УДК 519.683

ВЫБОР МЕЖДУ МОНОЛИТНОЙ И МИКРОСЕРВИСНОЙ АРХИТЕКТУРОЙ В РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ: АНАЛИЗ И ОБЗОР

Е.Н. Колб, 3 курс
Научный руководитель – **И.А. Янковский,** к.э.н., доцент
Полесский государственный университет

Архитектурные решения в разработке программного обеспечения играют важную роль в определении эффективности, гибкости и надежности разрабатываемых систем. Один из основных вопросов, с которым сталкиваются разработчики, заключается в выборе между монолитной и микросервисной архитектурой приложения. Этот вопрос остается предметом обсуждения, особенно в контексте современных трендов и практик в разработке программного обеспечения. В данной статье мы проанализируем факторы, влияющие на выбор между монолитной и микросервисной архитектурой, с целью предоставить более глубокое понимание преимуществ и недостатков каждой из них.

В современной разработке программного обеспечения разработчики сталкиваются с вопросом выбора между монолитной и микросервисной архитектурой. Этот выбор зависит от ряда факторов, включая требования проекта, масштаб системы, уровень сложности, ожидаемая нагрузка и другие. Важно понимать, что каждая из этих архитектур имеет свои преимущества и недостатки.

Монолитная архитектура предполагает разработку приложения как единого целого, в котором все компоненты и функции интегрированы в одной кодовой базе. Основные преимущества монолитной архитектуры включают упрощенную разработку, тестирование и развертывание, а также более простое масштабирование. При использовании монолитной архитектуры разработчики могут легко работать с одной кодовой базой. Однако недостатком может быть ограничение в гибкости и при увеличении системы. Что может повлиять на обновление отдельных компонентов приложения, а также в ограничениях масштабирования при увеличении нагрузки или объема данных.

Микросервисная архитектура представляет собой подход, при котором приложение состоит из набора независимых сервисов, каждый из которых отвечает за выполнение конкретной функциональности. Главное преимущество микросервисов заключается в их гибкости и масштабируемости, а также возможности использования различных технологий и языков программирования для каждого сервиса. Однако этот подход также увеличивает количество точек отказа [1].

Дальнейшее изучение факторов, влияющих на выбор между монолитной и микросервисной архитектурой, включает анализ уровня сложности, связанной с каждым из этих подходов. Рассмотрим увеличение сложности, которое может возникнуть при использовании микросервисной архитектуры

Микросервисная архитектура предполагает создание набора независимых сервисов, каждый из которых отвечает за конкретную функциональность. Одним из главных аспектов сложности микросервисов является увеличение необходимости управления инфраструктурой. С развертыванием микросервисов часто возникает потребность в инструментах, таких как Kubernetes, для управления всей инфраструктурой. Это включает в себя контейнер серверов, сеть, брандмауэры и другие аспекты, которые ранее не были необходимы для монолитных приложений. Вместо того, чтобы сосредотачиваться на разработке реального кода, разработчики вынуждены тратить время на управление инфраструктурой и развертыванием сервисов.

Важно отметить, что использование микросервисов также увеличивает список компетенций, которыми должны обладать разработчики. Например, в случае размещения приложения на платформе Microsoft Azure, разработчики должны иметь знания о множестве сервисов и инструментов, таких как Azure SQL, Azure Web Apps и Azure DevOps pipelines. Это требует дополнительного обучения и усилий со стороны разработчиков [2]. Кроме того, важно обратить внимание на вопрос о масштабировании. Часто аргументом в пользу микросервисной архитектуры является потребность в масштабируемости. Однако большинство компаний не имеют масштаба, который можно было бы оправдать. Масштаб монолита может обслуживать сотни тысяч, а то и миллионы пользователей, что может быть достаточным для большинства компаний. В таких случаях введение микросервисной архитектуры приводит к лишней сложности и затратам, которые не оправдываются.

Важным фактором, который часто упускается из внимания, является отсутствие необходимого персонала для управления сложной инфраструктурой. Ответственность за управление Kubernetes или аналогичными инструментами часто ложится на плечи разработчиков, что создает дополнительную нагрузку и увеличивает риск возникновения ошибок [4].

Другим аспектом, который необходимо учитывать при рассмотрении микросервисной архитектуры, является возможность создания распределенного монолита. Это концепция, при которой внутри монолитного приложения существуют некоторые компоненты, организованные как микросервисы. Однако следует быть предельно осторожным с таким подходом, поскольку он может привести к серьезным проблемам.

Распределенные монолиты наследуют как недостатки монолитов, так и недостатки микросервисов. Это означает, что они сталкиваются с увеличением сложности развертывания и изменения, а также с увеличением точек отказа. Сложнее управлять инфраструктурой, так как все компоненты должны разворачиваться и изменяться вместе, что создает дополнительные трудности при масштабировании и сопровождении [3].

Однако существуют случаи, когда использование микросервисов внутри монолитного приложения может быть обоснованным. Например, в некоторых ситуациях можно выделить определенную часть функциональности, которая может быть независимо масштабирована от остальной части приложения. Рассмотрим пример регистрации в университете: выделение части приложения, управляющей процессом регистрации, в виде микросервиса может позволить ему эффективно масштабироваться в периоды пиковой нагрузки, не влияя на работу других частей монолита. Такой подход может сэкономить ресурсы и уменьшить сложности внедрения и поддержки приложения.

Заключение. Рассмотрение выбора между монолитной архитектурой и микросервисами требует тщательного анализа и оценки конкретных потребностей проекта. Хотя микросервисы имеют свои преимущества, их применение не всегда оправдано и может повлечь за собой значительные сложности.

При работе с микросервисами важно быть осторожным, чтобы не случилось так, что вместо небольших и отдельных частей системы мы получили одну большую и сложную систему с множеством взаимосвязанных частей. Хотя есть случаи, когда микросервисы могут быть правильным решением, они не являются универсальным решением для всех проектов.

Вместо того чтобы стремиться к микросервисной архитектуре, можно рассмотреть возможность сделать монолит более модульным и менее связанным, что позволит избежать некоторых недостатков, связанных с монолитами, и в то же время сохранить их преимущества.

Важно помнить, что успешность проекта не зависит только от выбора архитектуры, но и от качества кода. Независимо от того, используете ли вы монолиты или микросервисы, важно писать оптимизированный и качественный код, который легко масштабировать и поддерживать.

Микросервисы могут показаться привлекательным решением, но они не являются панацеей для всех проблем. При принятии решения следует тщательно взвесить все факторы и выбрать архитектурный подход, который наилучшим образом соответствует конкретным требованиям вашего проекта и бизнеса.

Список использованных источников

1. Harris, C Microservices vs. Monolithic architecture / Atlasssian – Режим доступа: https://www.atlassian.com/microservices/microservices-architecture/ microservices-vs-monolith – Дата доступа : 03.04.2024

2. What are microservices? / Microservice Architecture – Режим доступа : https://microservices.io/ - Дата доступа : 03.04.2024

3. Juillet, R Web application and software: which architecture to choose? / Bocasay_ - Режим доступа : https://www.bocasay.com/web-application-software-architecture/ - Дата доступа : 03.04.2024

4. Doglio, F Монолитная и микросервисная архитектура. Сравнение / Habr - Режим доступа : https://habr.com/ru/companies/haulmont/articles/758780/ - Дата доступа : 03.04.2024