**РАЗРАБОТКА БИЗНЕС-ПРИЛОЖЕНИЙ НА ОСНОВЕ МИКРОСЕРВИСОВ**

**DEVELOPMENT OF BUSINESS APPLICATIONS BASED ON MICROSERVICES**

Ворсин В.А.

Санкт-Петербургский государственный экономический университет,

г. Санкт-Петербург, Россия

Исследованы теоретические и методологические аспекты разработки бизнес-приложений на основе микросервисной архитектуры. Рассмотрены определение микросервиса, плюсы и минусы архитектуры, ее отличия от монолитной архитектуры, отличия от SOA и Web-сервиса, связи в микросеврисах, контейнеры и шаблоны проектирования.

Ключевые слова: информационные системы, бизнес-приложения, информационные технологии, микросервисы, архитектура, контейнеры.

The theoretical and methodological aspects of developing business applications based on microservice architecture are investigated. The definition of microservice, the pros and cons of architecture, its differences from monolithic architecture, differences from SOA and Web service, communications in microservices, containers and design patterns are considered.

Key words: information systems, business applications, information technologies, microservices, architecture, containers.

Микросервисы - это метод разработки программного обеспечения, вариант структурного стиля сервис-ориентированной архитектуры (SOA), который организует приложение как набор связанных сервисов. В архитектуре микросервисов сервисы детализированы, а протоколы весят достаточно мало.

«Микро» в микросервисах подразумевает, что это небольшие приложения. Иногда это правда, но лучше думать о них так, что они могут быть настолько большими, насколько это необходимо, чтобы сделать одну конкретную вещь или решить одну конкретную проблему. Эта проблема должна быть концептуальной, а не технической. Как заявляет Microsoft, «микросервисы должны быть построены вокруг бизнес-возможностей, а не горизонтальных уровней, таких как доступ к данным или обмен сообщениями». Они взаимодействуют с другими микросервисами и внешними пользователями через относительно стабильные API для создания более крупного приложения.

Таким образом, внутренняя функциональность отдельного микросервиса может быть изменена или радикально улучшена, не затрагивая остальную часть системы.

Мы часто слышим, как о микросервисах говорят в терминах «архитектуры микросервисов ». Эта фраза охватывает не только сами микросервисы, но и компоненты для управления и обнаружения услуг, а также шлюз API, который управляет связью между микросервисами и внешним миром.

«Монолитная архитектура» является противоположностью микросервисов. В данной архитектуре весь код находится в одном большом двоичном исполняемом файле. Монолитное приложение сложнее масштабировать и труднее улучшать, но поскольку оно построено как единое связное приложение, оно не требует такого большого количества управления, как архитектура микросервисов. Разницу в построении архитектур вы можете видеть на Рис.1.

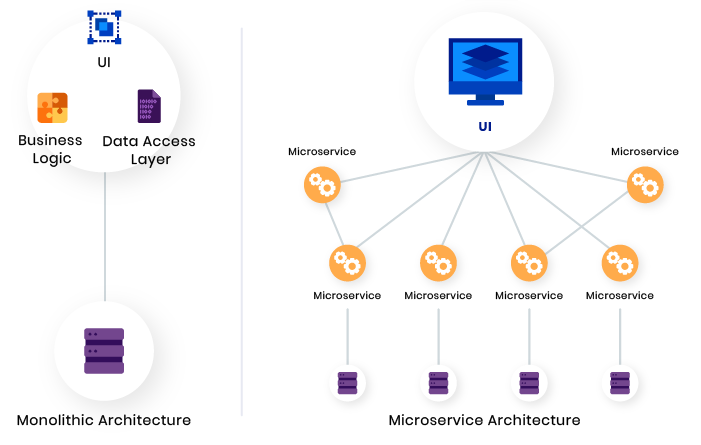


Рисунок 1. Монолитная и микросервисная архитектуры

Микросервисы должны выполнять одну конкретную функцию. К сожалению, на практике их функциональность часто запутывается, а разделение общей картины на составляющие становится все сложнее. Анализ предметной области и проектирование на основе предметной области - это теоретические подходы, которые помогают разделить общую задачу на отдельные проблемы, которые могут решать микросервисы.

Вы создаете абстрактную модель своей бизнес-области и в процессе обнаруживаете ограниченные контексты, которые объединяют функциональные возможности, которые определенным образом взаимодействуют со внешней средой.

Идея совместной работы небольших отдельных программ может напомнить вам о SOA (сервис-ориентированная архитектура) и Web-сервисах. Но между этими концепциями и микросервисами есть колоссальные различия. Вот основные из них:

* В SOA отдельные компоненты относительно тесно связаны между собой, часто разделяя ресурсы, такие как хранилище, и обмениваются данными через специальное программное обеспечение, называемое корпоративной шиной хранения. Микросервисы более независимы, совместно используют меньше ресурсов и обмениваются данными по более легким протоколам. Следует отметить, что микросервисы возникли из среды SOA и иногда считаются своего рода SOA или преемником этой концепции.
* Web-сервис - это общедоступный набор функций, к которым другие приложения могут обращаться через Интернет. Вероятно, наиболее распространенным примером является GoogleMaps, в который можно вставлять ссылки сайтов своих заведений, для более качественного информирования клиентов о веремени работы заведения. Это гораздо более слабое соединение, чем в архитектуре микросервисов.

Микросервисы должны стремиться использовать базовые и устоявшиеся методы связи, а не сложную и тесную интеграцию. Как уже отмечалось, это еще одна вещь, которая отличает микросервисы от сервис-ориентированной архитектуры. Связь между микросервисами должна быть асинхронной, в том смысле, что потоки кода не должны блокироватья в ожидании ответов. (Все еще нормально использовать протоколы синхронной связи, такие как HTTP, хотя асинхронные протоколы, такие как AMQP (Advanced Message Queuing Protocol), также распространены в архитектурах микросервисов.) Этот вид слабой связи делает архитектуру микросервисов более гибкой в ​​условиях сбоя отдельных компонентов или частей сети, что является ключевым преимуществом.

Основной технологией, которая пошла дальше всего в направлении внедрения микросервисов в массовое производство, являются контейнеры. Контейнер подобен экземпляру виртуальной машины, но вместо того, чтобы включать в себя всю автономную ОС, контейнер представляет собой просто изолированное пространство пользователя, которое использует ядро ​​операционной системы хоста, но в остальном сохраняет выполнение кода внутри него самодостаточным. Контейнеры намного меньше, чем экземпляры виртуальных машин, и их легко быстро развернуть как локально, так и в облаке, также их можно увеличить или уменьшить в соответствии с потребностями и доступными ресурсами.

Привлекательность контейнеров для микросервисов должна быть очевидной: каждый отдельный микросервис может работать в своем собственном контейнере, что значительно сокращает издержки на управление службами. Большинство реализаций контейнеров имеют дополнительные инструменты оркестровки, которые автоматизируют развертывание, управление, масштабирование, создание сетей и доступность приложений на основе контейнеров. Это сочетание небольших, простых в сборке микросервисов и простых в развертывании контейнеров, что делает возможной философию DevOps. Существует несколько реализаций концепции контейнера, но на сегодняшний день наиболее популярным является Docker , который обычно соединяется с Kubernetes в качестве платформы оркестровки.

Системы на основе контейнеров, с другой стороны, являются полиглотами: любой язык программирования, поддерживаемый ОС, может работать в контейнере, что дает программистам большую гибкость. Большое преимущество микросервисов состоит в том, что каждый отдельный сервис может быть написан на любом языке, который более информативный и удобный для разработчика. Действительно, служба может быть полностью перестроена на новом языке, не затрагивая систему в целом, если ее API остаются стабильными.

Независимо от того, какой язык вы используете для разработки микросервисов, вы столкнетесь с проблемами, с которыми ранее сталкивались другие разработчики. Шаблоны проектирования являются формализованными, абстрактными решениями проблем, а некоторые из них предназначены специально для микросервисов. Вот некоторые из них:

* Реестр сервисов: для подключения клиентов к доступным экземплярам микросервисов
* Посол: для подключения основного контейнера к внешнему миру, например, для проксирования локальных соединений с внешними
* Резерв: для предоставления альтернативы отказавшему сервису
* Sidecar: для предоставления вспомогательных услуг для основного контейнера, таких как ведение журнала, синхронизация служб или мониторинг
* Автоматический выключатель: для предотвращения повторного вызова сбойных служб
* Адаптер: для стандартизации или нормализации интерфейса между основным контейнером и внешним миром.

Одним из преимуществ использования контейнеров является то, что они могут быть легко развернуты в облаке, где доступны гибкие вычислительные ресурсы, позволяющие максимизировать эффективность вашего приложения. Не удивительно, что все крупные поставщики общедоступных облачных сервисов хотят, чтобы вы использовали их платформы для запуска своих приложений на базе микросервисов.

Хотелось бы выделить модульную платформу для разработки ПО .NET Core от Microsoft, созданном на основе упрощенной архитектуры микросервисов и контейнеров Docker, о которых упоминалось выше. Приложение является кроссплатформенным как на стороне сервера, так и на стороне клиента благодаря запуску служб .NET Core в контейнерах Windows или Linux, а также благодаря Xamarin для мобильных приложений, выполняемых на платформах Android, iOS или Windows/UWP.

В качестве протокола связи между клиентскими приложениями и микросервисами служит HTTP. Кроме того, поддерживается асинхронная связь для передачи обновленных данных нескольким службам на основе событий интеграции и шины событий, и множество других функций, определенных заранее в плане развития продукта.

Что касается текущей конфигурации этого решения для среды разработки, базы данных SQL с образцами данных развертываются автоматически в один контейнер SQL Server для Linux, поэтому все решение можно запустить независимо от облака или конкретного сервера. Каждую базу данных также можно развернуть в виде отдельного контейнера Docker, но тогда понадобится свыше 8 ГБ оперативной памяти для Docker на машине разработчика, чтобы запускать три контейнера Docker для SQL Server на используемом узле Docker Linux в среде разработки «Docker для Windows» или «Docker для Mac».

Однако в реальной экосистеме рекомендуется развертывать БД в службах высокой доступности, таких как БД Azure SQL, Redis как услуга и Azure CosmosDB, а не в контейнере MongoDB. При переходе на производственную конфигурацию вам просто нужно будет изменить строки подключения после настройки серверов в высоко доступном облаке или локально.

**Список литературы:**

1. Крис Ричардсон, Микросервисы. Паттерны разработки и рефакторинга // Питер. -2019. –С. 185-187
2. Michelle Major–Goldsmith, Simon Dorst, Service Integration and Management (SIAM®) Professional Body of Knowledge // Scopism Limited. -2017. –С. 215-216
3. Dave Hornford, Sriram Sabesan, Vidhya Sriram, Ken Street, The Seven Levers of Digital Transformation // The Open Group. -2017. C. 26-27
4. ГОСТ Р 56713-2015 (ISO/IEC/IEEE 15289:2011) Системная и программная инженерия. Содержание информационных продуктов процесса жизненного цикла систем и программного обеспечения (документация)
5. ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств