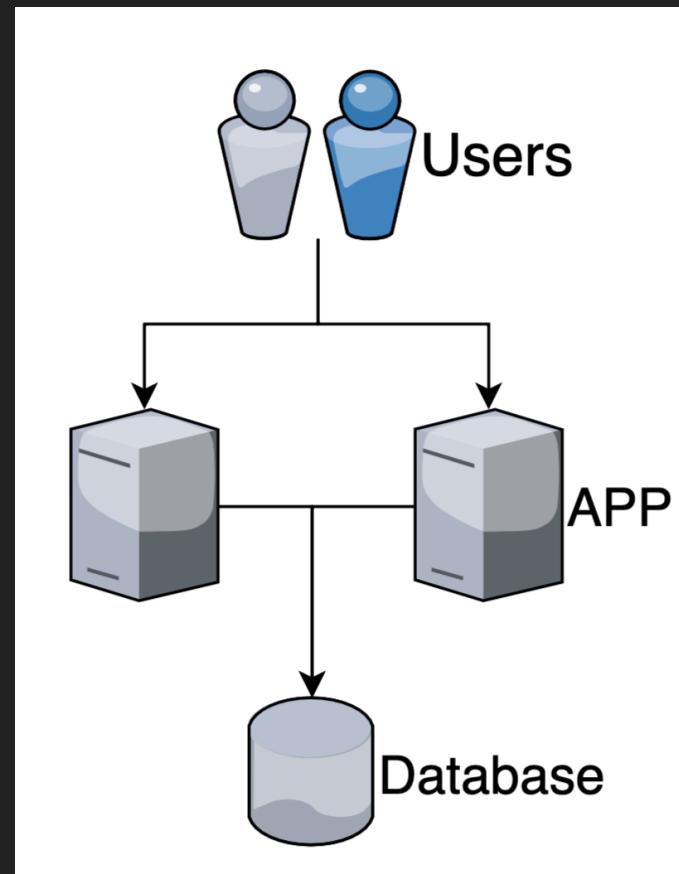
С увеличением нагрузки увеличиваем кол-во процессов/потоков для обработки запросов и/или переходим на асинхронную обработку іо.

Performance +



Но рано или поздно нам придется
поставить второй сервер/виртуалку
и добавить новые инстансы
приложения

Performance +
High availability +



Балансировка

▶ Web Server / Load Balancer



Http Web Access



NGINX



IIS

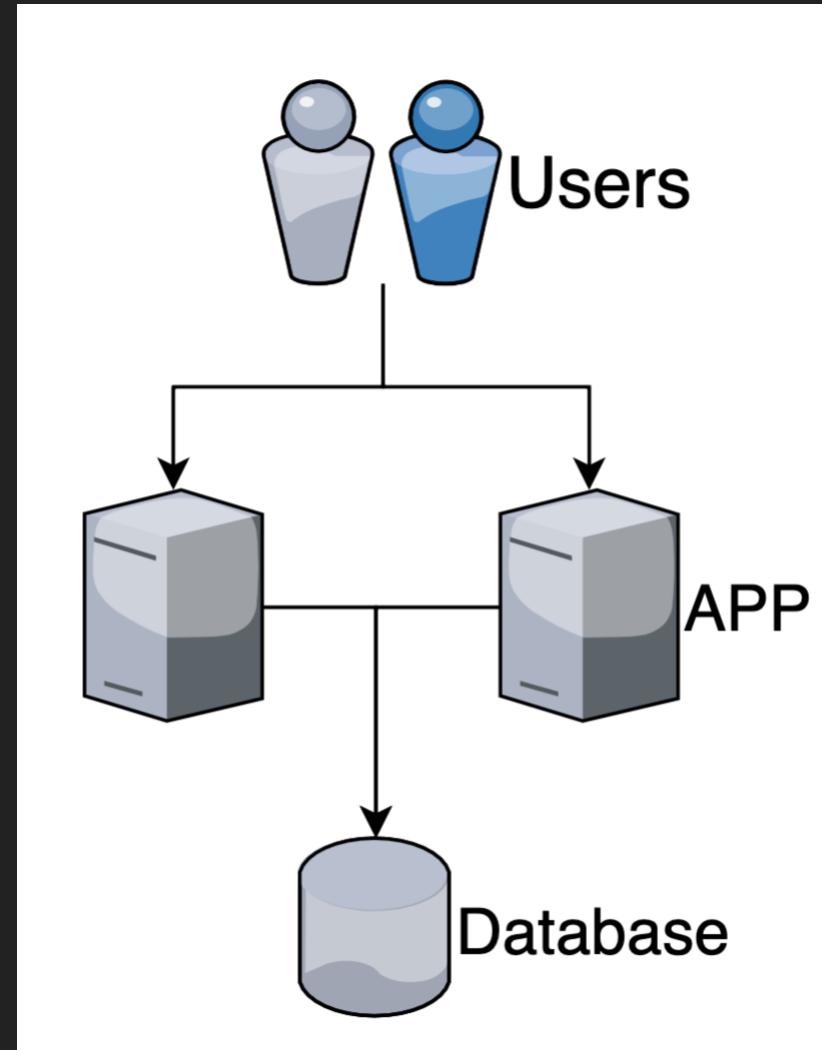


HAProxy

На стороне клиента

Клиенты знают адреса всех серверов, и сами выбирают на какойходить.

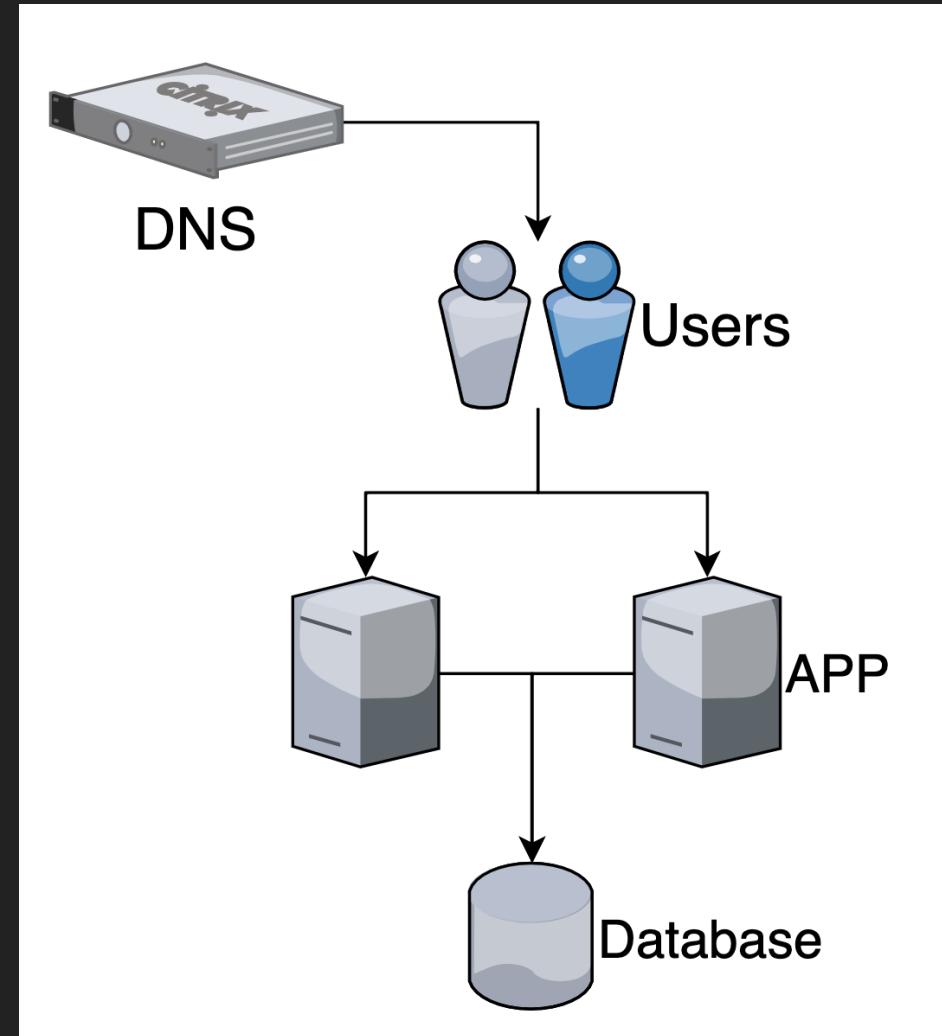
Нужно прописывать/менять адреса для клиентов по мере изменения набора хостов



DNS балансировка

Закрепляем за нашим доменом
несколько хостов (по их ip адресам).

Но DNS кэши медленно обновляются и
нельзя оперативно добавлять или
убирать хосты в случае обновлений/
аварий.



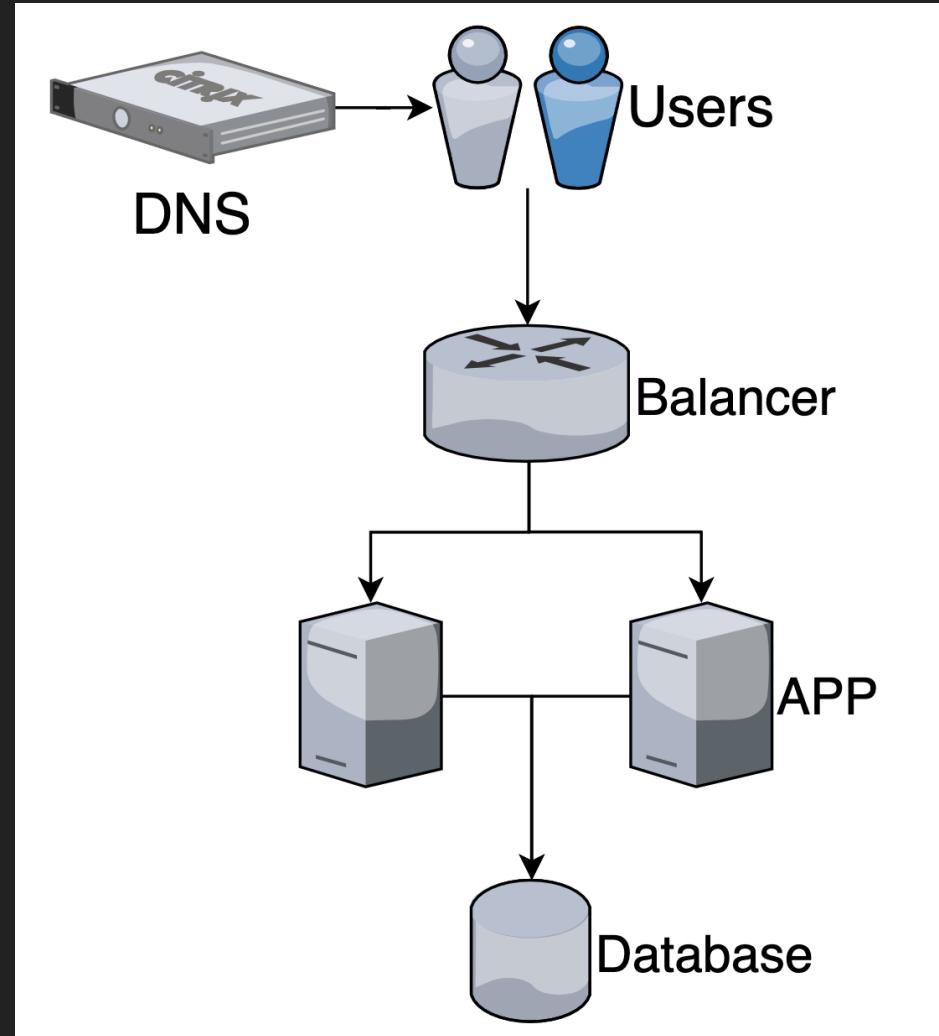
Revers-proxy (nginx, haproxy)

Выставляем наружу промежуточный сервер, который уже проксирует запросы в наши приложения.

Оптимальный вариант.

Может быть несколько уровней промежуточных балансеров (haproxy).

В DNS прописываем адрес нашего балансира.



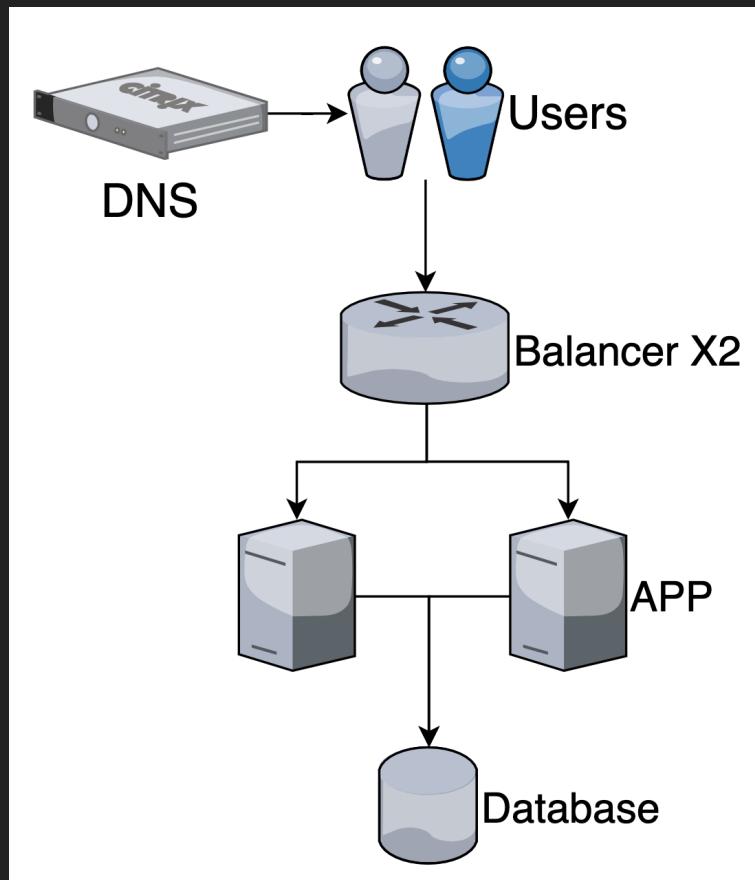
Алгоритмы балансировки

- Round robin - в порядке очереди
- По кол-ву активных запросов
- По весам (если хосты разные по мощности)

HTTPS, throttling и многое другое
МОЖНО настроить на стороне
reverse-proxy

Балансер тоже является точкой отказа и может быть продублирован

High availability +

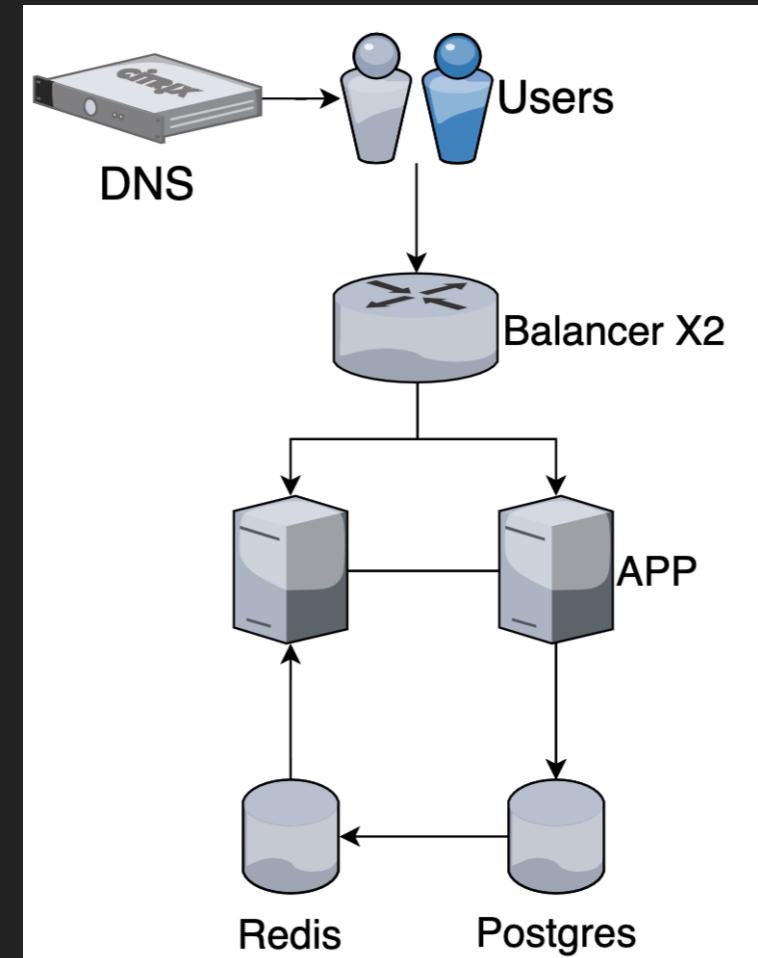


Запросы начинают
выполняться
медленно?

Добавляем кэширование

Performance +

High availability + (можем читать из кэша,
пока сервис недоступен)



Кэшировать можно на разных уровнях

- В памяти приложения
- В отдельном хранилище (redis)
- На балансере

Вспоминаем про
денормализацию

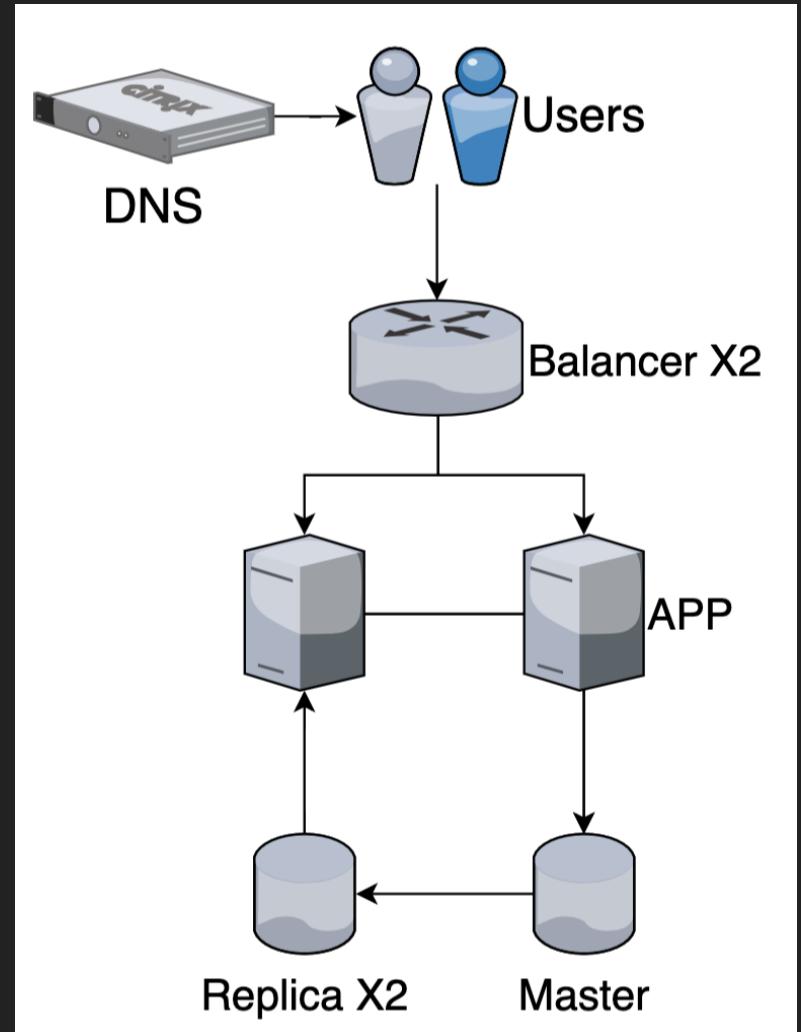
Слишком много
запросов на чтение в
бд?

Репликация

Репликация — это процесс, под которым понимается копирование данных из одного источника на другой (или на множество других)

Performance + (можем читать с реплик)

High availability + (failover)



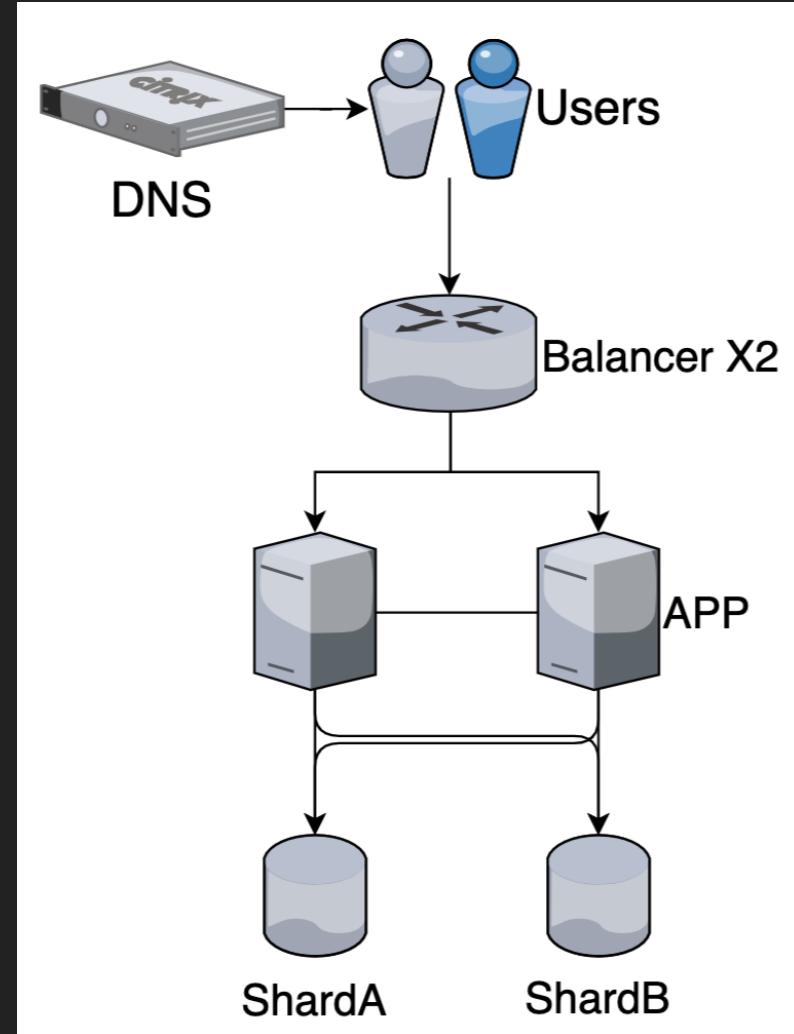
- + Увеличиваем возможность обрабатывать запросы на чтение
- + Отказоустойчивость (можно переключить запись на реплику, если мастер упал)
 - (async replication) **Eventual consistency** - изменения доходят до реплик с задержкой (для критично важных запросов читаем из мастера)
 - (sync replication) уменьшится скорость выполнения запросов на запись

Долго выполняются
запросы или данные не
помещаются на диск?

Шардирование

Принцип проектирования базы данных,
при котором логически независимые
строки таблицы базы данных хранятся
раздельно

Performance + (данных меньше на
конкретном шарде)



Данные можно делить по условию или типу

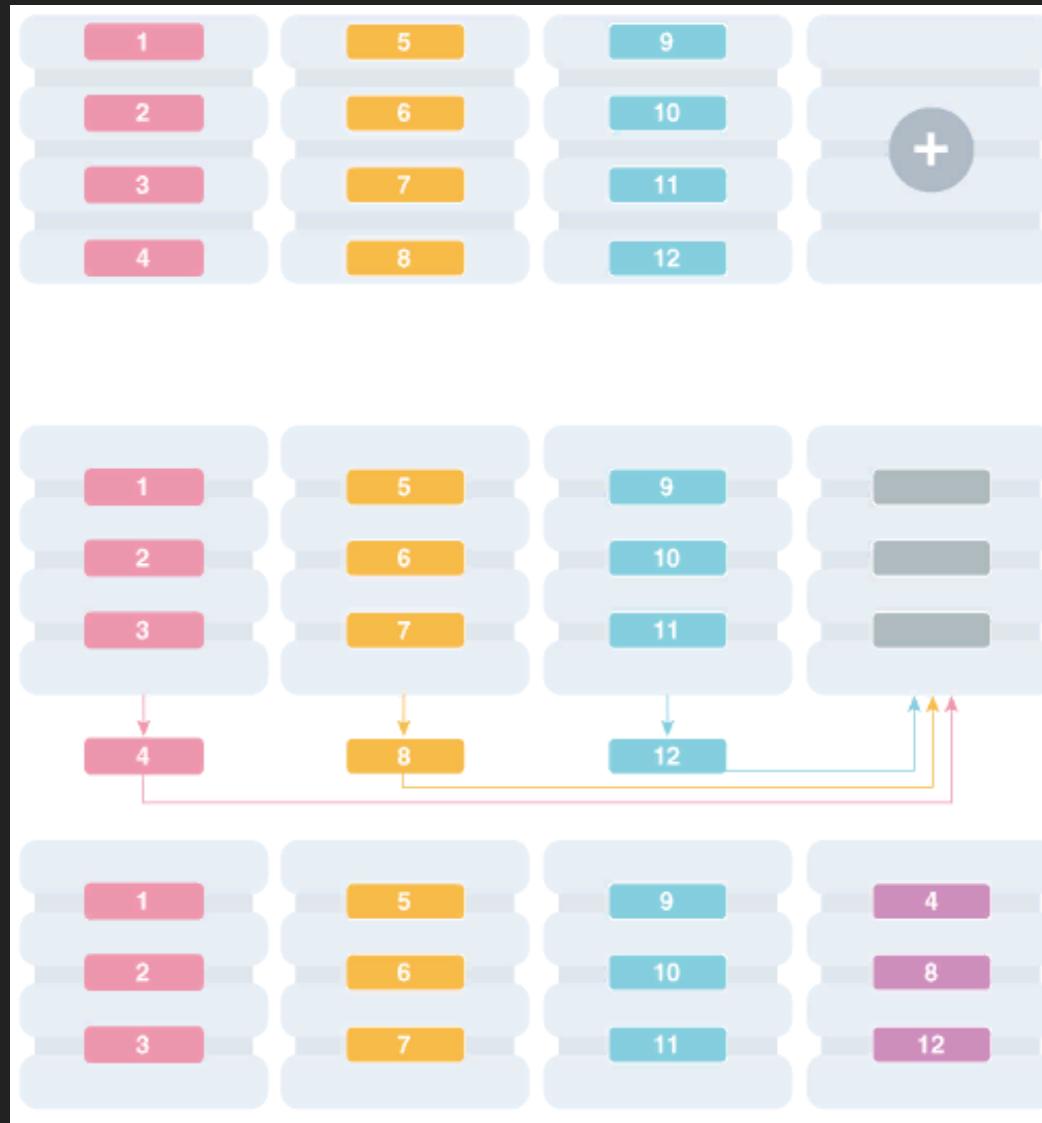
Например данные разных клиентов в разных базах, логи за разные периоды времени и тд - это разделение по условию. Под шардированием подразумевают такой способ.

Хранить сообщения чата в одной базе, аватарки пользователей в другой - это разделение по типу (тиปично для микросервисов)

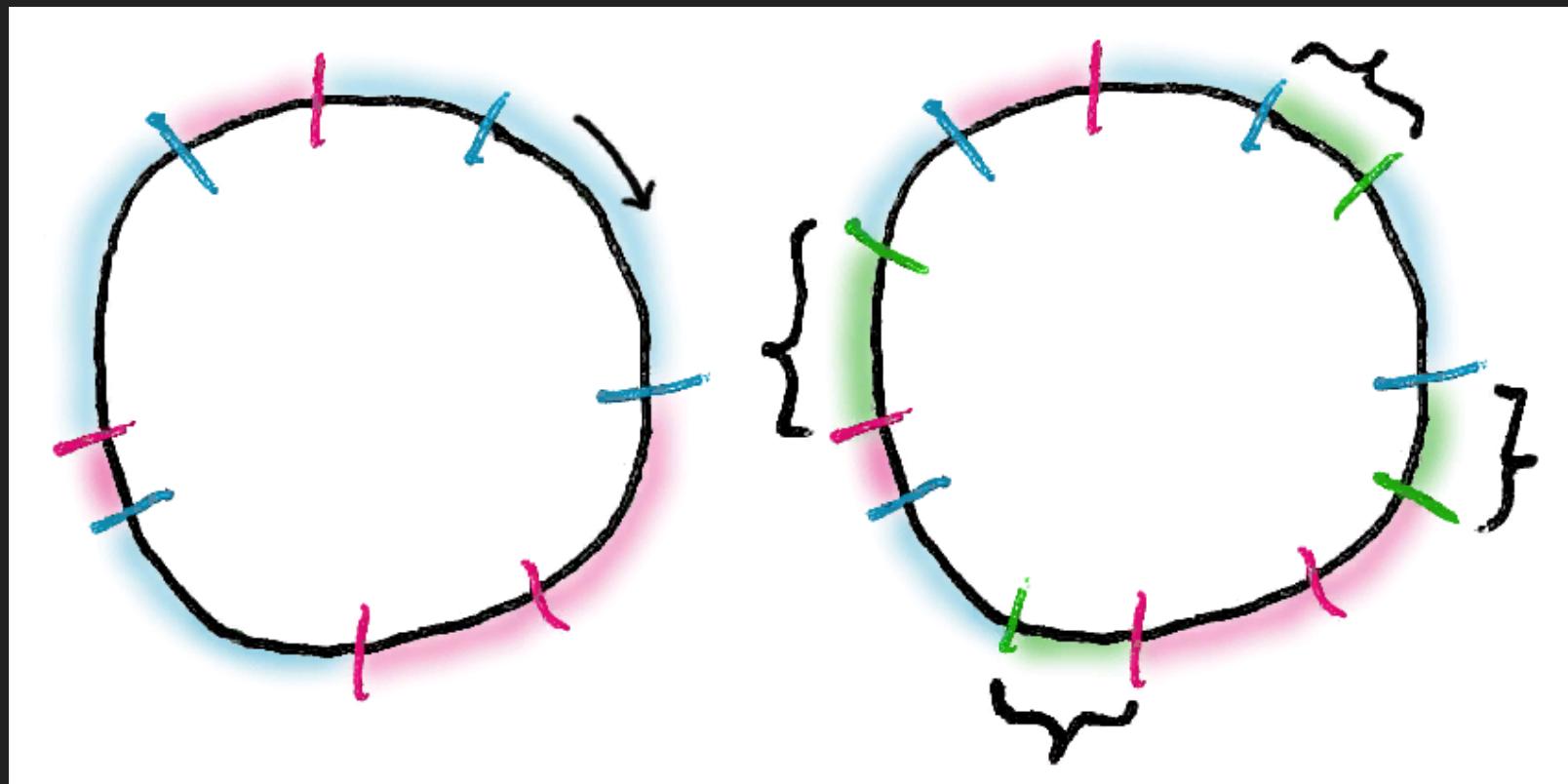
Консистентность не гарантируется,
например уникальный индекс по
данным на разных шардах придется
строить в отдельной базе данных.

Индексы для поиска по всем
данным также можно выносить в
отдельные базы (например elastic
для поиска по тексту)

Решардинг



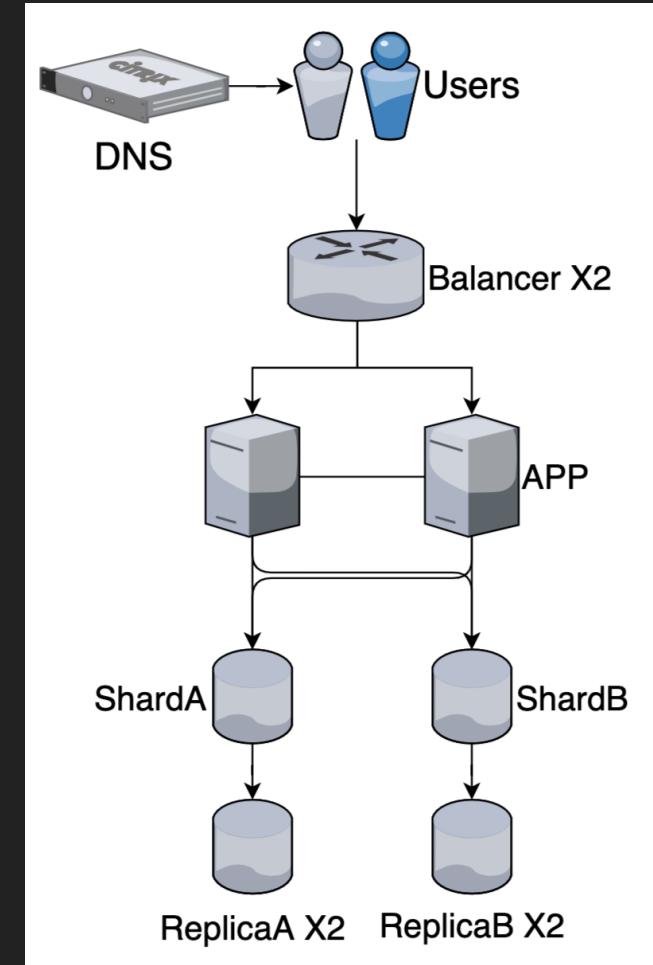
Hashring



Шардирование + Репликация

Прекрасно живут вместе!

High availability + (каждый шард
продублирован)

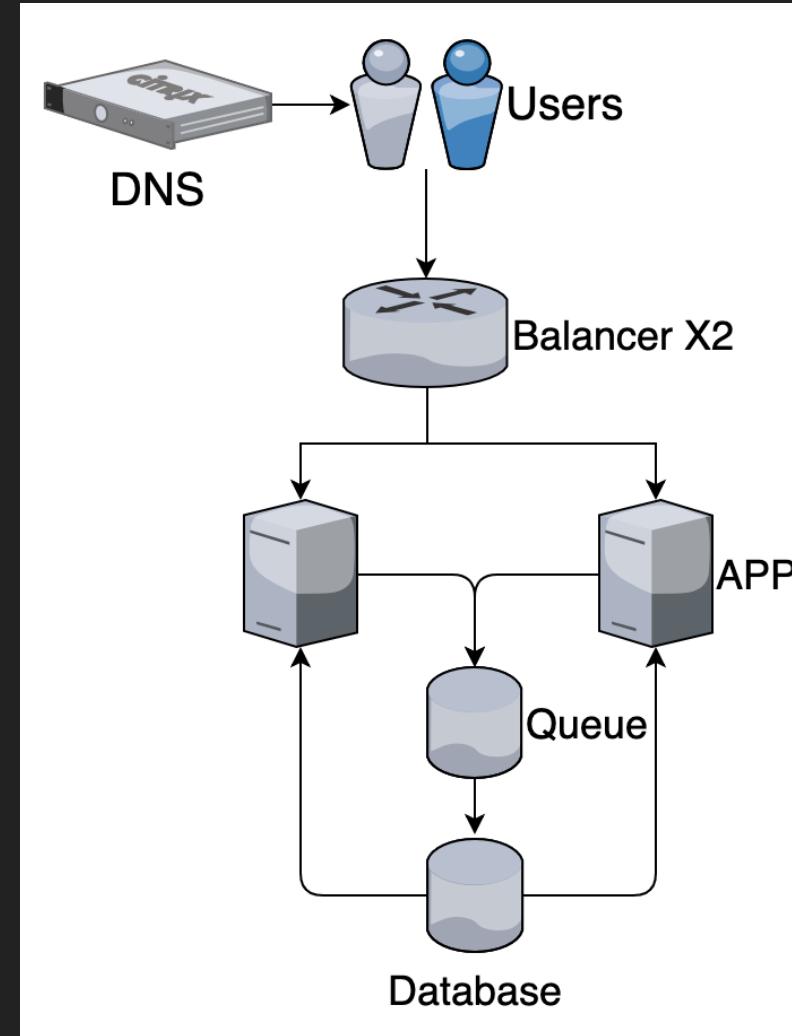


Слишком много
запросов на запись в
базу?

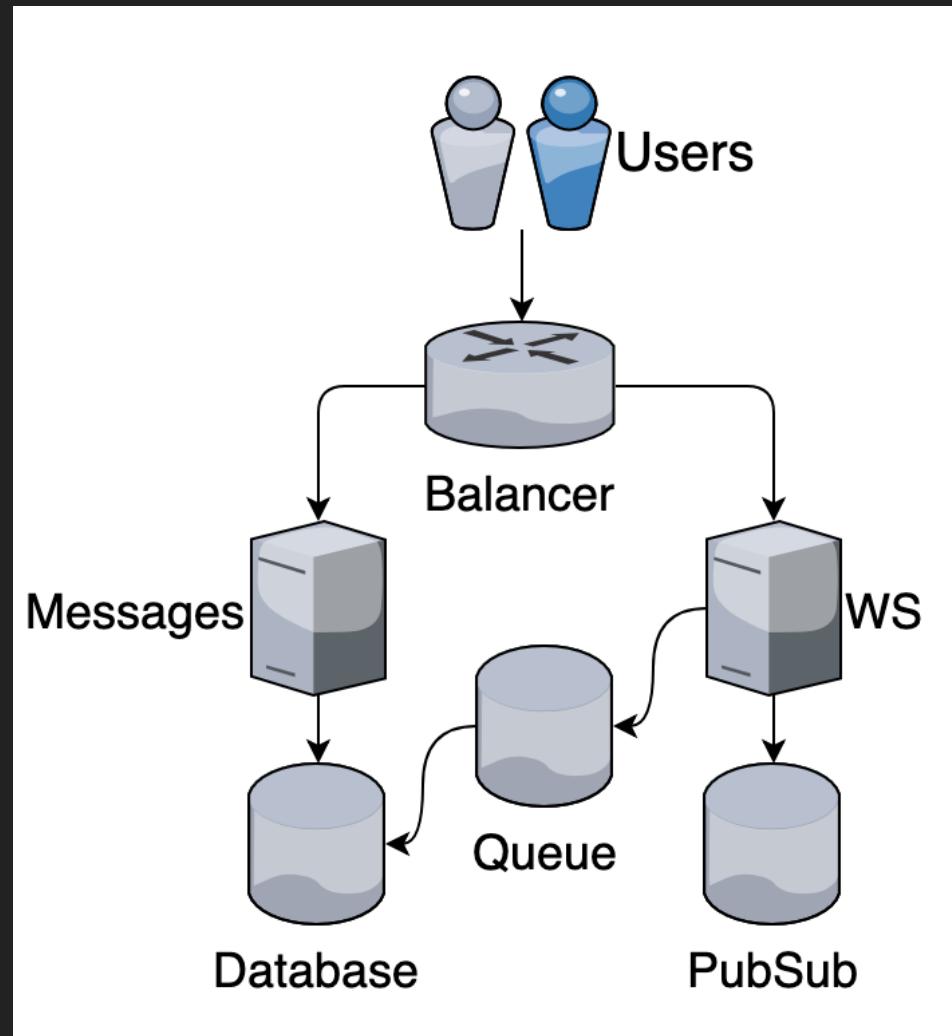
Пишем асинхронно через очередь (celery, rq, etc.)

Складываем запросы в промежуточную очередь и обрабатываем отдельными процессами с предсказуемой нагрузкой на базу

High availability + (размазываем нагрузку)



Чат



Спасибо, что были с нами все это
время! =)



Вопросы?