А. Е. Любанец

ОПТИМИЗАЦИЯ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ ПРОЛОЖЕНИЯ

Рассматриваются подходы к балансированию недостатков и преимуществ микросервисной архитекруры. Предлагаются подходы и средства для мониторинга, развертывания и разработки программ с микросервисной архитектурой .

Введение

За последнее десятилетие микросервисная архитектура (MCA) получила широкое распростарнение в связи с развитием гибких подходов к разработке ПО. В частности MCA предлагает простоту тестирования отдельных частей приложения, независимость развертывания, гибкость в отказоустойчивости и масшабировании как всей системы так и ее отдельных частей. Однако при декомпозиции и разделении на отдельные сервисы возникает вопрос во взаимодейсвтии и интеграции всех частей.

I. Организация сервиса вокруг его целей

Определение границ сервиса — самый важный шаг. От этого будет зависеть весь жизненый цикл микросервиса. Основной принцип определения зоны ответственности микросервиса – сформировать её вокруг некоторой бизнес-задачи. И чем она компактнее, чем формализованней её взаимоотношения с другими областями, тем проще создать новый микросервис. На этом основывается создание любых других компонентов. Однако на начальном этапе разработки бывает достаточно сложно определить четки границы или в середине разработки границы бизнесс-задачи сместились, что потребует изменения функциональности в других микросервисах.

Для минимизации ошибок при определении границ оправданным является подход Monolith First, когда вначале систему развивают в традиционной парадигме, а когда появляются устоявшиеся области, их выделяют в микросервисы. А при изменении границ главное, чтобы выигрыш от разбиения превышал сложности пересмотра этих границ.

II. Взаимодействие и интеграция

Одним из саммых сложных и важных вопросов проектирования MCA является взаимодействие между сервисами. Цепочка взаимодействия различных сервисов может быть различна и поэтому стоит использовать асинхронное взаимодействие (Kafka, asynchronous REST и т.д.). Это позволит более гибко настривать время ожидания от какого-нибудь медленного сервиса или организовать политику retry в случаях ошибки в сети.

Увеличение количество запросов между сервисами вынуждает уделить особое внимания мониторингу как всей системы так и отдельных ее частей (Prometheus, Grafana, Moira и т.п.). Система мониторинга для приложений на основе микросервисов должна отоброжать постоянное изменение ресурсов, иметь возможность собирать данные мониторинга в центральном местоположении и отображать информацию, отражающую часто меняющийся характер приложений на микросервисах.

Рост количества различных сервисов вынуждает автоматизировать процессы сборки и развертывания. Однако хорошо выстроенный автоматизированный процесс развертывания сервисов по отдельности позволяет осуществлять релизы часто и с нулевым временем простоя, что облегчает процессы тестирования и ускоряет время доставки ПО конечным пользователям.

III. Выводы

Написание независимого сервиса дает свободу в выборе средств и языков программирования наиболее подходящих для решения конкретной задачи. Для построения эффективной микросервисной инфраструктуры требуется инвестировать время в автоматизацию процессов разработки. Это позволит сделать процесс развертывание системы из многих составляющих (микросервисы) прозрачным и единообразным .

1. Martin L. Abbott. R. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise., Third Edition // Addison-Wesley Professional,. – 2015. – С. 624.
2. Martin Fowler, Microservices. - https://martinfowler.com/articles/microservices.html
3. Vikram Murugesan, Microservices Deployment Cookbook // Packt Publishing,.‘–2017.‘–С. 280

Любанец Андрей Евгеньевич, магистрант кафедры информационных технологий автоматизированных систем БГУИР, andr0sparrow@gmail.com.

Научный руководитель: Матвеенко Владимир Владимирович, кандидат физико-математических наук, доцент, vladzimir66@bsuir.by.