SERVERLESS ФУНКЦИИ И ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ.

АНАЛИТИЧЕСКИЙ ОБЗОР

Выполнил:

Величко Арсений Александрович

_____

(подпись исполнителя)

Санкт-Петербург

2025

ОГЛАВЛЕНИЕ

Введение

В последнее десятилетие в сфере облачных вычислений получил широкое распространение подход бессерверных функций (serverless computing). Данная модель облачных сервисов позволяет выполнять программный код без необходимости самостоятельно управлять серверной инфраструктурой [1]. Облачный провайдер берёт на себя запуск этого кода, автоматическое масштабирование и обслуживание серверов, а оплата взимается только за фактическое время работы функций (время простоя при этом не тарифицируется) [2]. Такой подход считается экономически эффективным: согласно совместному исследованию Yandex Cloud и Ipsos, 43% опрошенных специалистов отметили, что serverless-подход выгоден, главным образом за счёт сокращения времени на поддержку инфраструктуры (отметили 49% респондентов) и быстрого масштабирования (46%) [2]. В результате бессерверная архитектура стала одним из актуальных трендов ИТ-индустрии и продолжает активно развиваться [2].

Архитектурные особенности бессерверной архитектуры

В основе serverless-архитектуры лежит модель Function as a Service (FaaS), предполагающая разбиение приложения на независимые функции, которые запускаются облачным провайдером по наступлении определённых событий и выполняются в изолированном окружении [3]. В отличие от традиционных монолитных приложений (или даже микросервисов), такие функции являются кратковременными и не сохраняют своего состояния между вызовами [2]. Это означает, что при каждом новом вызове функция запускается как отдельный инстанс; все данные, требующие долговременного хранения, сохраняются во внешних облачных сервисах (база данных, файловое хранилище и т. д.), которые интегрируются в общую экосистему решения [2]. Разработчик при деплое предоставляет лишь исходный код функции и указывает привязанные к ней триггеры событий, тогда как развертывание и исполнение этой функции

полностью управляется провайдером и скрывает от пользователя детали серверной инфраструктуры [3]. Например, сервис AWS Lambda автоматически выполняет загруженную в него функцию при поступлении HTTP-запроса или добавлении файла в хранилище, при этом все необходимые серверные ресурсы выделяются платформой динамически. Таким образом, бессерверная архитектура предоставляет высокий уровень абстракции: разработчики могут сосредоточиться на написании функциональной логики, а масштабирование и управление средой исполнения происходят “за кулисами” облачной платформы.

Вопросы безопасности

В бессерверной модели ответственность за безопасность базовой инфраструктуры в значительной мере лежит на облачном провайдере. Сервисы функций реализуют строгую изоляцию исполнения кода разных клиентов за счёт технологий виртуализации: например, в платформе Yandex Cloud каждая функция запускается в отдельной лёгкой виртуальной машине, без разделяемого между пользователями ядра операционной системы [4]. Это предотвращает возможное взаимное влияние или несанкционированный доступ между функциями разных арендаторов облака. Крупные облачные провайдеры также проходят регулярные аудиты и сертификации по отраслевым стандартам информационной безопасности (включая нормативы РФ и международные стандарты типа ISO 27017/27018, PCI DSS и др.), что подтверждает высокий уровень защищённости их инфраструктуры [4].

При этом разработчик приложения отвечает за безопасность своего кода и правильную настройку доступов. Поскольку логика размещена в функциях, взаимодействующих с другими компонентами через чётко определённые API, устраняется ряд рисков, связанных с неправильной конфигурацией серверов (как в традиционных системах). Однако появляются и новые задачи: необходимо безопасно хранить и передавать секретные данные (ключи API, токены) в среде функций, настраивать политики доступа (IAM) для каждой

функции и обеспечивать мониторинг большого числа распределённых компонентов. В целом, подходы и лучшие практики для обеспечения безопасности в serverless остаются сходными с другими облачными моделями: используется шифрование данных, изоляция сетевых взаимодействий, управление правами на основе ролей, регулярное обновление зависимостей и т.д. При соблюдении этих мер и благодаря тому, что инфраструктура поддерживается профессиональным провайдером, уровень безопасности serverless-решений может быть не ниже, а зачастую выше, чем у развёртываемых собственными силами традиционных серверных приложений.

Масштабируемость и производительность

Автоматическое масштабирование – одно из ключевых достоинств бессерверной архитектуры. Платформа сама обеспечивает запуск дополнительных экземпляров функции при росте входящего трафика и освобождение ресурсов, когда нагрузка снижается [2]. Разработчикам не нужно заранее предусматривать политики масштабирования или вручную поддерживать резерв мощностей – инфраструктура гибко подстраивается под текущий поток запросов в режиме реального времени [1]. Более того, если функция не используется, ресурсы под неё не держатся вовсе (так называемое масштабирование до нуля), что устраняет затраты на простаивающие мощности.

Однако такая динамическая модель накладывает особенности на производительность приложений. После периода бездействия функция выгружается из памяти, поэтому первый вызов функции после долгого перерыва сопровождается задержкой на инициализацию окружения – так называемым «холодным стартом» [1]. Длительность этой дополнительной задержки зависит от множества факторов (язык и размер кода функции, внутренняя реализация провайдера и пр.) и в некоторых случаях может достигать нескольких секунд [2]. После прогрева функции последующие

вызовы обрабатываются значительно быстрее (режим warm start), но если функция вновь не востребована продолжительное время, она опять выгружается, и следующий запрос пользователя снова испытает эффект холодного запуска [1]. Некоторые провайдеры предлагают механизмы смягчения данной проблемы, позволяющие держать часть инстансов функции постоянно активными (например, зарезервированные контейнеры или поток выполнения) [2], однако это частично снижает экономические преимущества модели pay-as-you-go.

В целом serverless-архитектура отлично справляется со всплесками и нерегулярной нагрузкой, мгновенно распространяя вычисления на нужное число параллельных экземпляров функций. Но для задач, требующих постоянной, непрерывной работы или предельно низкой латентности отклика, данная модель может оказаться менее подходящей из-за описанных ограничений (задержки холодного старта, жёсткие лимиты времени исполнения и т. д.).

Управление и эксплуатация

Бессерверные платформы значительно упрощают эксплуатацию приложений по сравнению с традиционными моделями развёртывания. Многие типовые задачи администрирования и DevOps автоматизированы на стороне провайдера: непрерывная интеграция и доставка (CI/CD), сбор и анализ логов, мониторинг, поддержание работоспособности среды выполнения – всё это берёт на себя облачная платформа [2]. Разработчикам не требуется самостоятельно настраивать виртуальные машины, устанавливать обновления ОС или управлять масштабированием вручную, что существенно ускоряет цикл разработки и развертывания новых версий продукта [1]. Небольшие изолированные функции проще тестировать и обновлять по отдельности, благодаря чему новые функции и исправления могут внедряться быстрее, без длительных простоев на деплой всего приложения.

Наряду с преимуществами, serverless-подход предъявляет и некоторые новые требования. К примеру, отладка распределённой системы, состоящей из множества взаимосвязанных функций, может быть сложнее: для отслеживания выполнения придётся использовать распределённый трейсинг запросов, централизованный логгер и другие инструменты мониторинга. Возрастает зависимость от экосистемы конкретного облачного провайдера: перенос serverless-приложения в другую облачную среду может потребовать переработки, поскольку используются специфичные сервисы и конфигурации одного поставщика. Тем не менее, эти сложности обычно преодолимы за счёт грамотного архитектурного планирования и использования технологий инфраструктуры как кода. В результате, управление бессерверными функциями в целом требует меньше трудозатрат, позволяя ИТ-командам сосредоточиться на логике приложения, а не на поддержке инфраструктуры [1].

Платформы бессерверных функций

Впервые модель FaaS была коммерчески реализована компанией Amazon в сервисе AWS Lambda, запущенном в 2014 году [5]. Этот сервис стал фактически первым в мире инструментом бессерверных облачных вычислений, показав на практике жизнеспособность нового подхода. Вслед за Amazon собственные платформы функций представили и другие ведущие провайдеры: так появились Microsoft Azure Functions, Google Cloud Functions и ряд других. Все эти решения схожи по основным принципам: разработчик загружает код функции и определяет события-триггеры (HTTP-запрос, появление сообщения в очереди, изменение файла в хранилище и т. д.), после чего провайдер автоматически выполняет эту функцию в облаке при каждом возникновении указанного события.

В России также появились аналогичные облачные сервисы. В настоящий момент бессерверные функции доступны у трёх крупных отечественных

провайдеров: в Яндекс Облаке (сервис Yandex Cloud Functions), в облачной платформе Сбербанка (SberCloud) и у компании Selectel [6]. Эти решения позволяют выполнять пользовательский код по модели FaaS внутри страны, что помимо прочего упрощает вопросы расчётов (оплата в рублях, отсутствие требований к зарубежным платёжным средствам и т.д.). Помимо предложений публичных облаков, существуют и открытые реализации serverless-платформ для частных инфраструктур (такие как OpenFaaS, Apache OpenWhisk, Fission и др.), которые организации могут развернуть на собственных серверах. Однако массовое распространение данной модели связано прежде всего с услугами крупных облачных провайдеров, поэтому именно их возможности и ограничения наиболее интересны в прикладном плане.

Каждая платформа функций имеет определённые технические лимиты и особенности, которые постепенно эволюционируют. Например, Yandex Cloud Functions в текущей версии допускает выполнение одной функции продолжительностью до 60 минут и использование до 8 ГБ оперативной памяти на экземпляр [7]. У AWS Lambda исторически ограничения были строже (в ранние годы – до нескольких минут времени и около 3 ГБ памяти на функцию), однако со временем платформа расширила эти границы. В любом случае, общая концепция остаётся единой: детали масштабирования и управления ресурсами скрыты от разработчика, а сами функции тесно интегрируются с остальными сервисами своего облачного провайдера (базами данных, очередями сообщений, шлюзами API и т.д.), которые совместно образуют полноценную платформу для построения приложений.

Сравнение с традиционными моделями IaaS и PaaS

Модель бессерверных функций существенно отличается от традиционных подходов предоставления облачных ресурсов типа IaaS (Infrastructure as a Service) и PaaS (Platform as a Service). В случае IaaS облачный провайдер предоставляет клиенту виртуальные машины (инфраструктуру), и всё

последующее администрирование системы – установка и настройка ОС, middleware, масштабирование, обновления – ложится на команду клиента. Этот подход даёт максимальную гибкость (можно запускать любые необходимые сервисы и программные стеки), однако требует значительных усилий по поддержке и ресурсов даже в периоды простоя. Serverless же радикально упрощает жизнь разработчиков: им не нужно управлять серверами, и ресурсы фактически арендуются только на время выполнения кода [2]. При переменных или непредсказуемых нагрузках это позволяет избежать затрат на неиспользуемую мощность. В то же время за удобство оплаты по запросу приходится платить повышенной стоимостью вычислительной секунды: есть оценки, что эквивалентный объём ресурсов в функции может стоить значительно дороже (вплоть до порядка 30-кратного превышения) по сравнению с арендованной виртуальной машиной при непрерывной загрузке [7]. Поэтому для постоянных, фоновых задач традиционный выделенный сервер (или VM в модели IaaS) зачастую оказывается экономически выгоднее, тогда как serverless-модель выигрывает при эпизодических всплесках и непостоянном трафике.

Если сравнивать с PaaS, то бессерверный подход обеспечивает более гибкое масштабирование и распределение ресурсов. Платформа как услуга обычно предоставляет заранее настроенное окружение для приложения (например, контейнер с веб-сервером) и может поддерживать автоматическое масштабирование, но, как правило, не масштабируется до нуля и имеет заметные задержки при запуске новых экземпляров приложения [1]. Кроме того, PaaS-сервисы часто подразумевают постоянное резервирование некоторых ресурсов (даже при низкой нагрузке), тогда как при FaaS-архитектуре ресурсы выделяются строго под событие, исключая длительные простои без дела [2]. С другой стороны, приложения на PaaS обычно могут сохранять состояние в памяти между запросами и выполнять фоновые задачи, тогда как FaaS-функции полностью stateless и любое состояние требует внешнего хранилища, что

усложняет архитектуру. Также переход от монолитного приложения (или даже от крупнозернистой микросервисной архитектуры) к набору функций требует более тщательной декомпозиции логики и управления большим числом компонентов.

Таким образом, бессерверная архитектура предоставляет высшую степень абстракции вычислительных ресурсов. По сравнению с IaaS она существенно снижает административные усилия (не требуется управлять VM) и автоматически масштабируется под нагрузку, а по сравнению с PaaS – более эффективно использует ресурсы, избегая оплачиваемых простоев. Вместе с тем эти преимущества достигаются ценой отказа от полного контроля над средой выполнения, появления дополнительных задержек (cold start) и ограничений по времени и объёму ресурсов, поэтому выбор в пользу serverless-модели должен основываться на анализе конкретных требований приложения и характера нагрузки.

Заключение

Бессерверные технологии функций как сервисов заметно изменили подход к разработке и развёртыванию облачных приложений. Они дополняют традиционные модели IaaS/PaaS, позволяя быстро создавать масштабируемые системы с минимальными операционными издержками. На практике оптимальная архитектура часто включает сочетание разных моделей: например, постоянная база данных может размещаться на выделенной инфраструктуре (DBaaS или IaaS), а логика обработки событий – реализована через FaaS. При грамотном применении serverless-архитектура обеспечивает высокую гибкость и устойчивость к нагрузкам, однако выбирать данный подход следует с учётом его технических ограничений (время холодного старта, stateless-характер функций и др.) и специфики приложения. Рост числа сервисов функций у мировых и российских облачных провайдеров в последние годы подтверждает

востребованность serverless-подхода для решения широкого круга задач современного программирования в облаке.

Список литературы

1. Что такое serverless computing (бессерверные вычисления)? / Хабр. – Москва, [2021]. – URL: https://habr.com/ru/companies/cloud4y/articles/541422/ (дата обращения: 08.04.2025). – Режим доступа: свободный.
2. Serverless-архитектура или бессерверные вычисления: обзор. – Москва, [2024]. – URL: https://timeweb.cloud/blog/serverless-architecture (дата обращения: 08.04.2025). – Режим доступа: свободный.
3. Основы Serverless приложений в среде Amazon Web Services / Хабр. – Москва, [2016]. – URL: https://habr.com/ru/articles/309370/ (дата обращения: 08.04.2025). – Режим доступа: свободный.
4. Погружение в Serverless. Функции как основной элемент системы / Хабр. – Москва, [2021]. – URL: https://habr.com/ru/articles/542152/ (дата обращения: 08.04.2025). – Режим доступа: свободный.
5. Серверлесс архитектура: руководство по применению и преимуществам. – Москва, [2024]. – URL: https://ifellow.ru/media-center/serverless-arkhitektura-rukovodstvo-po-primeneniyu-i-preimushchestvam/ (дата обращения: 08.04.2025). – Режим доступа: свободный.
6. Сопоставление облаков: инструменты бессерверной разработки / Хабр. – Москва, [2022]. – URL: https://habr.com/ru/articles/671512/ (дата обращения: 08.04.2025). – Режим доступа: свободный.
7. Serverless сервисы. От AWS Lambda до Yandex Cloud Functions и их альтернатив / Хабр. – Москва, [2025]. – URL: https://habr.com/ru/companies/amvera/articles/884340/ (дата обращения: 08.04.2025). – Режим доступа: свободный.