

# Масштабирование stateful и stateless сервисов. Паттерны кэширования.

Архитектор ПО



Меня хорошо  
слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте  + если все хорошо

## Карта вебинара

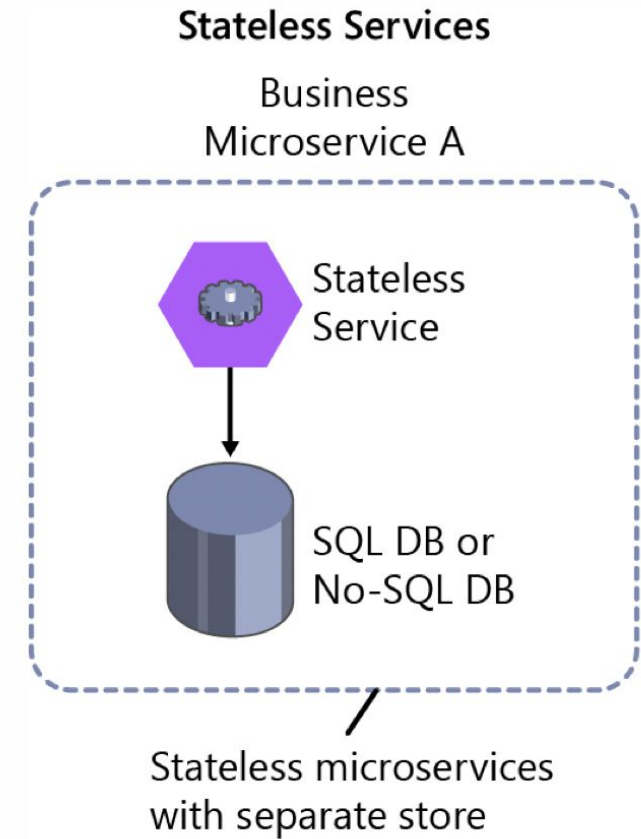
- Stateless vs stateful масштабирование
- Идемпотентность API
- Паттерны кэширования
- Кэширование данных в разных сервисах

# Stateless сервисы

**stateless** сервисы:

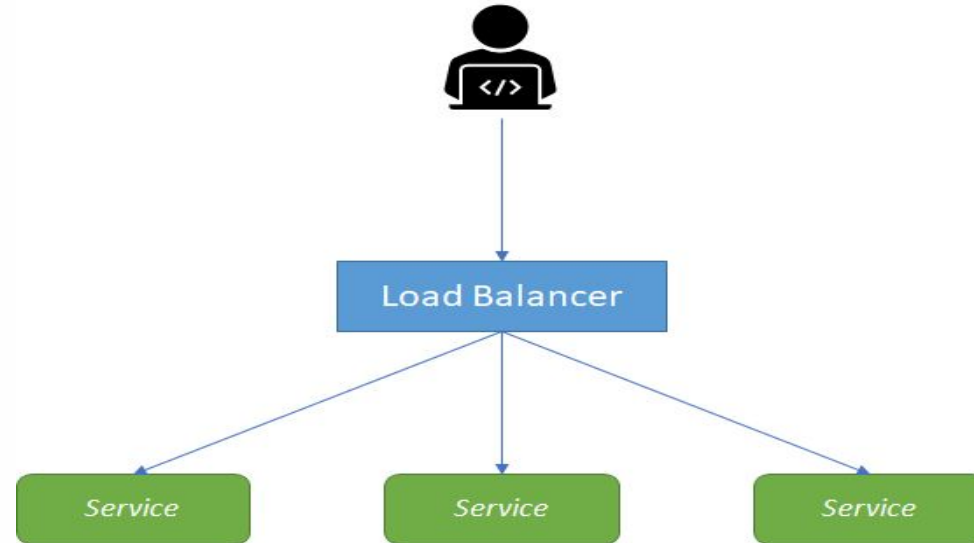
- Не хранят состояние между запросами клиента внутри себя
- Могут хранить состояние во внешнем сервисе

Примеры: проху, gateway, обычные CRUD-сервисы.

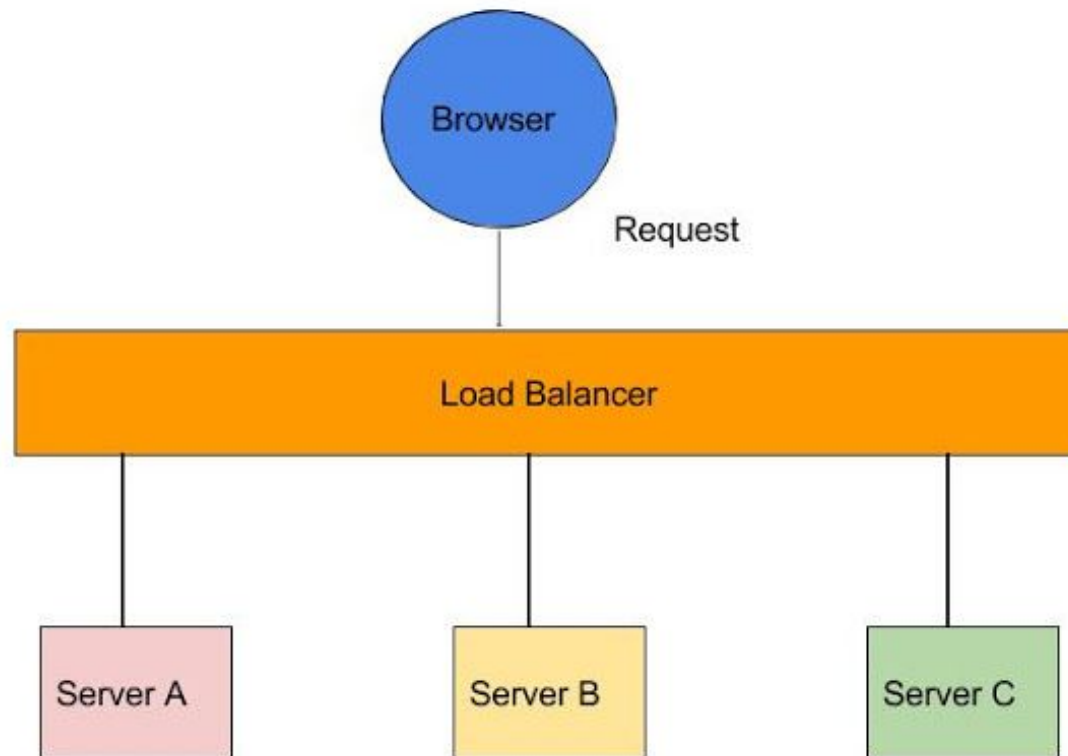


# Stateless сервисы

- **stateless** сервисы масштабируются горизонтально
- Можно направлять запрос на любую ноду

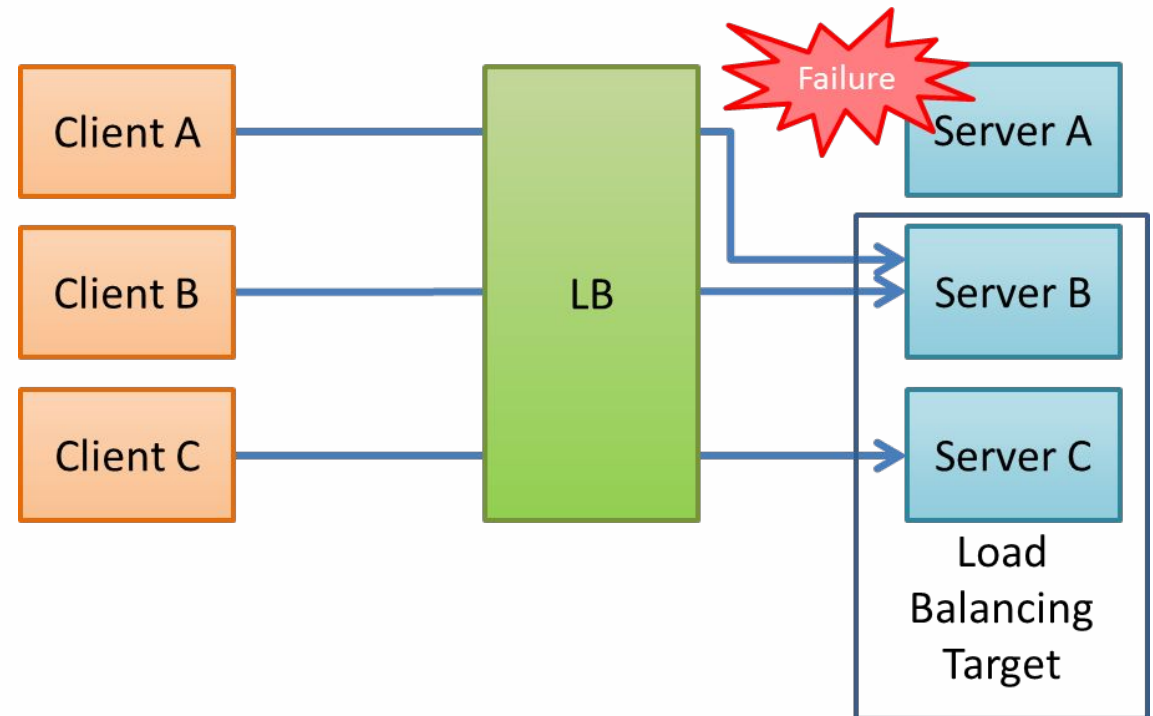


# Масштабирование stateless сервисов



# Отказоустойчивость при балансировании

- Отключить упавший сервис/сервер из балансирования
- Подготовить код приложения и архитектуру к частым и/или возможным падениям
- В микросервисной архитектуре балансинг делается на стороне оркестратора приложений (Kubernetes)

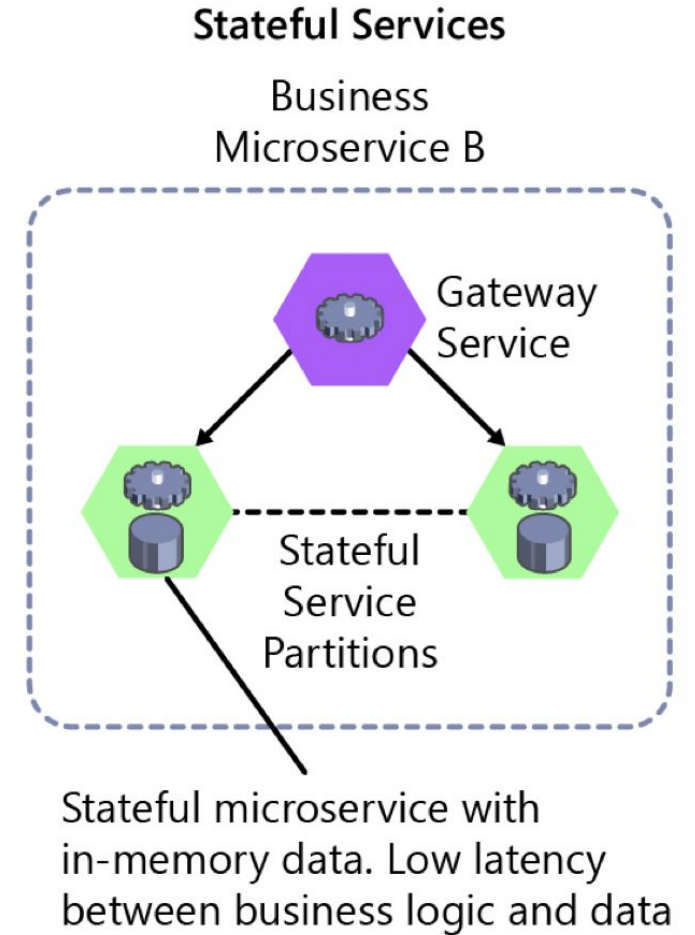


# Stateful сервисы

## stateful сервисы

- Хранят состояние
- Масштабируются за счет репликаций, шардинга и т.д.

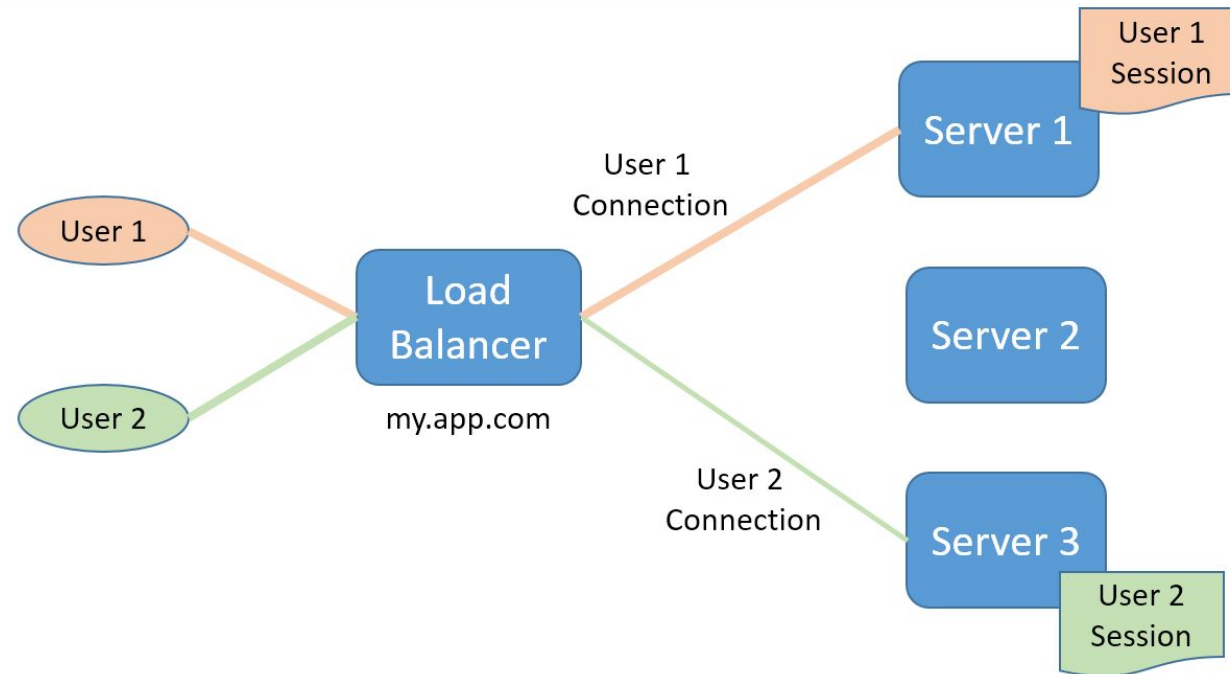
Примеры: БД (oracle, postgres, mongodb)  
in-memory cache (tarantool, ignite)  
web-socket приложения (чаты)





# Stateful сервисы

- **stateful** сервисы масштабируются за счет шардинга и репликации
- Запросы должны роутиться на конкретные ноды Sticky sessions, sharding и т.д



## Кейс

Есть приложение «Интернет-магазин». Когда пользователь нажимает кнопку «Оформить заказ», то происходит запрос POST /api/v1/orders/

```
{  
  "products": [{"id": 42, "price": "2500"}],  
  "shipping_to": "Большая Филевская 16к1, кв. 14"  
}
```

При этом деньги снимаются со счета в Личном кабинете, и происходит резервирование товара на складе.

Кнопка после нажатия остается активной. И иногда пользователи два раза нажимают на кнопку и происходит дублирование заказа со снятием двойной суммы.

Что бы вы предложили в качестве решения проблемы?

## Кейс

Пока не придет ответ сервера, не делаем кнопку «активной».

Что делать, если интернет отвалился в момент нажатия на кнопку «Оформить заказ»? Клиент получил ошибку таймаута, и считает, что запрос не прошел, а сервер его выполнил.

## Кейс

Добавляем ключ Request-Id в запрос

```
POST /api/v1/orders/
```

```
X-Request-Id: de7efba4-267c-11ea-978f-2e728ce88125
```

```
{  
  "products": [{"id": 42, "price": "2500"}],  
  "shipping_to": "Большая Филевская 16к1, кв. 14"  
}
```

Если запрос с таким ключом уже пришел, то мы заказ не создаем.

## Кейс

Пользователь нажал кнопку «Оформить заказ». Но ответ от сервиса был очень долгим. Клиент не стал дожидаться ответа от приложения и его полностью закрыл и выгрузил из памяти.

Когда он зашел в приложение, запрос еще не отработал и в списке заказов старого заказа не было. Клиент сформировал новый operation-id и отправил еще один запрос. В результате создалось 2 заказа.

## Решение

Делаем версионирование коллекции `/api/v1/orders/`.  
Сервер присылает заголовок `Etag` с версией коллекции `orders`.  
Клиент при изменении коллекции заголовок `If-Match` с версией, которую он знает.

```
/api/v1/orders/  
Etag: 42
```

Сервер проверяет: если `If-Match` совпадает с версией на сервере, то запрос проходит. Если нет, то отвечает ошибкой.

```
POST /api/v1/orders/  
If-Match: 42
```

429 Conflict

## Решение

Иногда Request-Id передается query параметром  
Иногда используется не версия, а hash от содержания  
коллекции – fingerprint

<https://cloud.google.com/compute/docs/reference/rest/v1/instances/setTags>

# Идемпотентность API

**Идемпотентность API** – можно послать несколько раз один и тот же запрос (сообщение), и состояние на сервере не изменится.



## Идемпотентность удаления

Можно несколько раз вызывать удаление ресурса и результат будет таким же.

DELETE /api/v1/orders/{id}/

- Если заказ с таким id уже был удален, то 200 OK
- Если заказа с таким id не было, то 400 BAD REQUEST

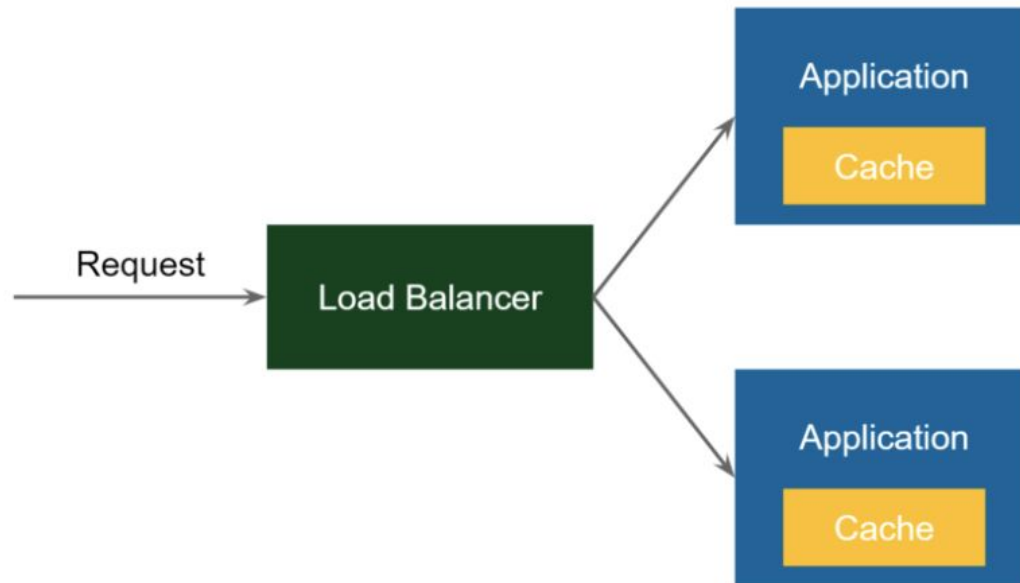
## Идемпотентность и внешние сервисы

Есть сервис notification. Он отправляет смс-ки. Сервис читает сообщения из очереди и отправляет запрос к внешнему сервису, после чего помечает, что сообщение отправлено.

Иногда сервис умирает или внешний сервис не отвечает. И тогда сервис снова берет тоже самое сообщение из очереди. Что потенциально приводит к нескольким доставкам сообщений клиенту

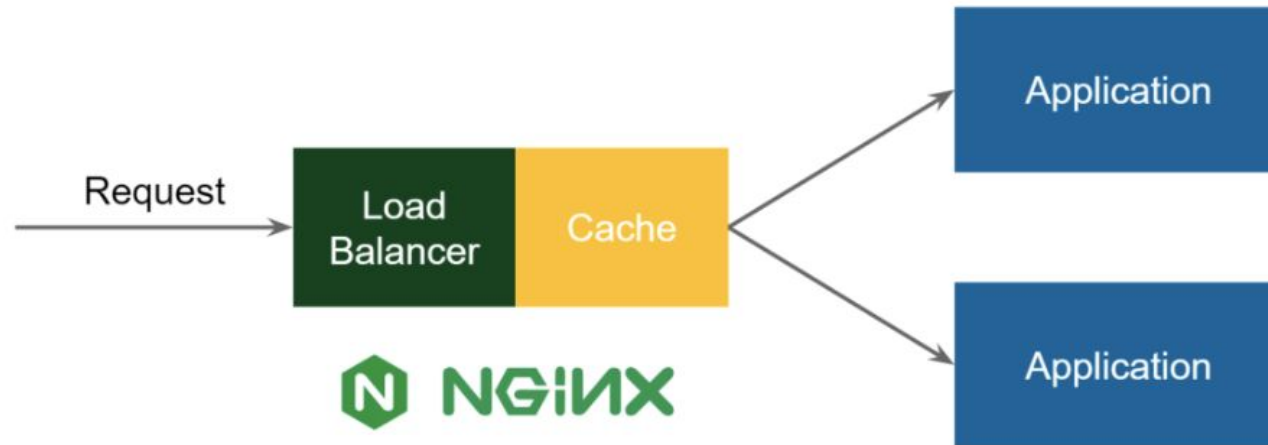
# Кэш в приложении

- кэш хранится на уровне приложения
- все кэши изолированы друг от друга
- приходится кэшировать одно и то же много раз на каждом из инстансов



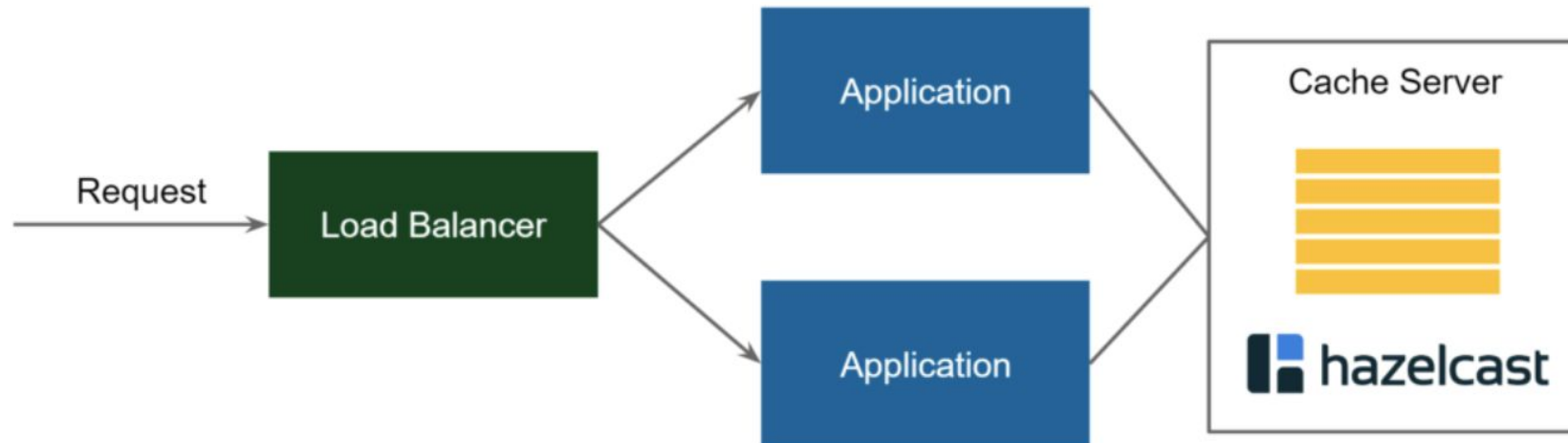
# Reverse proxy

- Кэш хранится на балансере
- Инвалидация кэша на стороне сервера, а не приложения



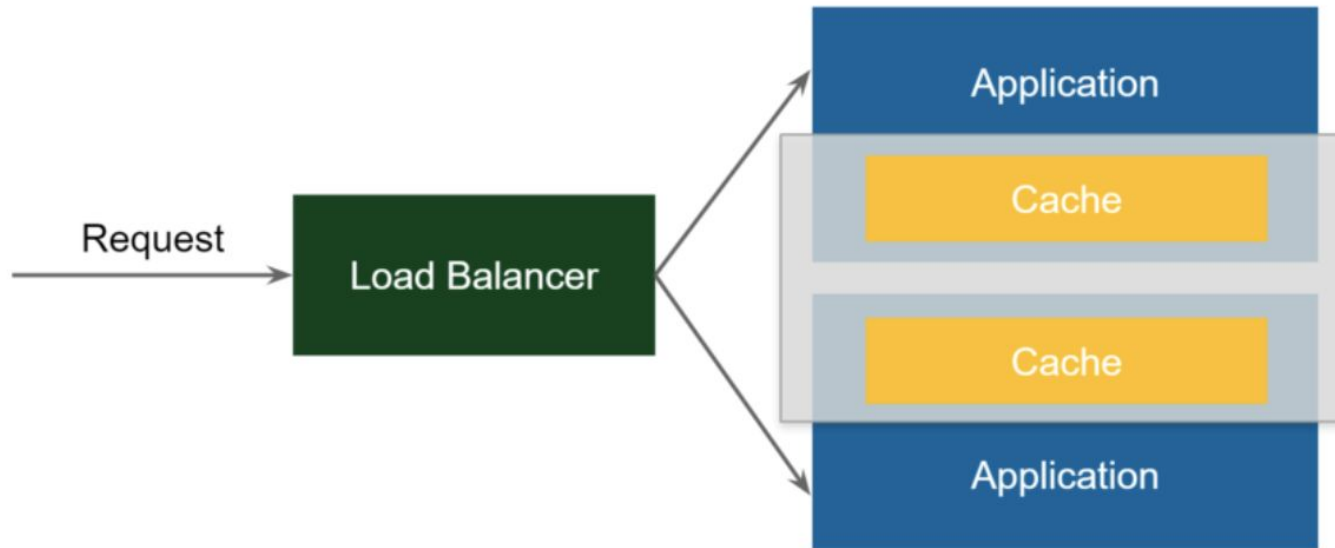
## Отдельный кэш

- Кэш хранится в отдельном сервисе
- Сервис кэша можно отдельно скейлить, если надо
- стек приложения не важен



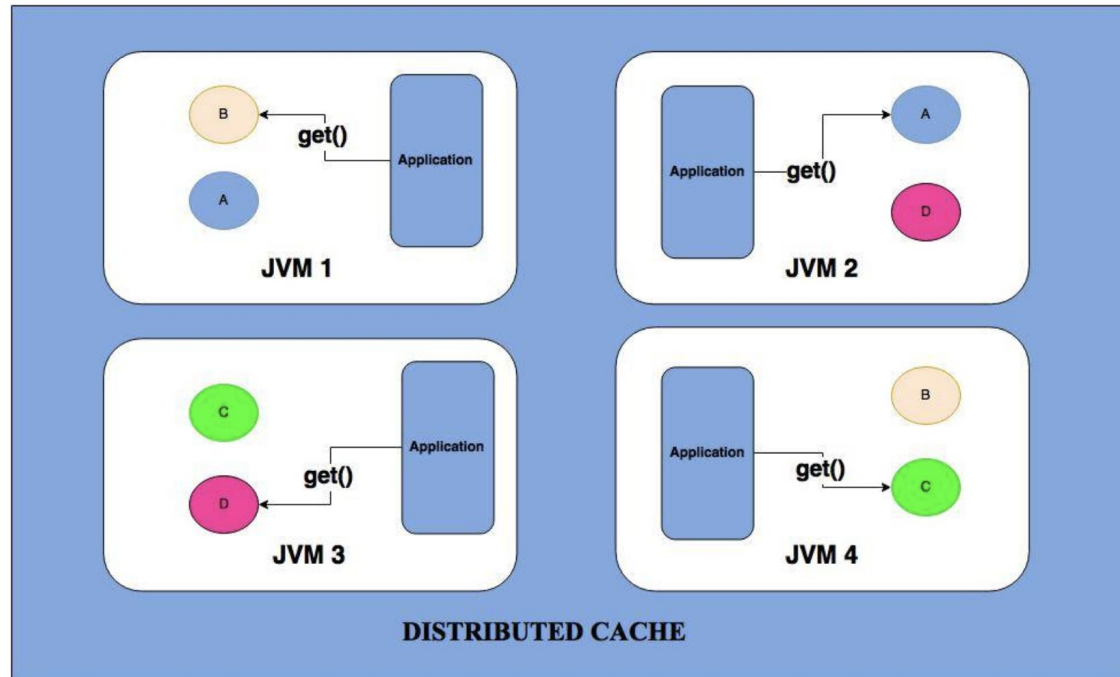
# Распределенный кэш

- кэш хранится на уровне приложения, но шарится на несколько приложений (реплицируется или распределяется)
- Зависит от реализации в фреймворке и языке программирования



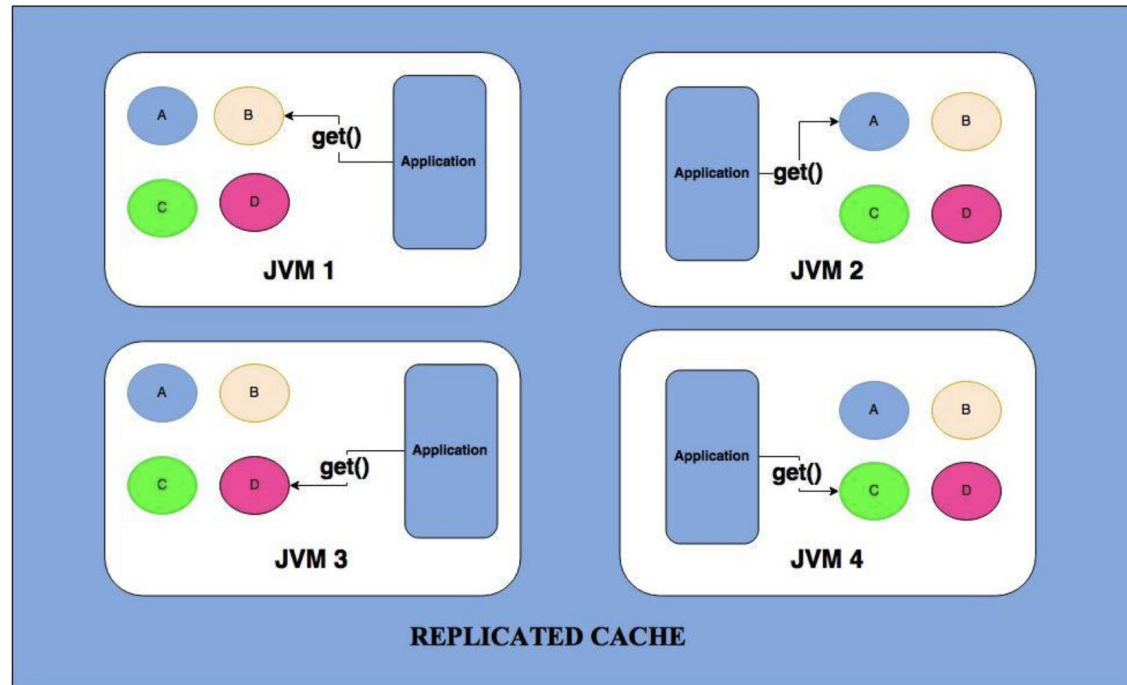
# Distributed кэш

- Распределенный кэш – если ключ не в текущем шарде, идем в другой инстанс



# Replicated кэш

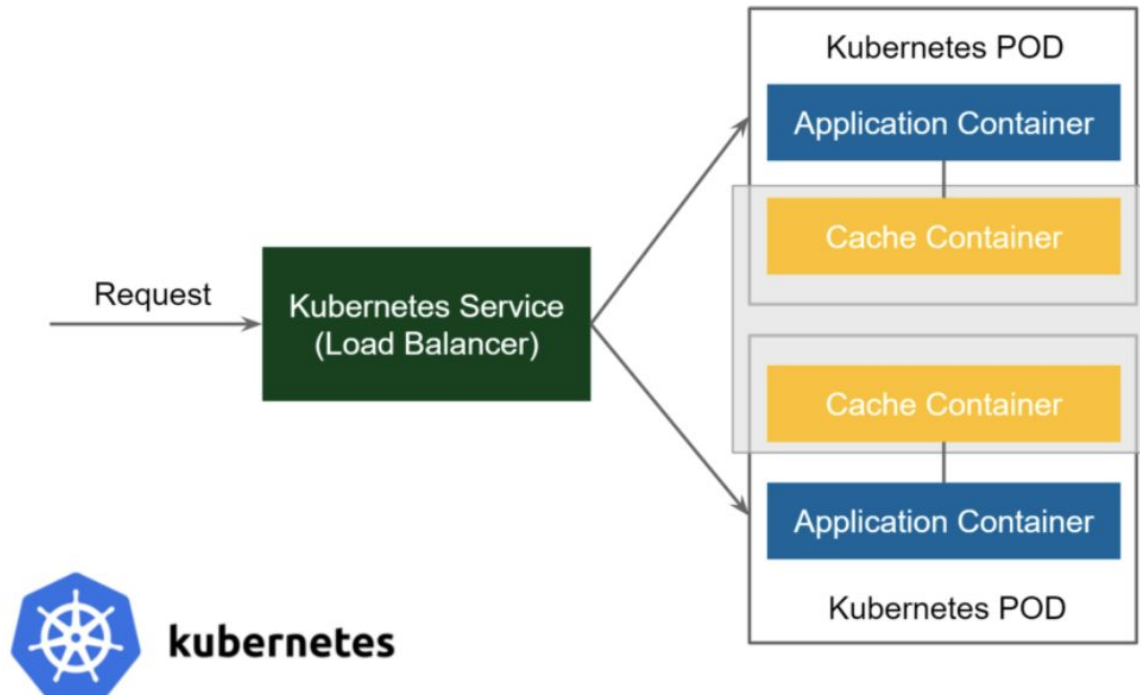
- Репликация кэша – кэш реплицируется на все инстансы.
- Быстрее распределенного, но требует больше ресурсов на инстанс





# Sidcar кэш

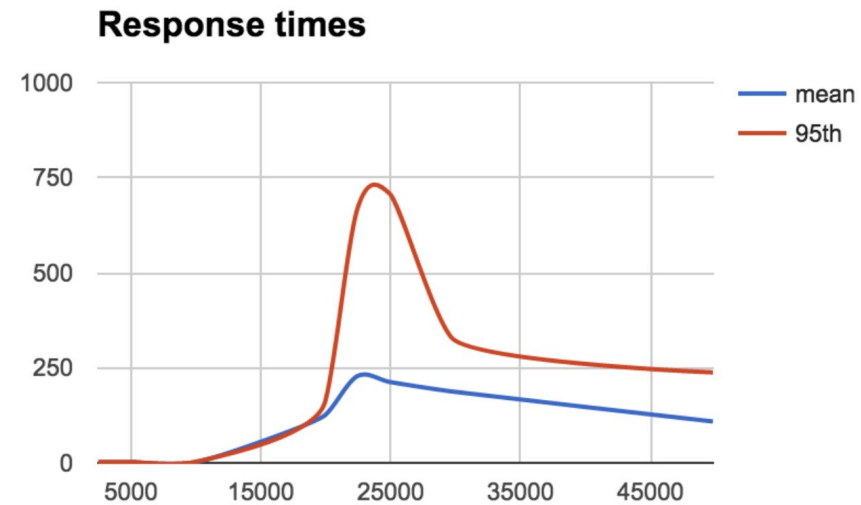
- кэш хранится на приложения кэша, но шарится на несколько приложений инстансов
- Может работать в режиме репликации, так и в распределенном режиме
- Ниже latency
- Не зависит от стека приложения



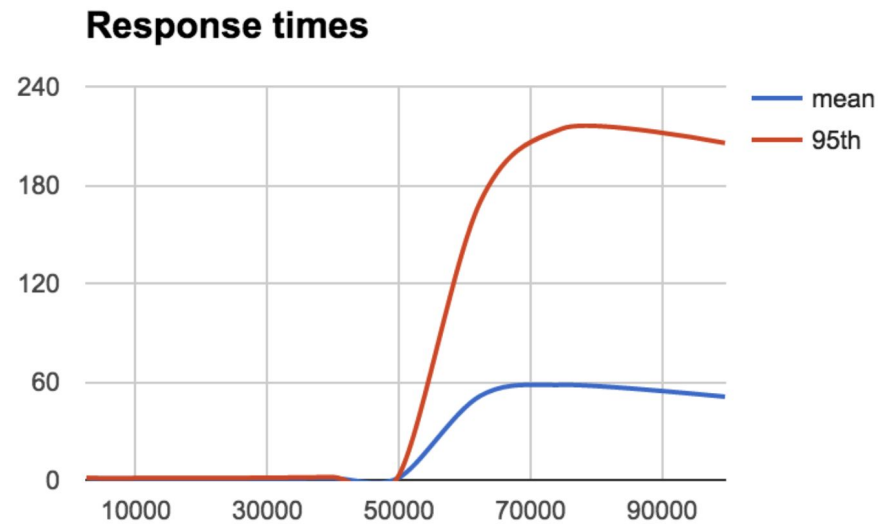
# Sidocar кэш

<https://www.unacast.com/post/high-performance-read-api-on-kubernetes-using-redis>

Выделенный кластер  
redis-a



Side-car кластер redis-  
a



# Опрос

<https://otus.ru/polls/6406/>

**Спасибо  
за внимание!**

