УДК 004.051

РАЗРАБОТКА БЕССЕРВЕРНЫХ ВЕБ-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ИНСТРУМЕНТОВ AMAZON WEB SERVICES

Ганьжа А.Ю., Карелова Р.А.

Нижнетагильский технологический институт (филиал)
ФГАОУ ВО «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина», Нижний Тагил, e-mail: riya2003@mail.ru

Данная статья посвящена обзору основных особенностей бессерверной архитектуры как способа реализации веб-приложений. Бессерверная архитектура подразумевает, что инфраструктура программного средства поддерживается сторонними провайдерами, а необходимая функциональность предоставляется в форме сервисов, отвечающих за процессы аутентификации, отправки уведомлений и интеграции со сторонними ресурсами. В статье рассматриваются ключевые особенности структуры бессерверных веб-приложений, а также реализация сервисных моделей BaaS (Backend-as-a-Service) и FaaS (Function-as-a-Service), раскрываются преимущества бессерверной архитектуры по сравнению с клиент-серверной реализацией веб-приложений. В представленных материалах описывается назначение облачных сервисов платформы Amazon Web Services, таких как API Gateway, Lambda и DynamoDB. В качестве примера совместного использования инструментов облачной платформы AWS демонстрируется процесс реализации гибкого API бессерверного веб-приложения для ведения блога. Пошагово описываются такие процессы, как создание таблиц базы данных DynamoDB, объявление HTTP API, а также реализация обработчиков POST и GET-запросов. Приводятся примеры кода в виде листингов. Результатом проведенной работы является систематизация теоретических сведений и демонстрация практических приемов для разработки бессерверных веб-приложений с использованием инструментов Amazon Web Services.

Ключевые слова: serverless, бессерверная архитектура, облачные сервисы, веб-приложения, Amazon Web Services

DEVELOPMENT OF SERVERLESS WEB-APPLICATIONS USING AMAZON WEB SERVICES TOOLS

Ganzha A.Yu., Karelova R.A.

Nizhny Tagil Technological Institute (branch) UrFU, Nizhny Tagil, e-mail: riya2003@mail.ru

This article is devoted to an overview of the main features of a serverless architecture as a way to implement web applications. Serverless architecture implies that the infrastructure of the software is supported by third-party providers, and the necessary functionality is provided in the form of services responsible for the processes of authentication, sending notifications and integration with third-party resources. The article discusses the key features of the structure of serverless web applications, as well as the implementation of the BaaS (Backend-as-a-Service) and FaaS (Function-as-a-Service) service models, reveals the advantages of a serverless architecture compared to the client-server implementation of the web.-applications. This presentation describes the purpose of Amazon Web Services platform cloud services such as API Gateway, Lambda, and DynamoDB. As an example of sharing AWS cloud platform tools, we demonstrate how to implement a flexible API for a serverless blogging web application. There is a description through processes such as creating DynamoDB database tables, declaring an HTTP API, and implementing POST and GET request handlers. There are code examples in the form of listings. The result of this work is the systematization of theoretical information and demonstration of practical techniques for developing serverless web applications using Amazon Web Services tools.

Keywords: serverless, serverless architecture, cloud serviced, web applications, Amazon Web Services

Многие разрабатываемые в настоящее время веб-приложения обладают функциями аутентификации пользователя, хранения пользовательской информации, а также рассылки уведомлений (по e-mail и/или SMS). Реализация такого функционала, как правило, предполагает наличие клиентской (front-end) части веб-приложения, базы данных и серверной (back-end) его части. По мере расширения функциональных возможностей таких приложений увеличивается и количество ресурсов, необходимых для поддержания производительности системы.

Снижение затрат на разработку серверной части веб-приложения путем ее переноса в готовые облачные технологии может значительно сократить сроки ввода приложения в эксплуатацию, а также освободить ресурсы для работы над клиентской частью. Такой подход к разработке веб-приложений называется бессерверным.

На сегодняшний день существует множество инструментов, способных помочь в создании бессерверных веб-приложений. Наиболее эффективными из них являются облачные сервисы таких крупных компаний, как Google, Microsoft и Amazon.

Цель исследования заключалась в иллюстрации особенностей применения инструментов AWS для разработки бессерверных веб-приложений.

Материалы и методы исследования

В рамках работы был проанализирован и систематизирован опыт современных разработчиков программного обеспечения, отраженный в публикациях, в том числе тематических форумах.

Результаты исследования и их обсуждение

Бессерверная архитектура представляет собой технологическое решение, в рамках которого задачи управления инфраструктурой и вычислительные процессы веб-приложения выполняются сторонними поставщиками облачных услуг [1, с. 3].

Как правило, бессерверный подход подразумевает использование предварительно настроенных событий, а для оптимизации процесса развертывания веб-приложения применяется ряд распределенных облачных сервисов. При таком подходе обычно реализуются две взаимодополняющие сервисные модели:

– back-end как услуга (BaaS);

– функция как услуга (FaaS).

Модель BaaS предоставляет разработчикам возможность связывать веб-приложения с серверным облачным хранилищем и API, а также реализовывать такие функции, как аутентификация пользователей, хранение данных в облаке и рассылка извещающих уведомлений [2, с. 1].

Назначение модели BaaS состоит в том, чтобы вместо разработки и поддержки собственных сервисов использовать готовые решения, набор которых формирует универсальный back-end для любого проекта.

В свою очередь, модель FaaS предоставляет возможность вызова серверного кода без необходимости управления серверами. Идея данной модели заключается в том, чтобы разбить серверную часть веб-приложения на набор не фиксирующих состояние функций для обработки событий и запросов [3, с. 8].

Таким образом, совместное использование моделей BaaS и FaaS позволяет разработчикам решать сложные задачи по размещению данных, инфраструктуры и элементов бизнес-логики будущего продукта.

Рассмотрим подробнее вариант применения данных моделей на примере сравнения классической клиент-серверной и бессерверной архитектур.

При клиент-серверном подходе веб-приложение разбивается на три уровня: клиентский уровень, уровень серверной логики и уровень данных (рис. 1). Так, клиентский уровень представляет собой веб-интерфейс, с которым непосредственно взаимодействуют пользователи. Уровень серверной логики, в свою очередь, отвечает за поведение веб-приложения, служит для обработки запросов клиентской части. Уровень данных представляет собой хранилище, в котором размещаются данные приложения.

При таком подходе вся логика веб-приложения располагается на серверном уровне.

При бессерверном подходе процессы аутентификации и обращения к базе данных реализуются с помощью модели BaaS. Кроме того, часть серверной логики передается на клиентский уровень, и, таким образом, помимо отображения информации, он занимается отслеживанием сессии пользователя, навигацией на странице, а также чтением из базы данных. Обработкой данных в базе занимается функция модели FaaS [4, с. 17–18].

Вариант архитектуры бессерверного веб-приложения представлен на рис. 2.

Таким образом, можно выделить ряд характерных преимуществ бессерверной архитектуры. Первое из них – сокращение финансовых затрат на разработку и развертывание веб-приложения за счет оплаты только используемых ресурсов.

Вторым характерным преимуществом бессерверной архитектуры является высокая скорость разработки: отсутствие необходимости в работе над второстепенными задачами, например, такими как обслуживание инфраструктуры или синхронизация данных, способно упростить и ускорить процесс разработки веб-приложения. За счет этого достигается не только снижение затрат на разработку, но и сокращение времени выхода программного продукта на рынок [5, с. 308].

Рис. 1. Архитектура клиент-серверного веб-приложения

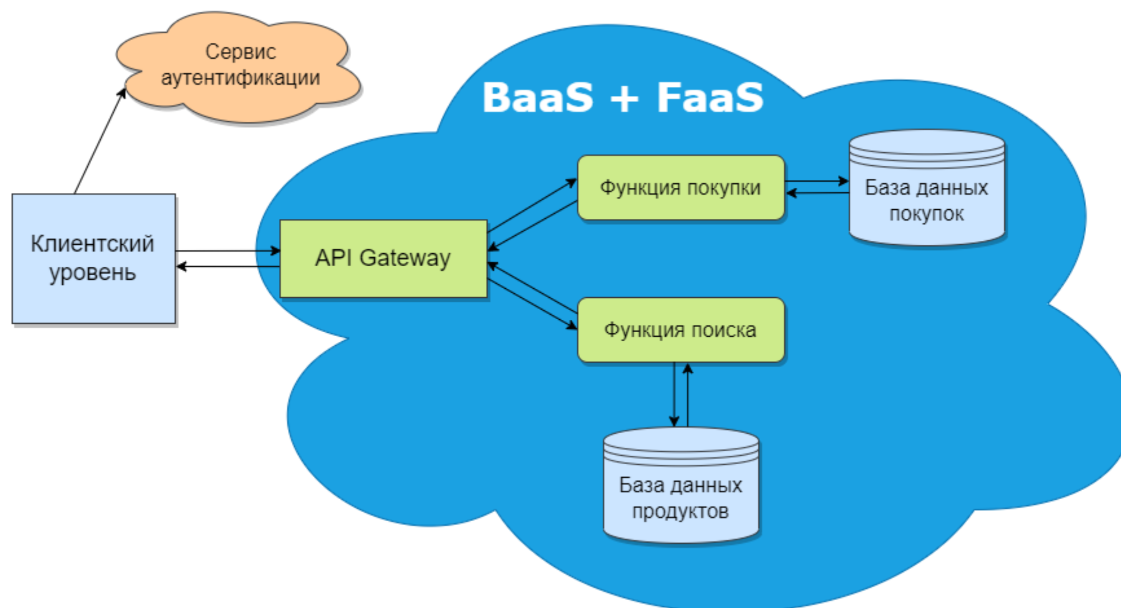

Рис. 2. Архитектура бессерверного веб-приложения

Еще одно преимущество такой архитектуры – автоматическое масштабирование. При использовании бессерверного подхода масштабирование вычислительных ресурсов и функциональных возможностей веб-приложения выполняется поставщиком бессерверной услуги именно тогда, когда в этом появляется необходимость.

Безопасность является одним из важнейших условий размещения веб-приложения в облачном сервисе. Так, возникает необходимость в обеспечении платформой трех основных аспектов безопасности: доступности, конфиденциальности и целостности данных.

Поставщики облачных услуг обеспечивают высокую доступность данных (в течение 99,5–99,9 % времени) [6]. Это означает, что потребители, имеющие доступ к информации, смогут получить её в нужное время.

Для обеспечения конфиденциальности пользовательских данных поставщики облачных услуг применяют шифрование. К наиболее распространенным способам шифрования относят использование криптографического протокола SSL, а также алгоритмов хеширования, таких как MD5 и SHA-2.

Также провайдеры облачных сервисов гарантируют обеспечение целостности пользовательских данных. Для этого клиентам предоставляются специальные возможности для резервирования данных веб-приложения, что позволяет сохранять целостность данных в резервной копии.

Перед началом создания любого бессерверного веб-приложения необходимо определиться с прикладными инструментами разработки.

На сегодняшний день существует множество различных инструментов для работы с бессерверной архитектурой, поддерживаемых такими крупными компаниями, как Google, Microsoft и Amazon. Наиболее гибким решением считается платформа Amazon Web Services (далее – AWS) компании Amazon.

Облачная платформа AWS предлагает надежные, масштабируемые и недорогие сервисы облачных вычислений, такие как AWS Lambda, AWS API Gateway и AWS DynamoDB. Рассмотрим каждый из этих сервисов подробнее.

AWS Lambda представляет собой вычислительный сервис, который отвечает за выполнение определенных функций в ответ на события в веб-приложении, например, на HTTP-вызовы или обновление таблицы в базе данных.

AWS API Gateway – это сервис, предназначенный для создания, публикации и обслуживания API. С помощью API реализуется доступ к данным и функциональным возможностям веб-приложения. API Gateway является основной частью бессерверного API, отвечающей за связь между определенным API и функцией, обрабатывающей запросы к этому API.

AWS DynamoDB представляет собой бессерверную базу данных NoSQL на осно-

ве пар «ключ – значение», предназначенную для хранения данных высоконагруженных веб-приложений [7, с. 3].

Совместное использование перечисленных сервисов способно значительно упростить разработку бессерверных веб-приложений, позволяя при этом создавать гибкий API для безопасного доступа, манипулирования и объединения данных из нескольких источников.

В качестве примера рассмотрим реализацию API веб-приложения для ведения блога. Для решения этой задачи необходимо зарегистрировать аккаунт AWS, а также установить фреймворк AWS Serverless Application Model (далее – SAM), предназначенный для создания и администрирования бессерверных веб-приложений.

После установки SAM необходимо перейти в директорию проекта и инициализировать его следующей командой (листинг 1).

Листинг 1. Инициализация проекта

```
sam init -r nodejs12.x -n blog
```

После выполнения команды в директории проекта появится каталог с установочными файлами. В файле template.yml необходимо определить таблицу DynamoDB, в которой будут храниться записи блога (листинг 2).

Листинг 2. Объявление таблицы DynamoDB

```
Resources:
  BlogPostsTable:
    Type: AWS::DynamoDB::Table
      // Объявление свойств таблицы
    Properties:
            // Название объявляемой таблицы
      TableName: blog-posts-table
            // Объявление атрибутов таблицы
      AttributeDefinitions:
              // Создание атрибута partKey
        - AttributeName: partKey
              // Тип атрибута partKey – строковый
          AttributeType: S
              // Создание атрибута createdAt
        - AttributeName: createdAt
              // Тип атрибута createdAt – числовой
          AttributeType: N
        // Объявление первичных ключей таблицы
      KeySchema:
            // Создание ключевого атрибута partKey
        - AttributeName: partKey
            // Тип ключевого атрибута partKey – ключ партиции
          KeyType: HASH
            // Создание ключевого атрибута createdAt
        - AttributeName: createdAt
            // Тип атрибута createdAT – ключ сортировки
          KeyType: RANGE
        // Установка режимов чтения и записи для таблицы
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
```

Платформа AWS создаст таблицу DynamoDB «blog-posts-table», где ключом партиционирования будет выступать атрибут partKey, а ключом диапазона – атрибут createdAt.

Далее необходимо объявить HTTP API, связывающий все конечные точки и функции веб-приложения (листинг 3).

Листинг 3. Объявление HTTP API веб-приложения

```
// Объявление HTTP API веб-приложения
BlogHttpApi:
  Type: AWS::Serverless::HttpApi
  // Объявление свойств создаваемого API
  Properties:
    // Объявление пути API
    StageName: Test
    CorsConfiguration: True
```

Объявление функции представлено на листинге 4.

Листинг 4. Объявление функций

```
// Объявление функции CreatePostFunction
CreatePostFunction:
    Type: AWS::Serverless::Function
     // Указание свойств объявляемой функции
    Properties:
      // Указание пути к функции
      Handler: src/handlers/createPost.handler
       // Указание среды выполнения лямбда-функции
      Runtime: nodejs12.x
       // Установка ограничения по использованию оперативной памяти на 128 МБ
      MemorySize: 128
       // Установка пятисекундного таймаута для функции
      Timeout: 5
       // Объявление события CreatePost
      Events:
        CreatePost:
              // Указание типа события HttpApi
          Type: HttpApi
              // Указание свойств события
          Properties:
              // Привязка события к созданному BlogHttpApi
            ApiId: !Ref BlogHttpApi
              // Указание, при каком запросе будет вызываться событие
            Method: POST
              // Указание пути вызова запроса
            Path: /posts
      Policies:
        - AmazonDynamoDBFullAccess

 // Объявление функции GetPostsFunction
 GetPostsFunction:
    Type: AWS::Serverless::Function
// Указание свойств объявляемой функции
    Properties:
 // Указание пути к функции
      Handler: src/handlers/getPosts.handler
       // Указание среды выполнения лямбда-функции
      Runtime: nodejs12.x
       // Установка ограничения по использованию оперативной памяти на 128 МБ
      MemorySize: 128
       // Установка пятисекундного таймаута для функции
      Timeout: 5

  // Объявление события GetPosts
      Events:
        GetPosts:
 // Указание типа события HttpApi
          Type: HttpApi
              // Указание свойств события
```

```
      Properties:
            // Привязка события к созданному BlogHttpApi
        ApiId: !Ref BlogHttpApi
            // Указание, при каком запросе будет вызываться событие
        Method: GET
            // Указание пути вызова запроса
        Path: /posts
    Policies:
      - AmazonDynamoDBFullAccess
```

Функции CreatePostFunction и GetPostsFunction выступают в роли обработчиков для запросов POST и GET.

Программный код обработчиков располагается в директории «src/handlers». Для этого в данной директории создается файл «createPost.js», в котором описывается алгоритм работы обработчика POST-запроса (листинг 5).

Листинг 5. Программный код обработчика POST-запроса

```
// Подключение необходимых модулей
const AWS = require('aws-sdk');
const dynamodb = new AWS.DynamoDB();

// Объявление обработчикаPOST-запроса
exports.handler = async (event) => {
  // Получение тела запроса
  const { body } = event;
  try {

    // Получение названия и тела записи из тела запроса
    const { title, text } = JSON.parse(body);
    // Возврат ошибки, если отсутствует название или текст записи
    if (!title || !text) {
      return {
        statusCode: 403,
        body: 'title and text are required!'
      }
    }
    // Если название и текст записи были получены, то занесение их в созданную таблицу DynamoDB
    await dynamodb.putItem({
        // Указание названия таблицы, в которую помещаются данные о записи
      TableName: 'blog-posts-table',
        // Указание элементов для отправки данных в таблицу
      Item: {
        postId: { S: 'blog' },
        title: { S: title },
        text: { S: text },
        createdAt: { N: String(Date.now()) }
      }
    }).promise();
    return {
        // Возврат сообщения о том, что запись успешно создана
      statusCode: 200,
      body: 'Post was created!',
    }
  } catch (err) {
    return {
        // Возврат сообщения об ошибке в случае сбоя
      statusCode: 500,
      body: 'Something went wrong!',
    }
  }
};
```

При обработке POST-запроса данная функция получает из тела запроса название и содержимое записи блога, а затем сохраняет их в базу данных DynamoDB.

Аналогичным образом создается файл «getPosts.js» для обработчика GET-запроса, с помощью которого будут получены пять последних записей блога (листинг 6).

Листинг 6. Программный код обработчика GET-запроса

```
// Подключение необходимых модулей
const AWS = require('aws-sdk');
const dynamodb = new AWS.DynamoDB();

// Объявление обработчика GET-запроса
exports.handler = async () => {
  try {
    // Создание запроса на получение записей блога
const result = await dynamodb.query({
 // Название таблицы, из которой будут получены записи
     TableName: 'blog-posts-table',
       // Объявление ключа, по которому будут получены записи
     KeyConditionExpression: 'partKey = :partKey',
       // Отображение последних записей сверху
     ScanIndexForward: false,
       // Указание лимита на кол-во возвращаемых записей
     Limit: 5,
     ExpressionAttributeValues: {':partKey': { S: 'blog'}}
    }).promise();
    return {
       // Возврат полученных записей в случае успешной обработки запроса
     statusCode: 200,
     headers: {
       'Content-Type': 'application/json',
     },
     body: JSON.stringify(result.Items),
    }
  } catch (err) {
    console.log(err);
    return {
       // Возврат сообщения об ошибке в случае сбоя
     statusCode: 500,
     body: 'Something went wrong!',
    }
  }
};
```

Для запуска созданного API необходимо развернуть его с помощью соответствующей команды (листинг 7).

Листинг 7. Команда развертывания API

```
sam deploy – guided
```

Таким образом, с помощью облачных сервисов AWS API Gateway, AWS Lambda и AWS DynamoDB был реализован простой API бессерверного веб-приложения для ведения блога (рис. 3).

Заключение

Применение бессерверной архитектуры является перспективным подходом к созданию веб-приложений. Это обусловлено такими преимуществами, как снижение затрат на разработку и развертывание веб-приложения, а также сокращение сроков выхода программного продукта на рынок. Кроме того, перспективность бессерверной архитектуры объясняется продвижением и поддержкой облачных технологий мировыми IT-гигантами. Одним из таких крупных поставщиков облачных сервисов является компания Amazon.

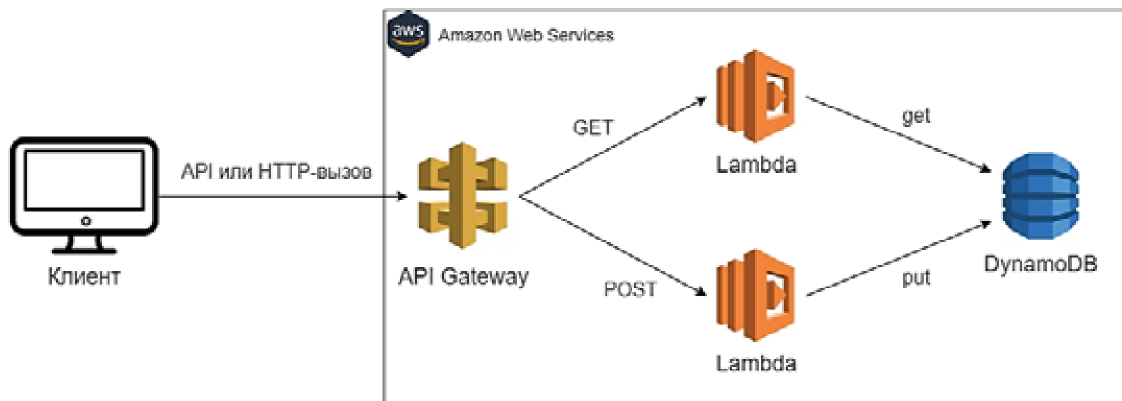

Рис. 3. Принцип работы реализованного API бессерверного веб-приложения

На сегодняшний день платформа Amazon Web Services, разработанная этой компанией, считается одним из лучших решений в работе с бессерверной архитектурой, предоставляющим такие инструменты, как AWS API Gateway, AWS Lambda и AWS DynamoDB. Использование перечисленных сервисов способно значительно упростить процесс разработки бессерверных веб-приложений, делая их гибкими, надежными и производительными.

Список литературы

1. Дешко И.П., Кряженков К.Г., Тулинов С.В., Цветков В.Я. Микросервисы и serverless платформы: учебное пособие. М.: МАКС Пресс, 2020. 64 с.

2. Мархакшинов А.Л., Тонхоноева А.А., Урмакшинова Е.Р. Разработка бессерверных мобильных приложений // Вестник Новосибирского государственного университета. Серия: Информационные технологии. 2019. Т. 17. № 4. С. 66-73.

3. Дешко И.П., Кряженков К.Г., Тулинов С.В., Цветков В.Я. Основы serverless: учебное пособие. М.: МАКС-Пресс, 2020. 72 с.

4. Казаков В.Е., Кузнецов А.А., Мурычева В.В. Архитектура serverless // Материалы докладов 54-й международной научно-технической конференции преподавателей и студентов. 2021. С. 16–18.

5. Пантелеев А.С., Соловьев Т.Г. Особенности разработки serverless приложений // Математика и математическое моделирование. 2021. С. 307–308.

6. 99.99 % uptime for Azure Active Directory [Электронный ресурс]. URL: https://techcommunity.microsoft.com/t5/azure-active-directory-identity/99-99-uptime-for-azure-active-directory/ba-p/1999628 (дата обращения: 20.11.2021).

7. Макосий А.И., Макосий Р. Современная облачная инфраструктура: бессерверные вычисления // Вестник Хакасского государственного университета им. Н.Ф. Катанова. 2019. № 2. С. 13–16.