УДК 004.9

ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ НА ОСНОВЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ, ЕГО РАЗВЕРТЫВАНИЕ И СОПРОВОЖДЕНИЕ В ОБЛАЧНОЙ СРЕДЕ

Рывков С.С.

Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь

Научный руководитель: Вышинский Н.В. – канд. техн. наук, профессор

Аннотация. Данная статья предоставляет основные аспекты и подходы, связанные с разработкой приложения на основе микросервисной архитектуры. Рассматриваются принципы взаимодействия компонент приложения в данной архитектуре, а также инструменты, позволяющие настроить инфраструктуру приложения для объединения разрозненных сервисов в единый механизм взаимодействия с последующим разворачиванием ее в облачной среде.

Ключевые слова: микросервис, архитектура, инфраструктура, контейнер, Docker, AWS, виртуальная машина, облачная среда, балансировщик.

Введение. Ввиду глобальной цифровизации, с каждым годом активных пользователей различных веб-сервисов и приложений становится больше. В связи с этим, начиная примерно с 2010 годов, появилась проблема нагрузок на сервисы различных приложений, написанных в основном с использованием монолитной или сервис-ориентированной архитектур. Для решения этой проблемы еще в начале 2000х годов была представлена микросервисная архитектура, позволяющая проектировать приложение наиболее гибким образом, а также максимально быстро масштабировать его при больших нагрузках.

Несмотря на то, что данная архитектура появилась в начале 2000х, активно применять ее стали только во второй половине 2010х, поскольку на тот момент еще не появился инструмент с достаточно хорошей инфраструктурой, позволяющий разработать подобное приложение. Разработка такой инфраструктуры требует значительных финансовых инвестиций, поэтому неудивительно, что впервые подобные средства представили такие компании как Amazon, с их Amazon Web Services, и Microsoft с Azure Cloud.

В данной статье микросервисная архитектура будет реализована при помощи инфраструктуры Amazon Web Services.

Основная часть. Для того, чтобы спроектировать приложение на основе микросервисов, нужно решить следующие задачи:

- выделение значимых частей приложения в отдельные сервисы
- подготовка контейнеров для отдельного сервиса
- настройка инфраструктуры приложения в облачном провайдере
- разворачивание приложение в облачной среде

Рассмотрим в подробностях особенности микросервисной архитектуры, на основе которой будет строиться предполагаемый сервис.

Микросервисная архтектура – особая форма сервис-ориентированного подхода к архитектуре, которая упорядочивает приложение как набор слабо связанных сервисов. В архитектуре микросервисов сервисы выполняют одну конкретную цель, а протоколы общения между микросервисами в приложении максимально легковесны, чтобы обеспечить быструю и качественную коммуникацию между ними.

В приведенной на рисунке 1 видна общая схема такой архитектуры

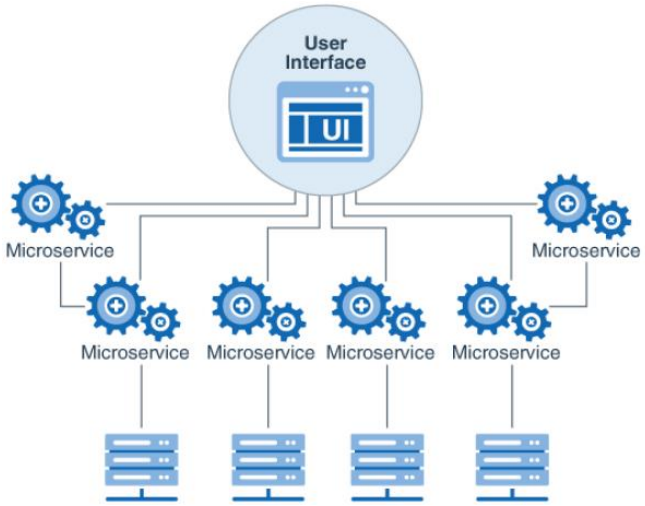

Рисунок 1 – Общая архитектура микросервисного подхода

Далее рассмотрим то, как данная архитектура применяется в сервисах AWS:

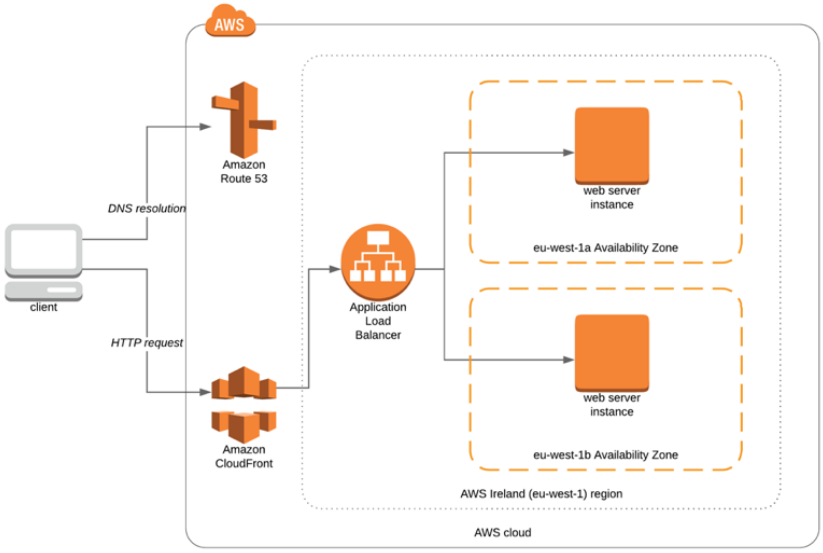

Рисунок 2 – Общая схема взаимодействия клиентского приложения с инфраструктурой AWS [1]

Как видно из схемы, клиентское приложение через сеть Интернет обращается к внутренним сервисам, которые разрешают DNS для интернет-адресов (компонента Route 53), а также кэшируют и дополнительно проверяет трафик к ключевым сервисам на наличие незащищенного соединения или возможных атак (компонента CloudFront). После идет компонента Load Balancer, отвечающая за равномерное распределение нагрузки на приложение.

Для выделения значимых частей приложения определимся с доменом приложения. Для примера был взят сервис по агрегации новостей. В данном сервисе выделим сервисы по принципу единственной ответственности [2]: сервис авторизации для обработки регистрации и аутентификации пользователей; сервис по сбору статистики информационных порталов, который содержит в себе API для сбора информации о просматриваемых новостях, а также подписок на порталы, для сбора конечной статистики об использовании; сервис перенаправления запросов, реализующий паттерн Front Controller, который позволяет избежать использования нескольких контроллеров и используется для применения политик в масштабе всего приложения, таких как отслеживание пользователей и безопасности [3]; непосредственный сервис серверной части приложения, обрабатывающего запросы с клиентских приложений:

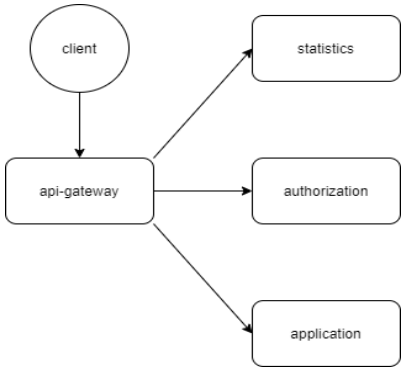

Рисунок 3 – Схема взаимодействия сервисов агрегатора новостей

После условной реализации представленных компонент, их нужно обернуть в особую абстракцию: контейнер. Контейнер представляет собой виртуальную машину, которая может использоваться для разных задач. В данном случае, контейнер будет являться оберткой над сервисами, написанными для приложения. Большим преимуществом данного подхода является свойство изолированности: контейнеры не могут повлиять на работу друг друга, следственно надежность таких приложений значительно повышается.

Стандартом для контейнеризации сервисов является инструмент Docker. Docker предоставляет полный API над виртуальной машиной, позволяющий указывать такие параметры как: порт; внутреннюю сеть, по которой обмениваются сообщениями внутренние контейнеры; тег контейнера; особые инструкции при запуске и тому подобное.

Docker контейнеры конфигурируются особыми файлами с расширением Dockerfile, в котором указываются все аспекты сборки и запуска конкретного приложения, предназначенного для развертывания в данном контейнере, а также использованию различных переменных, таких, например, как тайм-зона, а также различные ключи безопасности [4].

Инструкции, записанные в Dockerfile, будут использованы компонентами AWS для виртуализации в среде Amazon.

Инфраструктура Amazon предоставляет сервис Elastic Compute Cloud или EC2. Этот сервис предоставляет возможность разворачивания и управления удаленной виртуальной машиной, находящейся на серверах Amazon.

Для начала нужно настроить данную компоненту. Для этого нам потребуется в аккаунте Amazon перейти в раздел создания компонент, выбрать EC2 и приступить к его настройке. В настройках необходимо сконфигурировать следующие пункты:

–Выбрать AMI (Amazon Machine Image). Это настройка позволяет выбрать операционную систему, под которой будет запускать приложение. В нашем случае будет выбрана ОС Linux ввиду ее легковесности, а также высокой стабильности работы.

– Выбрать тип экземпляра. Типы экземпляров включают различные комбинации центральных процессоров, памяти, хранилища, пропускной способности сети и дают гибкость в выборе подходящего сочетания ресурсов для ваших приложений.

– Настройка экземпляра. В данном подпункте дается возможность выбрать изначальное количество экземпляров определенного сервиса, если сервис является высоконагруженным. Также конфигурируются настройки сети, его подсеть, IP-адрес, а также имя хоста.

– Добавление физической памяти. Данный подпункт позволяет сконфигурировать размер жесткого диска для экземпляра приложения. Это важно, если сервис хранит большое количество информации внутри виртуальной машины. Это такая информация как хэш, файлы с результатами логгирования и т.п.

– Добавление тэгов. Присваивает уникальное имя для сервиса и предназначено для упрощенной навигации среди большого количества сервисов в аккаунте AWS.

– Настройка пользовательских политик безопасности. Нужно для того, чтобы доступ к контейнеру имели только привилегированные пользователи.

После успешной настройки, виртуальная машина запускается и к ней появляется доступ через средства удаленного управления.

Последний шаг – развертывание приложения в виртуальной машине. Для этого также понадобится настройка: создать группу развертывания, которую нужно назвать, а также привязать аккаунт GitHub, после чего указать конкретный репозиторий, код которого и будет использован инструкцией Dockerfile для последующего запуска приложения.

Инфраструктура AWS устроена таким образом, что после того, как репозиторий был привязан к группе развертывания, каждый раз, когда в основной ветке репозитория происходит обновление, Amazon автоматически пересоберёт новый образ на основе нового кода и развертывает приложение. Данный процесс называется Continues Delivery или непрерывная доставка и позволяет в короткие сроки доставлять новую функциональность пользователям приложения [5].

Если определенный сервис по какой-то причине вышел из строя, средства AWS позволяют быстро перезапустить контейнер с сервисом. Также, есть возможность мгновенно добавить новые экземпляры сервиса, если нагрузка на приложение резко повысилась, и после убрать лишние после того, как пиковые значения проходят.

Заключение. Выполнен анализ основных аспектов микросервисной архитектуры, рассмотрены причины использования данного подхода в построении приложения. Также была изучена наиболее популярная инфраструктурная платформа для развертывания микросервисов, ее основные компоненты, настройки данных компонент, а также их предназначение. Рассмотрены инструменты сопровождения развернутых приложений в облачной среде, а также рассмотрен инструмент непрерывной доставки, наряду с динамическим изменением нагрузочных возможностей приложения.

Список литературы

1. Clean code: A Handbook of Agile Software Craftsmanship / Robert Cecil Martin // Pearson Education – 2010. – 210 p.

2. Building Microservice: Designing Fine-Grained Systems / Sam Newman // O'Reilly Media, Incorporated – 2014. – Pp. 9–27.

3. Amazon Web Services in Action / Michael Wittig, Andreas Wittig, Aldir Jose Coelho Correa Da Silva // Manning Publications – 2015. – Pp. 31–41.

4. Docker in action / Jeff Nickoloff, Stephen Kuenzli // Manning Publications. – 2019. – Pp. 47–62.

5. The Ultimate Guide from Beginners To Advanced For The Amazon Web Services / Theo H. King. // Independently Published. – 2019. – Pp. 47–60.

UDC 004.9

DESIGNING AN APPLICATION BASED ON A MICRO-SERVICE ARCHITECTURE, DEPLOYING AND MAINTAINING IT IN A CLOUD ENVIRONMENT

Ryvkov S.S.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Vyshinskiy N.V. – PhD of Engineering Sciences, professor

Annotation. This article provides the main aspects and approaches associated with developing an application based on a microservice architecture. Considered principles of interaction between application components in scope of microservices architecture, as well as tools that allow to configure application infrastructure in order to combine disparate services into a single interaction mechanism with its subsequent deployment in the cloud environment.

Keywords: microservice, architecture, infrastructure, container, Docker, AWS, virtual machine, cloud environment, load balancer.