ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

DOI: 10.18137/RNU.V9187.22.02.P.140 УДК 004

А.Т. Гордина, А.В. Забродин, А.Д. Хомоненко

ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЙ SERVERLESS-АРХИТЕКТУРЫ[1]

Аннотация. Рассматривается модель бессерверной архитектуры облачных вычислений. Проведено сравнение с традиционной многоуровневой архитектурой. Рассмотрено использование API-шлюза – связующего звена между клиентом и сервисами. Приведен пример спецификации приложения по управлению грузоперевозками. Рассмотрено внедрение бессерверной архитектуры в микросервисную. Определены принципы проектирования основных компонентов модели FaaS – функция как сервис – составляющей бессерверных вычислений. Обосновано создание бессерверной архитектуры и задач, решаемых при разработке приложения. В качестве провайдера облачных услуг выбран Yandex.Cloud.

Ключевые слова: бессерверная архитектура, микросервисы, FaaS-модель, функции, API-шлюз, принципы проектирования.

A.T. Gordina, A.V. Zabrodin, A.D. Khomonenko

DESIGNING SERVERLESS ARCHITECTURE APPLICATIONS

Abstract. The latest model of serverless cloud computing architecture is considered. A comparison with the traditional multi-level architecture is made. The use of an API gateway, a link between the client and services, is considered. An example specification for a freight management application is provided. The introduction of a serverless architecture into a microservice one is considered. The principles of designing the main components of the FaaS model – a function as a service – a component of serverless computing are determined. The creation of a serverless architecture and the tasks to be solved during the development of the application are substantiated. Yandex. Cloud was chosen as the cloud service provider.

Keywords: serverless architecture, microservices, FaaS model, functions, API-Gateway, design principles.

Введение

Архитектура приложения – это совокупность решений по проектированию программного приложения. Существует множество видов архитектур, каждому из которых свойственны свои подходы и методы в структуре отношений между компонентами программных систем [1]. Хорошо спроектированная архитектура делает процесс разработки программного обеспечения и дальнейшее его сопровождение более простым и эффективным.

В традиционном подходе разработки программного обеспечения (далее – ПО) на разработчика возлагаются все задачи по управлению инфраструктурой: обслуживание, масштабирование серверов, балансировка нагрузки между ними и многое другое. Поэтому в последнее время из процесса разработки ПО стали исключаться операции, направленные

[1] Исследования по теме статьи проводились в рамках реализации Федеральной программы поддержки университетов «Приоритет 2030».

Гордина Анна Тимофеевна
бакалавр, инженер-программист ООО «МТС Диджитал», Санкт-Петербург. Сфера научных интересов: проектирование и разработка программного обеспечения; объектно ориентированное программирование; облачные вычисления. Автор 1 опубликованной научной работы.
Электронный адрес: gordina.23@mail.ru

Забродин Андрей Владимирович
кандидат исторических наук, доцент кафедры информационных и вычислительных систем. Петербургский государственный университет путей сообщения императора Александра I, Санкт-Петербург. Сфера научных интересов: моделирование информационных систем; объектно ориентированный анализ и проектирование; программирование; базы данных; архитектура вычислительных систем. Автор более 20 опубликованных научных работ.
Электронный адрес: ivs@pgups.ru

Хомоненко Анатолий Дмитриевич
доктор технических наук, профессор, профессор кафедры информационных и вычислительных систем. Петербургский государственный университет путей сообщения императора Александра I, Санкт-Петербург. Сфера научных интересов: базы данных; интеллектуальные системы и технологии; информационные технологии в сфере безопасности; моделирование информационных систем; программирование. Автор более 250 опубликованных научных работ.
Электронный адрес: ivs@pgups.ru

на управление серверами. Такому подходу дали название бессерверный (Serverless), то есть подход, который позволяет «инкапсулировать» разработчика от работы, связанной с сопровождением серверов [2]. Приложение по-прежнему работает на сервере, но все задачи по его управлению берет на себя поставщик облачных услуг. Serverless предполагает свой особенный подход к проектированию архитектуры, например, использование сервисов, которые позволяют снизить сложность запуска и управление приложением.

Многоуровневая архитектура

Многоуровневая архитектура является одной из самых распространенных архитектур и служит фундаментальным шаблоном проектирования приложений; помимо этого, она предоставляет основу для модели архитектуры системы управления базами данных (далее – СУБД) и сетевой модели OSI. Главная идея многоуровневой архитектуры заключается в разделении программного обеспечения на уровни так, чтобы изменения отдельного уровня не затрагивали другие уровни. Такой подход удобен в обслуживании и контролировании различных модулей приложения, как по отдельности, так и групп, представляющих взаимосвязанный набор сервисов.

Наиболее распространенной архитектурой с использованием многоуровневого шаблона является трехуровневая архитектура [6]. Она состоит из уровня представления, логического уровня и уровня данных (см. Рисунок 1).

Уровень представления является компонентом, с которым взаимодействует пользователь. Этот уровень реализует пользовательский интерфейс.

Логический уровень содержит бизнес-логику приложения, то есть включает в себя правила и принципы, по которым приложение взаимодействует с пользователем.

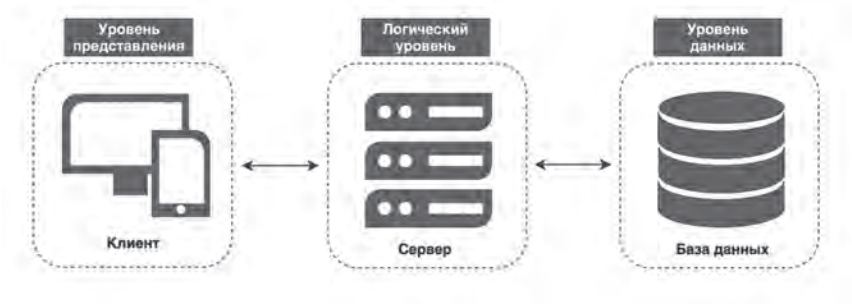

Рисунок 1. Традиционная архитектура трехуровневого приложения

Уровень данных – уровень хранения. Здесь находятся основные хранилища, например, базы данных, файловые системы и др., необходимые для размещения данных приложения.

Логический уровень может быть размещен на выделенном физическом сервере, и при повышенной нагрузке появляется необходимость в масштабирование его ресурсов. Serverelss является технологией без управления серверами, которая обеспечивает автоматическую оптимизацию, поэтому проектирование бессерверной трехуровневой архитектуры может дать существенные преимущества. При переходе на Serverless необходимо разбить приложение на микрозадачи, под каждую из которых будет спроектирована функция [7] – фрагмент программы, реагирующий на событие и обрабатывающий его. Коммуникация между клиентом и набором функций происходит посредством API-шлюза (см. Рисунок 2).

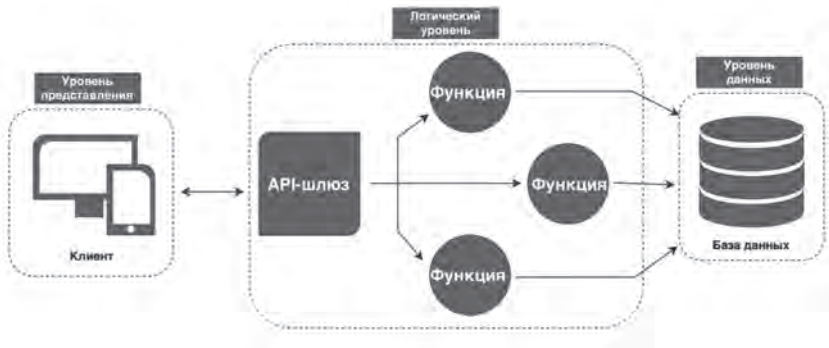

Рисунок 2. Serverless-архитектура

Совместное использование функций и API-шлюза позволяет создать бессерверное приложение с высоким уровнем доступности и безопасности без необходимости управления сервером и масштабирования.

Использование API-шлюза и функций

API-шлюз (API Gateway) – это инструмент управления, который является связующим звеном между клиентом и внутренними сервисами приложения.

Шлюз API маскирует внутреннюю реализацию от внешнего наблюдателя, обеспечивая единую точку входа для каждого API. По сути API Gateway является RESTful API. REST API представляет собой набор методов, интегрированных с конечными точками HTTP, функциями и прочими сервисами приложения.

Предположим, у нас есть приложение для грузоперевозок, которое использует Serverless-сервисы поставщика облачных услуг Yandex. Cloud. Пользователю необходимо управлять грузами и их перевозками. Каждый API для работы с ними реализован с помощью Yandex Cloud Functions.

Рассмотрим схему работы бессерверного логического уровня приложения. Конечными точками являются перевозки (movements) и грузы (goods). От пользователя поступает запрос GET или POST к конечным точкам, Yandex API Gateway обрабатывает его и направляет к функциям. В этом случае реализованы функции по поиску грузов, созданию перевозок и их отслеживанию (см. Рисунок 3).

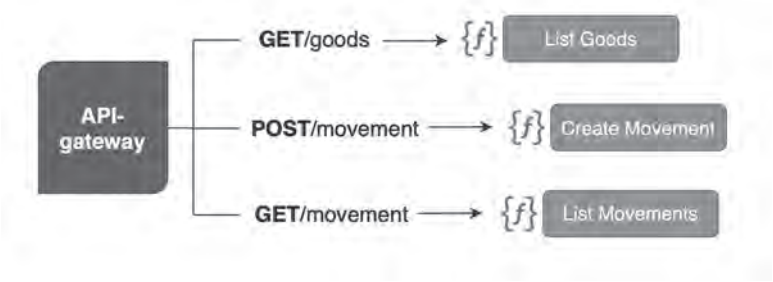

Рисунок 3. Работа API-шлюза

Основной стандарт при реализации REST API – OpenAPI версии 3.0 [10]. OpenAPI представляет собой интерфейс по взаимодействию пользователей с сервисами и позволяет понимать их возможности без необходимости доступа к исходному коду. Пример спецификации для приложения по управлению грузоперевозками приведен на Рисунке 4.

```
openapi: 3.0.0
info:
  title: Freight Movement API
  version: 1.0.0
paths:
    /goods:
      get:
        x-yc-apigateway-integration:
          type: cloud-functions
          function_id: list_goods_function
        operationId: listGoods
    /movements:
      get:
        x-yc-apigateway-integration:
          type: cloud-functions
          function_id: list_movements_function
        operationId: listMovements
      post:
        x-yc-apigateway-integration:
          type: cloud-functions
          function_id: create_movements_function
        operationId: createMovement
```

Рисунок 4. Спецификация API-шлюза

Пути вызова – /goods и /movements. Далее указывается метод get/post. Расширение x-yc-apigateway-integration является точкой входа для интеграции Yandex API Gateway с

другими сервисами. Поле type описывает тип расширения для действий при вызове по указанному пути. При вызове облачных функций, указанных в поле function_id, в поле type указывается cloud-functions.

Таким образом, API-шлюз упрощает соединение конечных точек HTTP и отдельных функций, что избавляет от необходимости в выделении специального API-сервера. Кроме того, так инкапсулируется функциональность приложения в каждой функции и разделяется бизнес-логика различных частей API.

Микросервисы и Serverless

Serverless-архитектура схожа с микросервисной; при ее проектировании задачи также разбиваются на независимые друг от друга микрозадачи [9], каждая из которых работает в собственном процессе (см. Рисунок 5). По сути бессерверная архитектура – это расширенная микросервисная архитектура.

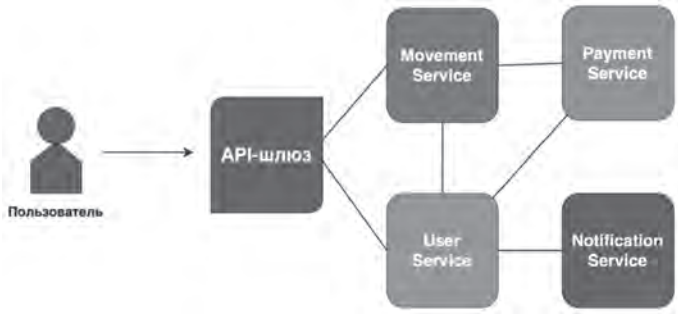

Рисунок 5. Микросервисная архитектура

Несмотря на сходство этих двух архитектур, их основным отличием является подход к проектированию микрочастей приложения. Микросервис может выполнять несколько задач; чаще всего он работает по системе «запрос – ответ», в то время как функция однонаправленная и выполняет только одну задачу [8]. Это значит, что микросервис может быть равен одной или нескольким Serverless-функциям. Помимо этого микросервисы относятся к сервисно ориентированной архитектуре (далее – СОА), а serverless продолжает развитие в событийно ориентированном мире архитектуры, что дает успехи и явные преимущества в сокращении времени выхода ПО на рынок и снижении эксплуатационных расходов.

Одним из распространенных решений является объединение двух архитектурных подходов, чтобы в полной мере использовать преимущества облака, так как данные технологии играют важную роль в облачных вычислениях. Кроме того, микросервисы могут быть развернуты как часть Serverless. Однако важно помнить, что приложения Serverless разворачиваются только внутри инфраструктуры выбранного поставщика облачных услуг.

При создании микросервисов Serverless каждый микросервис в приложении может быть разбит на одну или несколько функций (см. Рисунок 6).

Основным преимуществом Serverless-функций является простота их комбинирования с другими сервисами приложения – базами данных, очередями сообщений, инструментами управления API. Такая гибкость уменьшает количество повторяемого и шаблонного кода, который дополняется по мере масштабирования системы. Гибридная микросервисная Serverless-архитектура – разумный вариант для программного продукта, в ко-

тором микросервисы выполняют обработку тяжелых процессов, а serverless решает более простые задачи.

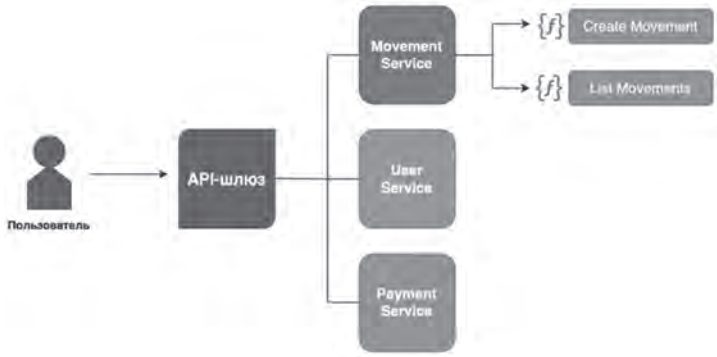

Рисунок 6. Микросервисная Serverless-архитектура

Принцип единой ответственности и принцип разделения ответственностей

При проектировании функции или микросервиса важно думать о понятии связности – меры когерентности элементов и группировании связного кода. Данное утверждение можно спроецировать на сформулированный Робертом Мартином принцип единой ответственности (Single Responsibility Principle, SRP), который гласит [5]: «Соберите вместе те вещи, которые меняются по одной и той же причине, и отделите те вещи, которые меняются по разным причинам». Иными словами, ответственность – это «причина для изменения», то есть каждая подсистема, класс, функция не должны иметь более одной причины для изменения. Каждая Serverless-функция должна быть нетривиальной и иметь уникальную ответственность.

При разработке Serverless-архитектуры, следуя принципу SRP, сложную функциональность модуля или микросервиса можно разбить на несколько управляемых функций, каждая из которых получает запрос от клиента, выполняет свою ответственность, генерирует вывод и передает его следующей функции. Таким образом, получается цепочка задач.

Вернемся к нашему примеру про сервис грузоперевозок. Когда при создании перевозки она будет помечена как готовая к отправлению, будет создано событие, которое вызовет функцию для проверки терминала отправки клиента. Далее последует другая функция для генерации отгрузочной метки для перевозки. После произойдет подтверждение перевозки отправлением уведомления на email клиента (см. Рисунок 7).

Разделение ответственностей (Separation of Concerns, SoC) – подход проектирования программы, который предполагает, что каждый ее функциональный блок должен как можно меньше дублировать функциональность другого блока. Принцип разделения ответственностей обеспечивает повторное использование функций, индивидуальную оптимизацию и изолирование от допустимого отказа других функций. Разделение задач обеспечивается за счет инкапсуляции внутри каждого блока кода, который имеет свой определенный интерфейс. Инкапсуляция – принцип, который позволяет объединить данные и функции по их обработке [3]. Это позволяет улучшать отдельные разделы программы и вносить изменения без ущерба на реализацию сторонних модулей.

Крис Рид сделал вывод, что идеи в подходе дифференциации задач сводятся к трем основным [4]: 1) описание вычислений, 2) организация последовательных вычислений на

небольшие шаги, 3) организация управления памятью во время вычислений. Он отмечает [4]: «В идеале программист должен иметь возможность сосредоточиться на первой из трех задач, не отвлекаясь на две другие, более административные, задачи. Очевидно, что администрирование важно, но, отделив его от основной задачи, мы, вероятно, получим более надежные результаты и сможем облегчить проблему программирования, автоматизируя большую часть администрирования».

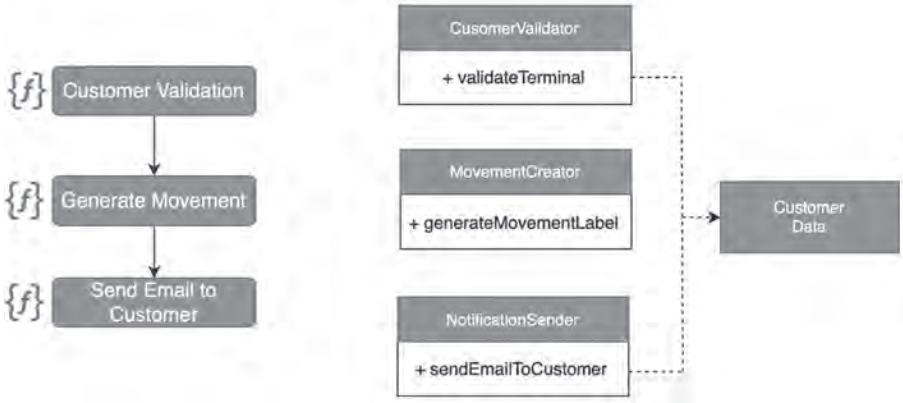

Рисунок 7. Принцип SRP для цепочки Serverless-функций

Таким образом, администрирование составляет, бесспорно, необходимую часть разработки, но автоматизация данных аспектов принесет больше эффективности и преимуществ.

В традиционной архитектуре могут возникать конфликты при чтении и записи из одного в одно и то же хранилище данных. Функции Serverless могут избавить систему от подобных проблем. Например, есть сервис, отвечающий за грузы на складе. Клиенту необходимо изменять, удалять и читать данные из этого сервиса. Для этого организуем функции под каждый необходимый запрос: добавление, удаление, чтение (см. Рисунок 8).

```
public class AddGoodsFunction {
    @PutMapping("/add")
    public void addGoods …
}

public class DeleteGoodsFunction{
    @DeleteMapping("/delete")
    public void deleteGoods …
}

public class ListGoodsFunction {
    @GetMapping("/list")
    public void listGoods …
}
```

Рисунок 8. REST API с применением SoC для функций

В правильном проектировании функций помимо принципа разделения ответственностей также помогает принцип единой ответственности. Такой подход в больших проектах позволяет системе быть хорошо масштабируемой, поскольку каждая команда работает

над разными задачами, и Serverless только помогает применять данные принципы в проектировании архитектуры приложений.

Заключение

Serverless-архитектура позволяет создавать адаптивные приложения, которые быстро обрабатывают запросы пользователей, потребляют мало ресурсов и не требуют управления серверами. Такой способ при разработке ПО показывает свою эффективность как в архитектуре, построенной полностью на Serverless-стеке, так и при комбинировании его с архитектурными подходами, которые давно получили широкое распространение.

Основной частью бессерверных технологий являются функции, которые обладают следующими свойствами.

Завершенность и целостность – каждая функция выполняет только свою ответственность.

Один вход и один выход – событие вызывает функцию, которая получает набор данных, выполняет логику по их обработке и возвращает результат.

Логическая независимость – работа функции не зависит от работы других модулей и функций.

На основании вышесказанного, несмотря на то, что Serverless требует особенного подхода к проектированию архитектуры, бессерверная архитектура продолжает набирать популярность, доказывая свою эффективность и развивая сферу бессерверных облачных вычислений, продвигая разработку ПО на новый уровень.

Литература

1. Архитектура информационных систем: учебник для учреждений высшего профессионального образования / Б.Я. Советов, А.И. Водяхо, В.А. Дубенецкий, В.В. Цехановский. М.: Академия, 2012. 288 с.
2. *Гордина А.Т., Забродин А.В.* Особенности технологий бессерверных вычислений // Интеллектуальные технологии на транспорте. 2022. № 1. С. 16–23.
3. Модели и методы исследования информационных систем: монография / А.Д. Хомоненко, А.Г. Басыров, В.П. Бубнов, А.В. Забродин. СПб.: Лань, 2019. С. 82–123
4. *Chris Reade* (1989) Elements of Functional Programming. International Computer Science Series. Addison-Wesley, January 1, 600 p.
5. *Kevlin Henney* (2010) 97 Things Every Programmer Should Know. O'Reilly Media, February, pp. 152–153.
6. *Mark Richards* (2017) Software Architecture Patterns, O'Reilly Media, Iss. 3, 53 p.
7. *Peter Sbarski, Yan Cui, Ajay Nair* (2022) Serverless Architectures on AWS. Manning Publications, February, 256 p. ISBN 9781617295423.
8. *Sam Newman* (2015) Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 278 p.
9. *Thomas Smart* (2020) Serverless Beyond the Buzzword: What Can Serverless Architecture Do for You? Partridge Publishing Singapore, 370 p.
10. Yandex. Cloud Yandex API Gateway. Available at: https://cloud.yandex.ru/services/api-gateway(date of the application: 05.04.2022).

References

1. Sovetov B.Ya., Vodyakho A.I., Dubenetsky V.A., Tsekhanovsky V.V. (2012) *Arhitektura informacionnyh sistem* [Architecture of information systems]. Moscow, Akademija Publishing, 228 p. (in Russian).

2. Gordina A.T., Zabrodin A.V. (2022) *Osobennosti tehnologij besservernyh vychislenij* [Features of serverless computing technologies]. *Intellektual'nye tehnologii na transporte*, No. 1, pp. 16–23 (in Russian).
3. Homonenko A.D., Basyrov A.G., Bubnov V.P., Zabrodin A.V. (2019) *Modeli i metody issledovanija informacionnyh sistem* [Models and methods of information systems research]. St. Petersburg, Lan Publishing, pp. 82–123 (in Russian).
4. Chris Reade (1989) Elements Of Functional Programming. International Computer Science Series. Addison-Wesley, January 1, 600 p.
5. Kevlin Henney (2010) 97 Things Every Programmer Should Know. O'Reilly Media, February, pp. 152–153.
6. Mark Richards (2017) Software Architecture Patterns, O'Reilly Media, Iss. 3, 53 p.
7. Peter Sbarski, Yan Cui, Ajay Nair (2022) Serverless Architectures on AWS. Manning Publications, February, 256 p. ISBN 9781617295423.
8. Sam Newman (2015) Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 278 p.
9. Thomas Smart (2020) Serverless Beyond the Buzzword: What Can Serverless Architecture Do for You? Partridge Publishing Singapore, 370 p.
10. Yandex. Cloud Yandex API Gateway. Available at: https://cloud.yandex.ru/services/api-gateway(date of the application: 05.04.2022).