УДК 004.415.23

Проблемы монолитной архитектуры промышленных приложений и способы их решения

Любайкин Илья Владимирович, магистрант, кафедра математики и бизнес-информатики, Самарский Государственный Университет им С.П. Королева, г. Самара.

Аннотация. В статье рассматриваются монолитные архитектуры промышленных приложений, внедренные в начале 20 века и устоявшихся в современных крупных программных системах, анализируются их проблемы, не позволяющие подстраиваться под новые требования, и предлагаются способы их решения путем внедрения сервисных архитектур, которые позволяют повысить масштабируемость, надежность и доступность системы.

Ключевые слова: архитектура ПО, монолит, сервис.

Issues of monolithic architecture and their solutions

Liubaikin Ilia Vladimirovich, master student, Samara State University n.a. S.P. Korolev, Samara.

Abstract. The article analyzes modern issues of monolithic software architecture and provides their solutions

Keywords: software architecture, monolith, service.

В настоящее время в крупных программных системах, как правило, используется устоявшаяся с начала 20 века монолитная архитектура. Она представляет собой единый программный модуль, в котором представлен весь код системы. Благодаря этому, можно в кратчайшие сроки произвести изменение в любой части системы, в последующем произведя полное её обновление.

С точки зрения разделения памяти в этом случае взаимодействие разных функционирующих модулей системы происходит через общую память. Такая архитектура не предоставляет разделения логики на программном уровне, причём

в случае проблем компиляции в какой-либо части кода перестаёт функционировать вся система.

Такой тип архитектуры нашел большое распространение в крупных промышленных и образовательных программных системах, а так же в системах с открытым исходным кодом. С начала 20 века, когда ИТ-индустрия стремительно набирала обороты, создавались первые крупные приложения для бизнеса, государства или других целей, монолитный тип системы представлял собой наиболее удобный тип для разработки и проектирования.

Это обуславливалось тем, что требования к приложениям успешно покрывались единым программным модулем, что разработчики могли быстрее вникнуть в способы функционирования системы, поскольку имели доступ ко всему коду. Также, важной особенностью является то, что в начале века серверная инфраструктура не предполагала модель «Инфраструктура как сервис», а требовала обязательного наличия у каждой компании-производителя программного обеспечения своих серверов, на которых велось развертывание, тестирование, и поддержка разрабатываемого программного обеспечения.

Однако современные тенденции показывают, что с ростом функциональных требований к современным программным системам, с повышением требований к надежности и работоспособности систем, и с увеличением конкуренции на рынке программных систем, вести разработку на основе монолитной архитектуры становится всё менее актуально, так как она не удовлетворяет современным требованиям.

Минусами монолитной архитектуры являются требования к прохождению компиляции всего кода системы, что делает её полностью неработоспособной при малейшей ошибке; сложное разделение логики кода, которое усложняет разбиение ответственности между командами разработчиков за тот или иной функционал; необходимость полного обновления системы при любом изменении кода, что требует отключения функционала на длительный срок, что не позволительно в случаях, когда приложения используются в бизнес-целях.

Решениями данной проблемы изначально предполагались двухзвенная и трехзвенная архитектуры. Они позволяли разделить логику между программным графическим интерфейсом и сервером-обработчиком. В качестве третьего звена выступала база данных. Однако подобного рода архитектуры, хоть и позволяют разделить функционал между клиентом и сервером, не обладают достаточным уровнем масштабируемости для удовлетворения требований к современным системам.

С недавних пор лидеры мировой ИТ-индустрии начали продвигать переход на сервисную архитектуру. Она предполагает наличие в системе программных модулей, каждый из которых выполняет четко определенную функцию. Связь между сервисами ведётся не через общую память, а посредством протокола HTTP.

Сервисная архитектура является более сложной для понимания разработчиков, так как необходимо спроектировать сервисы так, чтобы каждый из них не брал на себя лишней ответственности, работал исключительно с собственной базой данных и предоставлял программный интерфейс приложения (API) для остальных сервисов.

Приложения на таком типе архитектуры разворачиваются, как правило, на облачных сервисах, которые предоставляют свои вычислительные ресурсы посредством сети Интернет. Такими облачными сервисами являются Openshift, Kubernetes и др.

Главными плюсами сервисной архитектуры является высокая масштабируемость, позволяющая создать сколь угодно много сервисов, подстраиваясь под возрастающие требования к системе, а так же возможность четкого разделения команд разработчиков между сервисами, что позволяет обозначить границы зоны ответственности каждой из команд. В отличие от монолитной, сервисная архитектура требует перезагрузки исключительно того сервиса, в котором был добавлен функционал, что позволяет поддерживать высокую доступность системы [1].

Таким образом, внедрение сервисной архитектуры позволит ИТ-компаниям подстраиваться под возрастающие требования рынка и поддерживать свою конкурентоспособность.

Список используемых источников:

1. Осипенко А.А. Переход от монолита к микросервисам [Электронный ресурс]. URL: **https://habr.com/post/305826/** (Дата обращения: 17.12.2018).