DOI: 10.24412/2413-2527-2022-129-16-23

Особенности технологий бессерверных вычислений

А. Т. Гордина, к.и.н. А. В. Забродин
Петербургский государственный университет путей сообщения Императора Александра I
Санкт-Петербург, Россия
anngordtm@gmail.com, teach-case@yandex.ru

Аннотация. **Статья посвящена одному из перспективных направлений в области облачных вычислений, известному под названием Serverless Computing (бессерверные вычисления). Рассматриваются технологии, применяемые для Serverless, дается сравнение облачных моделей и поставщиков услуг, описание основных компонентов архитектурного шаблона FaaS (функций, ресурсов и событий). Показаны преимущества и недостатки технологии Serverless-вычислений, а также раскрываются проблемы миграции приложений с локальных серверов в облако, и как при этом из процесса разработки программного продукта исключаются задачи по управлению ресурсами и их оптимизации.**

Ключевые слова: **облачные вычисления, бессерверные вычисления, модель FaaS, функции, автомасштабирование, архитектура приложения.**

Введение

Для функционирования и организации работы любой информационной системы необходима инфраструктура, которая позволит объединить, организовать и трансформировать информацию в соответствии с поставленными задачами. Инфраструктура является комплексом взаимосвязанных информационных сервисов и систем, таких как специализированное программное обеспечение, серверное, сетевое и компьютерное оборудование, СУБД, каналы связи, персонал и многое другое, необходимых для функционирования и развития инструментов информационного взаимодействия в компании. Создание эффективной инфраструктуры — это довольно сложный процесс, требующий значительных ресурсов (как финансовых, так и временных), а также высокого уровня компетенций. Одним из направлений снижения капитальных затрат на инфраструктуру, а также повышения гибкости информационных задач и процессов является использование облачных технологий (хостинга). Период пандемии и массовый переход на удаленную работу способствовали росту востребованности облачных платформ.

Несомненно, миграция в облако также приносит проблемы в планировании производительности и управлении операциями серверной части [1]. Так провайдеры облачных услуг стали развивать решения, в которых облако автоматически и динамически занимается распределением вычислительных ресурсов. Одним из таких решений является стратегия бессерверных вычислений. В парадигме Serverless весь процесс разработки программного обеспечения превращается в совокупность рабочих процессов с администрированием извне, что позволяет не акцентировать внимание на инфраструктуре вокруг, а сфокусироваться на самом процессе разработки. Так, Serverless решения становятся следующим этапом развития архитектуры вычислительных систем в эпоху облаков.

Модель обслуживания «как услуга»

Сущность облачного хостинга заключается в том, что физические серверы провайдера соединяются в единый пул, мощности которого могут гибко распределяться между клиентами в зависимости от их необходимостей.

Облачный рынок в настоящее время продолжает развиваться, расширяются модели облачных вычислений [2]. При планировании использования облаков нужно понимать, какие облачные сервисы существуют, чтобы остановить свой выбор на правильном и наиболее подходящем решении в пользу той или иной модели работы, что позволит найти необходимый баланс между гибкостью и простотой.

Неотъемлемыми характеристиками при выборе инфраструктурной платформы являются:

- увеличение скорости и качества разработки;
- автоматизация обслуживания и эксплуатации;
- безопасность и стабильность.

Концепция «как услуга» (англ. *as-a-Service*, *aaS*) подразумевает использование услуги с применением облачных технологий по необходимости. Каждая модель IT-инфраструктуры, которая поставляется в этом формате, уникальна по-своему. Рассмотрим самые востребованные aaS-сервисы, представленные на рынке облачных вычислений.

Модель обслуживания «инфраструктура как услуга» (англ. *Infrastructure-as-a-Service*, *IaaS*) стала первым шагом в модернизации IT-инфраструктуры при переходе от физических к виртуальным серверам. В 2012 году поставщики «инфраструктуры как услуги» занимали ведущие позиции в развитии облачных вычислений [3].

IaaS-провайдеры располагают услугами разного масштаба. Есть возможность арендовать единственный виртуальный сервер или пул виртуальных серверов с возможностью их объединения в виртуальную сеть.

Парадигма IaaS предоставляет полные административные права внутри арендованных виртуальных серверов. Все, что касается настройки операционных систем арендованных серверов, необходимо выполнять самостоятельно. Изначально IaaS-провайдер лишь гарантирует, что ваш сервер будет доступен по сети в соответствии с соглашением об уровне услуг (англ. *Service Level Agreement*, *SLA*) [4].

Преимуществом в выборе IaaS является тот факт, что такая модель подходит для случаев, когда значение имеет масштабируемость. IaaS также может быть полезна для приложений с постоянной нагрузкой или необходимостью наращивать вычислительные ресурсы.

Основной недостаток IaaS: по-прежнему остается необходимость и ответственность за управление безопасностью. Кроме того, IaaS-провайдеры предоставляют только серверы и API, а настройкой всего остального приходится заниматься самостоятельно.

Создание качественного программного обеспечения зависит от информационно-технологических платформ: среды разработки и тестирования, операционных систем, средств для развертывания, инструментов управления базами данных. Модель «платформа как услуга» (англ. *Platform-as-a-Service*, *PaaS*) позволяет размещать готовые программы на хостинге с поддержкой всего необходимого. Комплект сервисов PaaS передает ответственность по управлению всей информационно-технологической инфраструктурой провайдеру. Облачный провайдер берет на себя установку всех зависимостей операционной системы, необходимых для поставляемой версии среды выполнения. Остается только сосредоточиться на создании приложения, которое работает в выбранной среде выполнения.

Zimki (2006) и Heroku (2007) были первыми компаниями, которые использовали модель разработки «платформа как услуга» [5]. В то время было ясно, что такой подход значительно ускоряет время разработки и сокращает расходы на обслуживание. Это позволяет разработчикам настраивать и управлять приложениями, значительно сокращая время от начала разработки до выпуска продукта и последующей его поддержки.

Недостаток PaaS-модели заключается в том, что управление сфокусировано лишь на том, что построено на самой платформе. Провайдер берет ответственность за всю физическую инфраструктуру, а также администрирование на уровне операционных систем. Соответственно, PaaS предоставляет меньше управления над вычислительной инфраструктурой в сравнении с IaaS.

На данный момент практика показывает, что PaaS как отдельная модель в мире облачных услуг утратила свою актуальность. На смену независимым PaaS-платформам пришли PaaS-подобные сервисы [6].

Разработка программного продукта требует оптимального распределения ресурсов, при котором обеспечивается их наиболее эффективное использование, что позволяет внедрять инновации, снижать риски и сокращать время выхода продукта на рынок.

Приложение может использовать сотни серверов, и выбор Serverless для проектирования архитектуры приложения позволит избавится от проблем, с которыми сталкиваются при управления серверами. Serverless вовсе не означает отказ от сервера, данная парадигма позволяет разработчикам не беспокоиться о его настройке и администрировании.

В 2014 году Amazon анонсировал AWS Lambda, что позволило Serverless выйти на новый уровень [7]. Данный бессерверный, управляемый событиями вычислительный сервис представил иную тенденцию к минимизации кода и фокусировании на нем. Так появился архитектурный шаблон «функция как услуга» (англ. *Function-as-a-Service, FaaS*), который позволяет сократить необходимую инфраструктуру и инвестиции.

Модель FaaS вводит новый уровень абстракции, в котором исчезает физическая инфраструктура и архитектура программного обеспечения, остаются только «функции», работающие по необходимости. FaaS предоставляет наибольшую гибкость при регулировании производительностью: в период простоя функция не потребляет ресурсы, а при необходимости платформа может быстро выделить необходимую мощность, которая справится с почти любой нагрузкой.

Развитие облачных вычислений

Модель облачных вычислений основывается на традиционном IT-стеке ПО, который состоит из пяти уровней: аппаратное обеспечение, виртуализация, операционная система, среда выполнения, приложение.

Уровень аппаратного обеспечения включает все вычислительные, сетевые и ресурсы хранения. Уровень выше — уровень виртуализации, где происходит виртуализация всех необходимых аппаратных ресурсов с помощью гипервизора для создания виртуальных машин для горизонтального масштабирования. Следующий уровень — операционная система. Здесь располагается программное обеспечение, которое находится на самом деле на удаленном компьютере и позволяет запускать программы. Вдобавок к этому запускается среда выполнения. Она является неотъемлемым связующим звеном между операционной системой и приложением [8]. И также будет средой, где работает приложение. Следующий уровень — уровень приложения — включает в себя всю бизнес-логику. Команды разработчиков проводят на данном уровне большую часть своего времени, потому что именно здесь происходит проектирование, создание и развертывание приложения.

IT-стек в стандартном виде сложен в обслуживании для компании, отказ на одном из уровней может повлиять на любой другой. Но с приходом облачных моделей подход к управлению изменился (рис. 1).

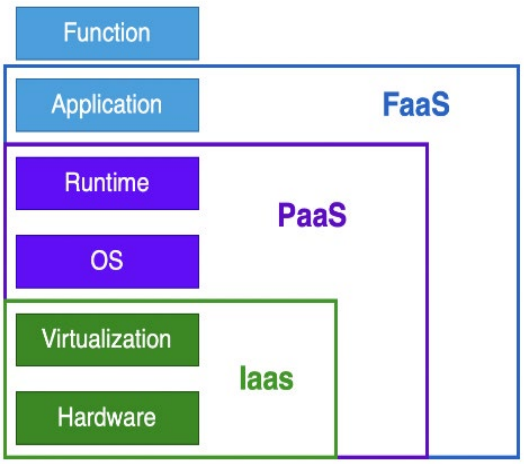

Рис. 1. Развитие облачных моделей

IaaS на самом деле абстрагирует нижние уровни инфраструктуры. Они запускаются от провайдера облачных услуг, что позволяет быстрее подготовить и запустить приложение.

PaaS поднимает уровень абстракции еще дальше. Используя данную модель, разработчики беспокоятся только о своем приложении: это единственное, что нуждается в поддержке. Пропадает необходимость думать о среде выполнения и операционной системе.

FaaS позволяет абстрагировать приложение и позволяет сосредоточиться на функциях. Они представляют собой отдельные компоненты, составляющие это приложение. Такой подход помогает сфокусироваться на разработке без необходимости эксплуатации или управления серверами.

Выбор Serverless FaaS

Одна из особенностей Serverless-архитектуры — отсутствие необходимости общения с серверами напрямую. Поэтому помимо термина «бессерверный» прибегают к его синониму — «бесхостовый» (hostless). Положение, которое исключает централизованную архитектуру, исключает расходы на обслуживание парка серверов. Hostless к тому же означает, что архитектура становится эластичной [9]. Это освобождает от ручного управления ресурсами, поэтому многие проблемы, связанные с распределением ресурсов, исчезают.

Идея, которую представляет модель FaaS, на самом деле проста. Традиционное программирование основано на функциях. Они обеспечивают предсказуемый результат и не меняют ничего в самой программе, что выходит за пределы функции. Сами функции являются «stateless», то есть не сохраняют состояние. Stateless также предполагает, что количество ошибок существенно уменьшается.

Модель FaaS предоставляет клиентам выполнение кода в облаке, а все операции по серверной части берут на себя провайдеры. Клиенты отвечают лишь за свои данные и функции, которые выполняются.

Одной из причин, по которой предпочтение отдают использованию Serverless FaaS, является возможность миновать управление процессами на уровне операционной системы и администрирования серверов. Некоторые сервисы PaaS также предоставляют эту возможность. В действительности концепции PaaS и FaaS похожи, но большинство приложений PaaS не ориентированы на то, чтобы запускать приложения в ответ на события, а платформы FaaS делают именно это.

Ключевое отличие FaaS от PaaS — масштабируемость приложения, основным назначением которого является задача смягчить проблемы, возникающие при сбое или перегрузке приложения. При использовании PaaS, как правило, необходимость думать о том, как масштабировать, все еще присутствует. Так, например, в одной из наиболее используемых облачных PaaS-платформ — Heroku — представлены «dynos» — изолированные виртуальные Linux-контейнеры, которые предназначены для выполнения кода по запросу заданной от пользователя команды. То есть один dyno — один запущенный экземпляр приложения. И сколько dynos необходимо запустить? С FaaS же этот процесс становится прозрачным. Даже если использовать автомасштабирование в PaaS, приложение FaaS все еще эффективнее, когда дело доходит до затрат.

Еще одной абстракцией, которая активно используется в наше время, являются контейнеры приложений. Эта технология обеспечивает приложению среду, отдельную от операционной системы, позволяющую работать независимо от платформы. Системы управления контейнеризованными рабочими нагрузками и сервисами, такие как Mesos и Kubernetes, позволяют также абстрагировать приложения от развертывания на уровне операционной системы [10, 11]. А облачный сервис управления контейнерами Amazon ECS к тому же позволяет еще и избегать управления собственными серверами. Тогда стоит ли останавливать свой выбор на Serverless FaaS? Аргумент, который использовался при сравнении с PaaS, также актуален и для контейнеров: масштабирование все еще прозрачно и автоматически управляемо. А контейнерные платформы все еще требуют управления размером кластеров. Кроме того, FaaS рассматривается как наиболее подходящий выбор для событийно-ориентированной архитектуры, а при выборе контейнеров отдают предпочтение для компонентов, управляемых синхронным запросом с несколькими точками входа.

Сочетание контейнеров и Serverless может быть и полезным. Например, если в случае сложной контейнерной системы есть необходимость выполнить некоторые дополнительные задачи, которые вызваны событиями. Эти задачи будет выгоднее и удобнее перенести от контейнерных настроек к бессерверным функциям [12]. Таким образом, из этого следует интегрирование и передача информации о состоянии между контейнерной и серверной архитектурой.

При разработке программного обеспечения с помощью Serverless сначала определяется логика, необходимая для запуска провайдером, то есть проектируется необходимое количество функций. Шлюз-система API Gateway, которая является общей точкой входа в приложение, принимает и обрабатывает запросы от клиентов к функциям. Затем по требованию эти функции выполняются. Любые данные, которые сгенерированы функциями, возвращаются обратно устройству, которое их вызвало, также возможен вариант выгрузки во внешнее хранилище. Монолитный API заменяется функциями (рис. 2).

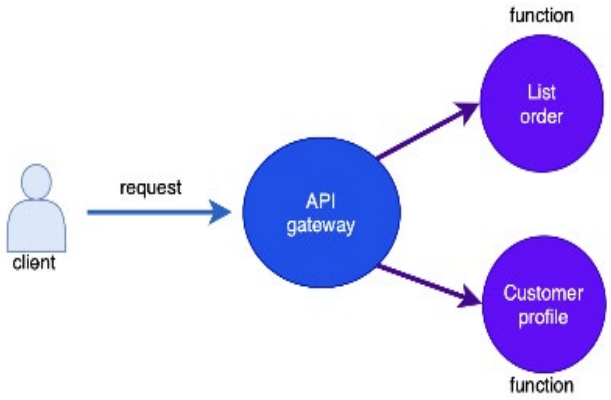

Рис. 2. Схема Serverless-приложения

Serverless-решения хорошо подходят для встраивания в микросервисную архитектуру. В случае небольшой кодовой базы разворачивание отдельного контейнера с операционной системой смысла никакого не имеет, так как это неэкономично с точки зрения использования ресурсов. В таком случае легче использовать функцию. Кроме того, возможно использовать функции как вспомогательные сервисы для тестовой среды. Каждое выполнение функции изолировано и эфемерно, поэтому проектирование приложения с архитектурой Serverless с использованием FaaS требует кроме функции еще и управления данными в хранилищах, механизмов запуска, масштабирования, логирования. В связи с этим построение Serverless не ограничивается

только FaaS. Провайдеры бессерверных вычислений предлагают целые экосистемы состоящие из Serverless-сервисов.

Провайдеры Serverless

Как правило, Serverless-вычисления представляют провайдеры облачных услуг. Бессерверные вычисления способствуют росту бизнеса по той причине, что облегчение программирования приложения на Serverless-архитектуре помогает привлекать новых клиентов, а уже существующим клиентам использовать больше возможностей облачных приложений.

Выбирая подходящий сервис, отвечающий необходимым условиям, нужно учитывать следующие ключевые факторы:

- поддержка языков программирования;
- типы триггера функций;
- безопасность;
- ценообразование и факторы биллинга;
- время выполнения и параллелизм;
- метод развертывания.

Serverless развивается, и вместе с ним на рынке появляется все больше возможностей и сервисов, которые предлагают провайдеры.

AWS Lambda

Компания Amazon была пионером в развитии Serverless-технологий. На текущий момент в рамках Amazon Web Services корпорация предлагает сервис для бессерверных вычислений AWS Lambda.

Суть Lambda заключается в выполнении кода при реагировании на различные события. Иными словами, единственным, что требуется в использовании данного сервиса, является загрузка кода, который должен исполняться в функции, и настройка триггера или события, которое отвечает за выполнение кода. Всю остальную ответственность за администрирование платформы AWS Lambda полностью берет на себя. Таким образом, пропадает необходимость в обслуживании серверов, управлении операционной системой, выделении нужных ресурсов, масштабировании и мониторинга кода. В AWS Lambda от пользователя скрыта внутренняя реализация модели и почти все внутренние процессы, поэтому теряется актуальность безопасности данной платформы.

Список сервисов AWS для бессерверных вычислений широк. Вместе с Lambda активно используется Amazon API Gateway, с помощью которого можно значительно сократить расходы. Изменение данных происходит в корзинах Amazon S3 и базах данных Amazon DynamoDB. С сервисом для оркестрации бессерверных вычислений AWS Step Functions обычно проектируется и выполняется процесс, который может координировать несколько функций AWS Lambda. Также, используя комплект средств разработки AWS SDK, есть возможность запускать код через вызовы API. Кроме того, Lambda охватывает широкий спектр языков программирования.

Google Cloud Functions

Следующим по распространенности поставщиком услуг FaaS в мире является Google Cloud Functions. Данный сервис также позволяет абстрагироваться от всей базовой инфраструктуры, что позволяет сосредоточиться на коде.

Google Cloud Functions отлично интегрирована с сервисами Google. Сервис обеспечивает быстрый переход от кода к развертыванию, который включает интегрированный мониторинг. Полный обзор и диагностику приложения позволяет получить сервис Google Cloud Trace, предназначенный для оптимизации работы в облаке, и сервис Google Cloud Debugger, который проверяет состояние запущенного приложения в режиме реального времени. Кроме того, Cloud Functions позволяет использовать сторонние облачные сервисы, которые являются строительными блоками для кода.

Ценообразование основывается на количестве вызовов функций. Первые два миллиона вызовов бесплатны, каждый последующий миллион вызовов — 0,40 долларов США. Однако Google взимает дополнительную плату за использование памяти и ЦП.

Присутствует автомасштабирование: Cloud Functions автоматически управляют и масштабируют необходимую инфраструктуру в зависимости от рабочей нагрузки.

Microsoft Azure Cloud Functions

На текущий момент компания Microsoft является лидером среди поставщиков облачных услуг [13] и представляет платформу для бессерверных вычислений Azure Cloud Functions.

Время на написание кода значительно сокращается при использовании триггеров и привязок, что позволяет приложениям реагировать на события и без особых проблем подключаться к другим службам. Функции Azure предоставляют создание более масштабируемых и стабильных событийно-ориентированных приложений.

Платформа Azure для бессерверных вычислений позволяет создавать надежные приложения со встроенными средствами защиты и мониторинга. Сервис Azure Application Insights предоставляет средства для отслеживания и анализа производительности кода. Контроль политики доступа происходит в безопасном хранилище секретов Azure Key Vault, где можно хранить исходные параметры приложения, при этом оставляя код без изменений.

Основное преимущество Azure Functions — применение любого стека технологий. Использование сервиса не предполагает ограничения на выбор какого-то конкретного языка или сервера. Что дает возможность выбрать оптимальный язык программирования для каждого сценария. В случае использования языка, не имеющего встроенной поддержки, Azure Cloud Functions предлагают особые стратегии для создания функций [14].

Метод ценообразования в Azure такой же, как и в Lambda, и также предоставляются бесплатные пакеты на первые вызовы функций, далее взимается 0,000016 долларов США за гигабайтосекунду. Стоимость при высокой нагрузке у Azure немного ниже, если сравнивать с Lambda. Кроме того, Microsoft также взимает плату за использование памяти.

Yandex Cloud Functions

Отечественная IT-компания также не отстает от зарубежных и активно развивает свою Serverless-экосистему на платформе Yandex.Cloud. Она берет на себя обслуживание и все необходимые условия для непрерывной работы. Yandex.Cloud предоставляет отказоустойчивое и автомасштабируемое окружение для всей Serverless-экосистемы.

В нее входит сервис Yandex Cloud Functions, который позволяет запускать код приложения в виде функции.

В сервисе присутствуют триггеры для запуска функции; с помощью данной возможности интеграция между Cloud Functions и другими сервисами происходит легко и без использования интеграционного кода. К поддерживаемым языкам программирования на текущий момент относятся: Node.js, Python, Go, PHP, Bash, Java, C# и R [15].

Автомасштабирование позволяет при увеличении нагрузки на количество вызовов создавать дополнительные экземпляры функции. Функции, которые запущены, имеют возможность выполняться параллельно.

В сервисе Cloud Functions действуют удобные тарифы: каждый месяц использования в тариф не входят один миллион вызовов функций и 10 ГБ/ч выполнения функций. Далее за 1 ГБ/ч взимается плата 3,42 руб., стоимость вызова 1 млн функций — 10 руб.

В Serverless-экосистему Yandex.Cloud помимо Yandex Cloud Functions также входят следующие сервисы:

1. Yandex API Gateway. Сервис управления API-шлюзами, который передает запросы от приложений к сервисам Yandex.Cloud и сокращает задержку при обработке запросов к API.

2. Yandex Database (YDB). База данных имеет поддержку Serverless-режима и предоставляет автоматизированное масштабирование выполнения запросов, хранения и резервного копирования. Кроме того, YDB совместима с Amazon DynamoDB.

3. Сервис очереди сообщений Message Queue. При масштабировании системы использование сервиса очередей является неотъемлемой частью обмена сообщениями между компонентами. Очередь сообщений позволяет освобождать ресурсы для новых запросов при выполнении длительных задач.

Таким образом, использование Serverless-сервисов Yandex.Cloud позволяет создавать надежные проекты с простым управлением. В бессерверной инфраструктуре обработка запросов происходит посредством API Gateway, далее они перенаправляются в Cloud Functions. Хранение данных в Yandex Database автоматически масштабируется при росте интенсивности запросов.

Сравнительная характеристика вышеупомянутых провайдеров услуг представлена на рисунке 3.

	AWS Lambda	Google Cloud Functions	Azure Cloud Functions	Yandex Cloud Functions
Стоимость	Первые 1M в месяц бесплатно Далее $0.2/1M	$0.4/M	Первые 1M в месяц бесплатно Далее - $0.16/1M	Первые 1M в месяц бесплатно Далее - ₽10/1M
Встроенная поддержка языка	Java, Go, PowerShell, Node.js, C#, Python, Ruby	Node.js, Python, Go, Java, .NET, Ruby, PHP	Node.js, C #, F #, Python, PHP, Bash, Batch, PowerShell	Node.js, Python, Go, PHP, Bash, Java, C#, R
Интеграция со сторонними сервисами	X	✓	✓	✓
Выполнение функции	1000 выполнений одновременно	1000 выполнений одновременно Для HTTP триггеров не ограничено	Не ограничено Max время выполнения - 5 мин	10 в каждой зоне доступности
Параллелизм	Для функции	Для приложения Для функции	Для функции	Для функции

Рис. 3. Сравнение провайдеров услуг

При выборе провайдера важно помнить и о рисках Serverless, которые связаны с Vendor Lock, привязкой к поставщику. Функции, созданные с помощью AWS, сложно импортируются в другие платформы, например в Google Cloud. Помимо функций обычно используются базы данных, сервисы очередей, логирования и прочие Serverless-сервисы, которые у каждого провайдера могут быть свои. Например, Yandex Cloud Functions предусматривает совместимость с AWS [16].

Преимущества Serverless

Основные потенциальные преимущества бессерверных вычислений заключаются в следующем:

1. Абстракция от внешней инфраструктуры. Несомненным и главным преимуществом Serverless является абстракция от рабочих процессов, связанных с администрированием серверов и управлением операционной системой.

2. Низкие эксплуатационные расходы. Поддержка меньшего количества компонентов означает меньшее ко-

личество затраченной работы. Serverless-приложения выпускаются быстрее, а значит, время для получения обратной связи от конечного пользователя также уменьшается, такая возможность влечет улучшения в будущем в течение всей разработки программного обеспечения.

3. Максимальная эластичность. Одна из сильных сторон Serverless — масштабирование. Оно происходит без внесений изменений в уже написанный код, и при росте трафика количество запущенных функций также возрастает.

4. Эффективная стоимость. Оплата происходит только за используемые вычислительные ресурсы, то есть не взимается, когда приложение простаивает.

5. Балансировка нагрузки. В случае высокой посещаемости сервиса возрастает нагрузка на железо. В традиционной архитектуре балансировщик распределяет нагрузку на виртуальные машины. Несомненно, что внедрение Serverless окажет положительный эффект, то есть функции, запросы к которым приходят от балансировщика через API Gateway, будут брать на себя ответственность в обработке части запросов. Таким образом, система оптимизируется, распределяя нагрузку (рис. 4). В таком случае запуск функций обходится выгоднее, чем все время работающие виртуальные машины.

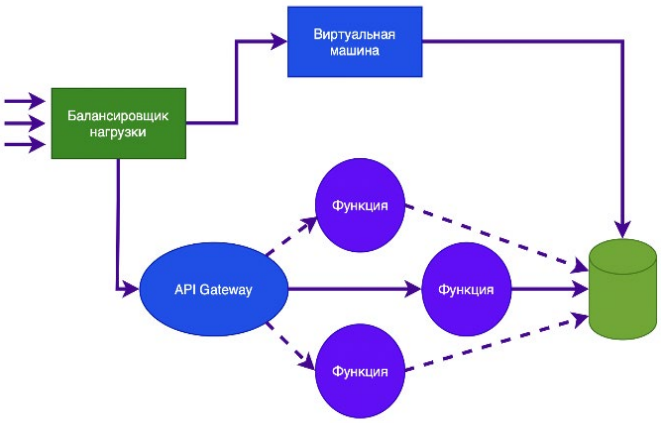

Рис. 4. Архитектура с Serverless-подходом и балансировщиком нагрузки

6. Встроенные интеграции. Большинство представленных на рынке провайдеров Serverless предлагают интеграцию с другими сервисами. Это преимущество благоприятно сказывается на создании программного обеспечения, так как основное внимание уделяется разработке, а не настройке и управлению.

Недостатки Serverless

Как и любая технология, Serverless обладает рядом недостатков. Некоторые из них неподвластны прогрессу и всегда будут актуальны, другие же связаны с текущей реализацией и в скором времени найдут решение, которое позволит технологии быть лучше.

1. Контроль за поставщиками. Прибегая к услугам провайдера, он берет на себя контроль над частями вашей системы. Отсутствие контроля может выражаться в простое, неожиданном изменении стоимости услуг, потере функциональности, принудительном обновлении API и прочим проблемам. Кроме того, определенные способы развертывания, мониторинга, создания Serverless-функций у разных провайдеров разные. И в данном случае не стоит забывать об обратной совместимости: от любого интерфейса может зависеть другой сервис или функция.

2. Время холодного запуска. Время холодного старта при запуске бессерверных вычислений появляется вследствие того, что при срабатывании триггера сначала происходит загрузка среды выполнения, а лишь потом запуск функции (рис. 5).

Рис. 5. Схема запуска функции при холодном старте

Управлять средой выполнения у разработчика нет возможности, в итоге жизненный цикл среды может быть непредсказуем. Помимо этого, на холодный запуск влияет много зависящих друг от друга факторов: язык программирования, объем написанного кода, встроенные дополнительные ресурсы, такие как базы данных. Разработчик может этим управлять, сократив время старта, но доступ к управлению контейнером есть только у провайдера.

Холодный старт может стать теплым в случае, если количество обращений на функцию растет, или если разработчик запускает функции, следуя времени по таймеру.

3. Serverless не подходит для долгосрочных процессов. При проектировании приложения нужно учитывать, что если в архитектуре присутствует функция, которая используется приложением постоянно, то и потребление ресурсов не будет отличаться от традиционного подхода. Поэтому накладывать ответственность на функцию нужно разумно, тщательно обдумав ее дальнейшее использование.

Заключение

На основании вышесказанного логично предположить, что облачные вычисления продолжают набирать свою популярность, как и бессерверные технологии, которые предоставляют упрощенную среду для программирования, что намного облегчает использование облака. Несомненно, это способствует привлечению людей, заинтересованных в его использовании.

Неоспоримое преимущество Serverless заключается в фокусировании на разработке кода и избавлении от необходимости управления ресурсами и их оптимизации вручную. Такой подход схож с переходом от языка ассемблера на высокоуровневые языки. При этом следует отметить и повышенную сложность разработки и тестирования приложений по причине того, что бессерверная архитектура требует структурного разделения одного приложения на сервисы и функции, а это в свою очередь значительно увеличивает время проектирования архитектуры [17].

Очевидной ценностью Serverless является сокращение объема инвестиций, необходимых для поддержки работы приложений, что позволяет получить больше пространства для экспериментов и инноваций.

Несмотря на уже достигнутый успех в Serverless вычислениях, они продолжают свою эволюцию, поставщики услуг упорно работают над развитием новых услуг и улучшением уже существующих, устраняя выявленные недостатки. Разработчики уже приступили к изучению вопроса

о дальнейшем процессе развития бессерверных платформ и повышения эффективности рабочей нагрузки [18]. Решаются проблемы в областях абстракции, систем, сетей, безопасности и архитектуры. Нетрудно спрогнозировать, что дальнейший рост и изменение бессерверности будет и идти дальше. Так через десятки лет Serverless станет стандартом среди облачных технологий, заменив серверные вычисления.

Литература

1. Occupy the Cloud: Distributed Computing for the 99% / E. Jonas, Q. Pu, S. Venkataraman, [et al.] // Proceedings of the Eighth ACM Symposium on Cloud Computing (SoCC`17) (Santa Clara, CA, USA, 24–27 September 2017). — New York (NY): Association for Computing Machinery, 2017. — Pp. 445–451. DOI: 10.1145/3127479.3128601.

2. Переход в облако стал повседневной задачей для бизнеса // C-News. — 2018. — 31 октября. URL: http://www.cnews.ru/articles/2018-10-31_perehod_v_oblako_stal_povsednevnoj_zadachej_dlya_srednego_biznesa (дата обращения 07.03.2022).

3. Тренды развития облачных вычислений // TAdviser — Государство. Бизнес. Технологии. — 2022. — 24 января. URL: http://www.tadviser.ru/a/147459 (дата обращения 07.03.2022).

4. Что такое IaaS-облако, сравнение с PaaS и SaaS на примерах // 1cloud. URL: http://1cloud.ru/services/private-cloud/iaas-paas-saas (дата обращения 08.03.2022).

5. Desde IaaS, CaaS, PaaS y FaaS, ¿Cómo Elegir la Plataforma Adecuada? / Traducido por Fu Hui // Programador Clic. URL: http://programmerclick.com/article/80131207853 (дата обращения 09.03.2022).

6. Стельмах, С. Облачные вычисления: пик популярности пройден? // itWeek. — 2020. — 10 февраля. URL: http://www.itweek.ru/its/article/detail.php?ID=211336 (дата обращения 09.03.2022).

7. Avram, A. FaaS, PaaS, and the Benefits of the Serverless Architecture // InfoQ. — 2016. — 25 June. URL: http://www.infoq.com/news/2016/06/faas-serverless-architecture (дата обращения 09.03.2022).

8. Delimitrou, C. Quasar: Resource-Efficient and QoS-Aware Cluster Management / C. Delimitrou, C. Kozyrakis // Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIX) (Salt Lake City, UT, USA, 01–05 March 2014). ACM SIGPLAN Notices. 2014. Vol. 49, Is. 4. Pp. 127–144. DOI: 10.1145/2644865.2541941.

9. Peeking Behind the Curtains of Serverless Platforms / L. Wang, M. Li, Y. Zhang, [et al.] // Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC`18) (Boston, MA, USA, 11–13 July, 2018). — Washington (D.C.): USENIX Association, 2018. — Pp. 133–145.

10. Chandra, B. Mesos — Creating Your Own Serverless // Medium. — 2019. — 17 July. URL: http://bhanuandchandra.medium.com/mesos-creating-your-own-serverless-2aaf65cfb0b7 (дата обращения: 10.03.2022).

11. Brewer, E. A. Kubernetes and the Path to Cloud Native // Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC`15) (Kohala Coast, HI, USA, 27–29 August 2015). — New York (NY): Association for Computing Machinery, 2015. — P. 167. DOI: 10.1145/2806777.2809955.

12. Serverless vs Containers: Choose One or Use Both? // Cloud Native Wiki. URL: http://www.aquasec.com/cloud-native-academy/serverless-architecture/serverless-vs-containers-choose-one-or-use-both (дата обращения 10.03.2022).

13. Cloud Wars Top 10 — The World's Top Cloud Vendors // Acceleration Economy. URL: http://acceleration-economy.com/cloud-wars-top-10 (дата обращения 15.03.2022).

14. Supported Languages in Azure Functions // Microsoft Docs. — 2021. — 21 October. URL: http://docs.microsoft.com/en-gb/azure/azure-functions/supported-languages (дата обращения 15.03.2022).

15. Среда выполнения. Обзор // Yandex.Cloud. URL: http://cloud.yandex.ru/docs/functions/concepts/runtime (дата обращения 12.03.2022).

16. Serverless — бессерверные вычисления // Yandex.Cloud. URL: http://cloud.yandex.ru/solutions/serverless (дата обращения 12.03.2022).

17. Hennesy, J. L. A New Golden Age for Computer Architecture / J. L. Hennessy, D. A. Patterson // Communications of the ACM. 2019. Vol. 62, Is. 2. Pp. 48–60. DOI: 10.1145/3282307.

18. Serverless Computing: One Step Forward, Two Steps Back / J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, [et al.] // Proceedings of the 9th Biennial Conference on Innovative Data Systems Research (CIDR 2019) (Asilomar, CA, USA, 13–16 January 2019). 9 p. URL: http://cidrdb.org/cidr2019/papers/p119-hellerstein-cidr19.pdf (дата обращения 12.03.2022).

DOI: 10.24412/2413-2527-2022-129-16-23

Characteristics of Serverless Computing Technologies

A. T. Gordina, PhD A. V. Zabrodin
Emperor Alexander I St. Petersburg State Transport University
Saint Petersburg, Russia
anngordtm@gmail.com, teach-case@yandex.ru

Abstract. **This article is devoted to one of the prospective types of cloud computing known as Serverless computing. The paper considers the technologies used for Serverless, gives comparison of cloud models and service providers, also it describes main components of FaaS architectural pattern (functions, resources and events). The authors consider the advantages and disadvantages of Serverless computing. Besides that, the article discloses problems of application migration from local servers to the cloud and how resource management and optimization are excluded from the software development process.**

Keywords: **cloud computing, Serverless, FaaS model, functions, autoscaling, application architecture.**

REFERENCES

1. Jonas E., Pu Q., Venkataraman S., et al. Occupy the Cloud: Distributed Computing for the 99%, *Proceedings of the Eighth ACM Symposium on Cloud Computing (SoCC`17), Santa Clara, CA, USA, September 24–27, 2017*. New York (NY), Association for Computing Machinery, 2017, Pp. 445–451. DOI: 10.1145/3127479.3128601.

2. Migration to Cloud Has Become Routine Task for Business [Perekhod v oblako stal povsednevnoy zadachey dlya biznesa], *C-News*. Published online at October 31, 2018. Available at: http://www.cnews.ru/articles/2018-10-31_perehod_v_oblako_stal_povsednevnoj_zadachej_dlya_srednego_biznesa (accessed 07 Mar 2022).

3. Development Trends of Cloud Computing [Trendy razvitiya oblachnykh vychisleniy], *T-Adviser — Government. Business. Technology [TAdviser — Gosudarstvo. Biznes. Tekhnologii]*. Published online at January 24, 2022. Available at: http://www.tadviser.ru/a/147459 (accessed 07 Mar 2022).

4. What is IaaS Cloud, Comparison with PaaS and SaaS on Examples [Chto takoe IaaS-oblako, sravnenie s PaaS i SaaS na primerakh], *1cloud*. Available at: http://1cloud.ru/services/private-cloud/iaas-paas-saas (accessed 08 Mar 2022).

5. From IaaS, CaaS, PaaS and FaaS, How to Choose the Right Silver-Form, *Programmer click*. Available at: http://programmerclick.com/article/80131207853 (in Spanish) (accessed 09 Mar 2022).

6. Stelmakh S. Cloud Computing: Boom Has Gone? [Oblachnye vychisleniya: pik populyarnosti proyden?], *itWeek*. Published online at February 10, 2020. Available at: http://www.itweek.ru/its/article/detail.php?ID=211336 (accessed 09 Mar 2022).

7. Avram A. FaaS, PaaS, and the Benefits of the Serverless Architecture, *InfoQ*. Published online at June 25, 2019. Available at:http://www.infoq.com/news/2016/06/faas-serverless-architecture (accessed 09 Mar 2022).

8. Delimitrou C., Kozyrakis C. Quasar: Resource-Efficient and QoS-Aware Cluster Management, *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIX), Salt Lake City, UT, USA, March 01–05, 2014. ACM SIGPLAN Notices*, 2014, Vol. 49, Is. 4, Pp. 127–144. DOI: 10.1145/2644865.2541941.

9. Wang L., Li M., Zhang Y., et al. Peeking Behind the Curtains of Serverless Platforms, *Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC`18), Boston, MA, USA, July 11–13, 2018*. Washington (D.C.), USENIX Association, 2018, Pp. 133–145.

10. Chandra B. Mesos — Creating Your Own Serverless, *Medium*. Published online at July 17, 2019. Available at: http://bhanuandchandra.medium.com/mesos-creating-your-own-serverless-2aaf65cfb0b7 (accessed 10 Mar 2022).

11. Brewer E. A. Kubernetes and the Path to Cloud Native, Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC`15), Kohala Coast, HI, USA, August 27–29, 2015. New York (NY), Association for Computing Machinery, 2015, P. 167. DOI: 10.1145/2806777.2809955.

12. Serverless vs Containers: Choose One or Use Both, *Cloud Native Wiki*. Available at: http://www.aquasec.com/cloud-native-academy/serverless-architecture/serverless-vs-containers-choose-one-or-use-both (accessed 10 Mar 2022).

13. Cloud Wars Top 10 — The World's Top Cloud Vendors, *Acceleration Economy*. Available at: http://accelerationeconomy.com/cloud-wars-top-10 (accessed 15 Mar 2022).

14. Supported Languages in Azure Functions, *Microsoft Docs*. Published online at October 21, 2021. Available at: http://docs.microsoft.com/en-gb/azure/azure-functions/supported-languages (accessed 15 Mar 2022).

15. Runtime. Overview [Sreda vypolneniya. Obzor], *Yandex.Cloud*. Available at: http://cloud.yandex.ru/docs/functions/concepts/runtime (accessed 12 Mar 2022).

16. Serverless Computing [Serverless — besservernye vychisleniya], *Yandex.Cloud*. Available at: http://cloud.yandex.ru/solutions/serverless (accessed 12 Mar 2022).

17. Hennesy J. L., Patterson D. A. A New Golden Age for Computer Architecture, *Communications of the ACM*, 2019, Vol. 62, Is. 2, Pp. 48–60. DOI: 10.1145/3282307.

18. Hellerstein J. M., Faleiro J., Gonzales J. E., et al. Serverless Computing: One Step Forward, Two Steps Back, *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research (CIDR 2019), Asilomar, CA, USA, January 13–16, 2019*. 9 p. URL: http://cidrdb.org/cidr2019/papers/p119-hellerstein-cidr19.pdf (accessed 12 Mar 2022).