УДК 004.254

Маг. М.Н. Карпович; доц. Н.В. Пацей
(БГТУ, г. Минск)

КЕШИРОВАНИЕ ДАННЫХ МЕЖДУ МИКРОСЕРВИСАМИ В БЕССЕРВЕРНОЙ АРХИТЕКТУРЕ

При реализации микросервисной архитектуры организациям приходится перестраивать архитектуру традиционных монолитных приложений. Это помогает повысить гибкость и масштабируемость сервисов и быстрее выводить на рынок новые продукты [1].

Бессерверные вычисления – это метод предоставления серверных услуг на основе фактического использования сервисов. Бессерверный провайдер позволяет пользователям писать и развёртывать код, не беспокоясь о базовой инфраструктуре. Компания, которая получает бэкенд-услуги от бессерверного поставщика, платит по за используемые ресурсы и не должна резервировать и оплачивать фиксированную пропускную способность или количество серверов, поскольку услуга автоматически масштабируется, для предоставления клиенту бессерверных вычислений используются физические серверы, но разработчикам нет необходимости думать об их конфигурации, производительности, ядрах, памяти и прочем.

На сегодняшний день архитектура, основанная на микросервисах, является наиболее распространённой при разработке больших и средних проектов. Основные преимущества данной архитектуры: масштабирование, возможность выбора технологий, небольшие команды разработки, устойчивость к сбоям и независимая модель данных [2]. Кеш снижает задержку и ускоряет взаимодействие между сервисами в микросервисных архитектурах. Кеш – это уровень хранения данных с высокой скоростью доступа, на котором находится подмножество всех ваших данных. Из кеша они будут доставлены быстрее, чем при обращении к основному хранилищу данных.

Кеширование может быть реализовано несколькими способами. Мы рассмотрим два таких сценария. В обоих случаях слой микросервисов создаётся с использованием бессерверных вычислений на платформе *AWS*. При этом требуются данные из нескольких источников, развёрнутых локально в облаке или на клиентском оборудовании. Вычислительный уровень построен с использованием функций *AWS Lambda*. Хотя сами *Lambda*-функции живут недолго, кешированные данные могут использоваться последующими экземплярами того же микросервиса без обращения к бэкенду.

Шаблон *Cache-Aside*, представленный на рис. 1, применяется

для ленивой загрузки часто используемых данных. Это означает, что объект кешируется, только когда потребитель его запрашивает, а соответствующий микросервис решает, стоит ли сохранять объект [3].

Такой способ полезен, когда уровень микросервисов выполняет несколько вызовов для извлечения и обработки данных в режиме реального времени. Объём этих вызовов можно существенно сократить, если кешировать на короткое время часто используемые данные.

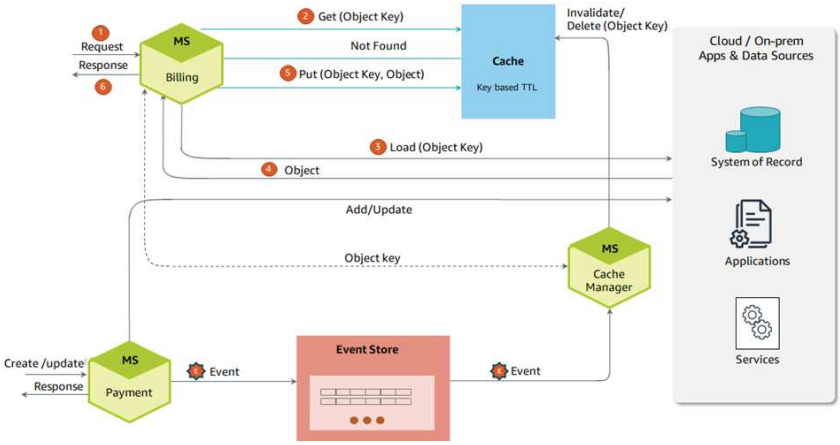

Рисунок 1 – Кеширование данных с использование шаблона Cache-Aside

На рис. 2 показаны сервисы *AWS* для реализации данного шаблона. Уровень микросервисов (*Billing*, *Payments* и *Profile*) создаётся с помощью функции *AWS Lambda*.

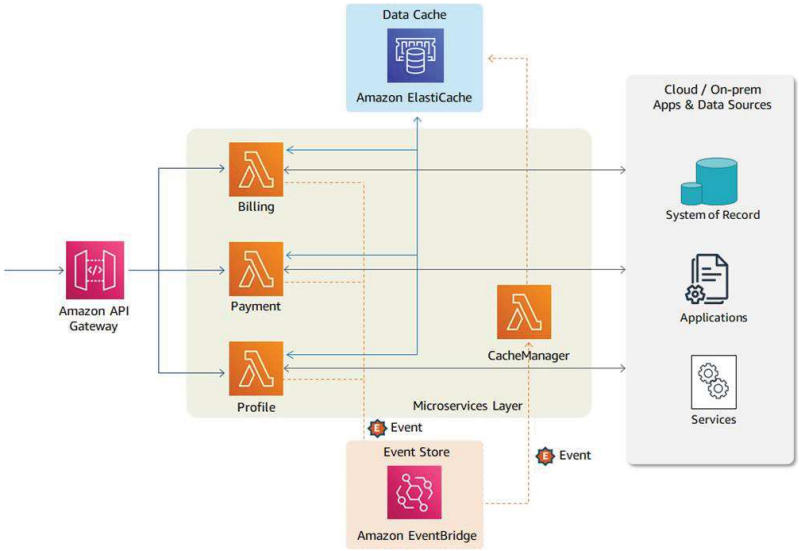

Рисунок 2 – Сервисы AWS для реализации шаблона Cache-Aside

Шлюз *Amazon API Gateway* представляет *Lambda*-функции в виде *API*-операций и позволяет внутренним или внешним потребителям обращаться к ним.

Все три микросервиса подключены к кешу данных и могут сохранять в него и извлекать оттуда объекты. Кеш организован в опера-

тивной памяти с помощью технологии *Amazon ElastiCache*. Объекты данных хранятся в кеше недолго. Каждому объекту присваивается соответствующее значение времени жизни (*TTL*). Когда оно истекает, объект удаляют. Пользовательские события, например, *payment processed*, публикуются в *Amazon EventBridge*, а затем обрабатываются.

Далее рассмотрим упреждающее кеширование больших объёмов данных. В крупных проектах модернизации и миграции не все источники данных развёртываются на сравнительно небольшое время. Некоторые унаследованные системы, например, работающие на мейнфреймах, требуют более длительного периода вывода из эксплуатации. Многие устаревшие бэкенд-системы обрабатывают данные с помощью периодических пакетных заданий. В таких сценариях фронтендные приложения могут использовать кешированные данные от нескольких минут до нескольких часов, в зависимости от характера данных и способа их использования. При этом серверные системы не справляются с большим объёмом вызовов на уровне интерфейсного приложения [4].

В таких сценариях необходимые данные или объекты можно идентифицировать заранее и загрузить непосредственно в кеш, пример предоставлен на рис. 3. Автоматизированный процесс загружает данные/объекты в кеш во время начальной загрузки. Изменения на уровне источников данных (в БД на мейнфрейме или другой системе хранения данных) фиксируются и применяются на уровне кеша с помощью автоматизированного конвейера *CDC* (*Change Data Capture*).

В отличие от первого сценария, микросервисы не выполняют вызовы в реальном времени для подгрузки данных в кеш, а используют те данные, которые уже были для них закешированы.

Тем не менее на уровне микросервисов могут сформироваться события, если данные в кеше устарели или определённые объекты были изменены другим сервисом, например, сервисом *Payment* при совершении платежа. События хранятся в менеджере событий *Event Manager*. При получении события *CacheManager* инициирует бэкенд-процесс для обновления устаревших данных по требованию. Все изменения данных поступают непосредственно в систему записи.

Данный сценарий также полезен, когда фронтенд-приложения должны кешировать данные на длительное время, например, в корзине покупок.

Как показано на рис. 4, объекты данных хранятся в базе данных *Amazon DynamoDB*, которая обеспечивает низкую задержку при доступе к данным в любых масштабах.

Извлекает данные акселератор *DynamoDB Accelerator (DAX)* – полностью управляемый, высокодоступный кеш, который работает в оперативной памяти. Он десятикратно повышает скорость извлечения данных – даже при миллионах запросов в секунду. *API Gateway, Lambda* и *EventBridge* предоставляют те же функции, что и в первом сценарии [5].

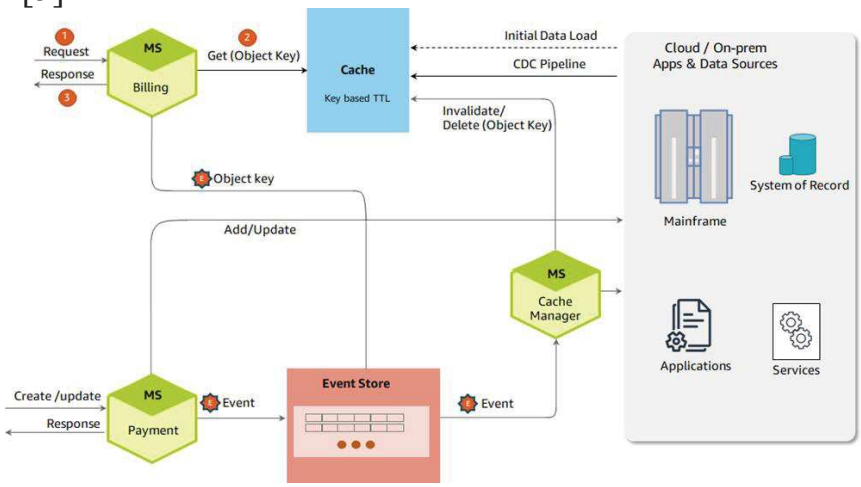

Рисунок 3 – Упреждающее кеширование массивных объёмов данных

Можно сделать выводы, что архитектура микросервисов позволяет построить несколько уровней кеширования в зависимости от сценария.

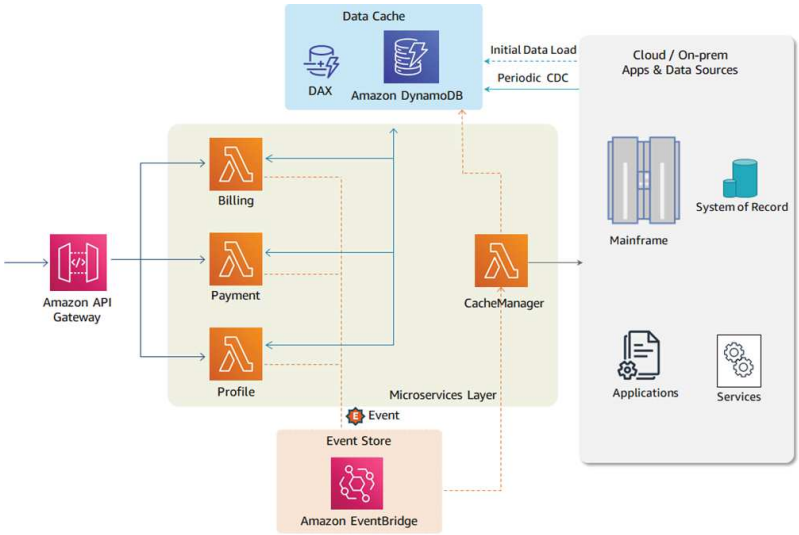

Рисунок 4 – Сервисы AWS для реализации упреждающего кэширования

В этой работе мы рассмотрели, каким образом кеширование данных на вычислительном уровне способно уменьшить задержку при получении данных из разных источников. В сценарии 1 количество обращений к серверной системе в режиме реального времени сокращается благодаря тому, что наиболее часто используемые данные сохраняются в кеше. Сценарий 2 поможет хранить большие объёмы

данных в кеше продолжительное время, когда вызовы внутренней системы в реальном времени невозможны.

ЛИТЕРАТУРА

1. Карпович М. Н. Проектирование микросервисных архитектур информационных систем // Информационные технологии: материалы 86-й науч.-техн. конференции профессорско-преподавательского состава, научных сотрудников и аспирантов (с международным участием), Минск, 31 января – 12 февраля 2022 года [Электронный ресурс] / отв. за издание И.В. Войтов; УО БГТУ. – Минск: БГТУ, 2022. – С. 82-85.

2. Ньюман, С. От монолита к микросервисам: эволюционные паттерны для преобразования вашего монолита / С. Ньюман – США: O'Reilly Media, 2019. – 94 с.

3. Ричардсон, С. Шаблоны микросервисов: с примерами на Java / С. Ричардсон – США: Manning Publications, 2018. – С. 65-110.

4. Кристудас, Б. Практические архитектурные шаблоны микросервисов / Б. Кристудас - США: Апрель, 2019 – С. 77-104.

5. Черноусов А. Кэширование данных в микросервисной архитектуре [Электронный ресурс] / Habrahabr. – URL: https://habr.com/ ru/post/651829 (дата обращения: 10.02.2023).

УДК 004.85

Ст. преп. И.Г. Сухорукова
(БГТУ, г. Минск)

ПРИМЕНЕНИЕ МЕТОДОВ ВЛОЖЕНИЯ В ЗАДАЧАХ МАШИННОГО ОБУЧЕНИЯ

Для работы с методами машинного обучения изображения, текст и прочие объекты обычно представляется в виде вектора или матрицы с действительными числами. Этот вектор (или матрица) в контексте машинного обучения называется вложением (embedding) входных данных.

Первоначально методы вложения получили распространение при обработке естественного языка. Наибольший интерес представляют методы вложения слов, которые позволяют учитывать семантическое значение слов. К примеру модель вложения слов *Word2Vec* обучается методами глубокого обучения на огромном количестве текста с предсказанием, какое слово возникает в схожих контекстах. После обучения *Word2Vec* генерирует вектор из 300 измерений для каждого слова в словаре, этот вектор и есть вложение слова. Имея вектор-