8. NCOIC Interoperability Framework (NIF v. 2.1) and NIF Solution Description Reference Manual (NSD-RM v. 1.2). [S. l.]: NCOIC, 2008. 125 p.
9. Systems, Capabilities, Operations, Programs, and Enterprises (SCOPE) Model for Interoperability Assessment. Version 1.0. [S. l.]: NCOIC, 2008. 154 p.

Literatura

1. *Bashlykova A.A., Kozlov S.V., Makarenko S.I., Olejnikov A.Ya., Fomin I.A.* Podkhod k obespecheniyu interoperabel'nosti v setetsentricheskikh sistemakh upravleniya // Zhurnal radioelektroniki. 2020. № 6.
2. GOST R 55062–2012. Informatsionnye tekhnologii (IT). Sistemy promyshlennoj avtomatizatsii i ikh integratsiya. Interoperabel'nost'. Osnovnye polozheniya. M.: Standartinform, 2014. 12 s.
3. GOST R ISO 11354-1–2012. Usovershenstvovannye avtomatizirovannye tekhnologii i ikh primenenie. Trebovaniya k ustanovleniyu interoperabel'nosti protsessov promyshlennykh predpriyatij. Ch. 1: Osnova interoperabel'nosti predpriyatij // Kodeks. URL: http://docs.cntd.ru/document/1200102044 (data obrashcheniya: 09.07.2020).
4. GOST R ISO/MEK 25010–2015. Informatsionnye tekhnologii (IT). Sistemnaya i programmnaya inzheneriya. Trebovaniya i otsenka kachestva sistem i programmnogo obespecheniya (SQuaRE). Modeli kachestva sistem i programmnykh produktov // Kodeks. URL: http://docs.cntd.ru/document/1200121069 (data obrashcheniya: 09.07.2020).
5. GOST R ISO/MEK 25012. Sistemnaya i programmnaya inzheneriya. Trebovaniya i otsenka kachestva sistem i programmnykh sredstv (SQuaRE). Modeli kachestva dannykh // Kodeks. URL: http://docs.cntd.ru/document/1200121069 (data obrashcheniya: 09.07.2020).
6. GOST R ISO/MEK 25024. Sistemnaya i programmnaya inzheneriya. Trebovaniya i otsenka kachestva sistem i programmnykh sredstv (SQuaRE). Izmerenie kachestva dannykh // Kodeks. URL: http://docs.cntd.ru/document/1200111326 (data obrashcheniya: 09.07.2020).
7. *Makarenko S.I., Ivanov M.S.* Setetsentricheskaya vojna – printsipy, tekhnologii, primery i perspektivy: monografiya. SPb.: Naukoemkie tekhnologii, 2018. 145 s.
8. NCOIC Interoperability Framework (NIF v. 2.1) and NIF Solution Description Reference Manual (NSD-RM v. 1.2). [S. l.]: NCOIC, 2008. 125 p.
9. Systems, Capabilities, Operations, Programs, and Enterprises (SCOPE) Model for Interoperability Assessment. Version 1.0. [S. l.]: NCOIC, 2008. 154 p.

DOI: 10.25586/RNU.V9187.20.03.P.131 УДК 004.451

Г.В. Резенов

АРХИТЕКТУРА МИКРОСЕРВИСОВ И ЕЕ РЕАЛИЗАЦИЯ С ПОМОЩЬЮ ТЕХНОЛОГИИ КОНТЕЙНЕРИЗАЦИИ

Рассмотрена архитектура микросервисов, причины популярности ее использования в последние несколько лет, взаимосвязь между микросервисами и технологией контейнеризации, которая стала основным способом ее реализации.

Ключевые слова: микросервисы, монолитная архитектура, контейнер, Docker, Kubernetes.

G.V. Rezenov

ARCHITECTURE OF MICROSERVICES AND ITS IMPLEMENTATION WITH CONTAINERIZATION TECHNOLOGY

The architecture of microservices, the reasons for its popularity in the last few years, the relationship between microservices and containerization technology, which has become the main way of its implementation, are considered.
Keywords: microservices, monolithic architecture, container, Docker, Kubernetes.

Введение

Два года назад я начал работать в компании, предоставляющей услуги хостинга. Хостинг – это предоставление ресурсов сервера, постоянно подключенного к сети, как правило, к Интернету. Сначала я был оператором технической поддержки, затем стал дежурным инженером отдела эксплуатации. За это время я достаточно неплохо познакомился с операционной системой Linux и поэтому решил стать DevOps-инженером. В ходе изучения того, какими навыками должен обладать этот специалист, я все чаще сталкивался со словами «микросервисы», «контейнеры», «оркестрация». Я начал читать специальную литературу, блоги на интернет-форумах системного администрирования и обнаружил, что в последние годы в области системного администрирования стали очень популярны микросервисы и технология контейнеризации. Пытаясь ответить на вопрос, почему все мировые IT-гиганты переходят на использование контейнеров в своей инфраструктуре, я получил ответы, представленные ниже.

Как я познакомился с микросервисами

В последние годы в кругах специалистов технической направленности все чаще упоминается термин «микросервисы». Особенно часто им пользуются в области разработки программного обеспечения. Недавно я стал работать тестировщиком ПО в одной из компаний по производству отечественного оборудования, поэтому и сам невольно стал тем, кто попал в данную сферу деятельности. После этого я и задался вопросом, что же такое микросервисы и почему они сейчас так популярны. Для ответа на этот вопрос не так просто найти информацию, так как понятие является относительно новым. При этом его часто путают с сервисориентированной архитектурой (SOA), что также затрудняет поиски соответствующей литературы.

Оказалось, что термин «микросервисы» описывает один из стилей разработки ПО. Данный подход говорит о том, что любое сложное приложение лучше реализовывать как набор небольших сервисов, каждый из которых работает как отдельный процесс и взаимодействует с остальными с помощью простых механизмов запросов, например HTTP. Как правило, сервисы разворачиваются в полностью автоматизированной среде, в которой присутствует минимально необходимое централизованное управление. Интересен также момент, что сервисы могут быть написаны на разных языках программирования, но при этом будут корректно взаимодействовать между собой.

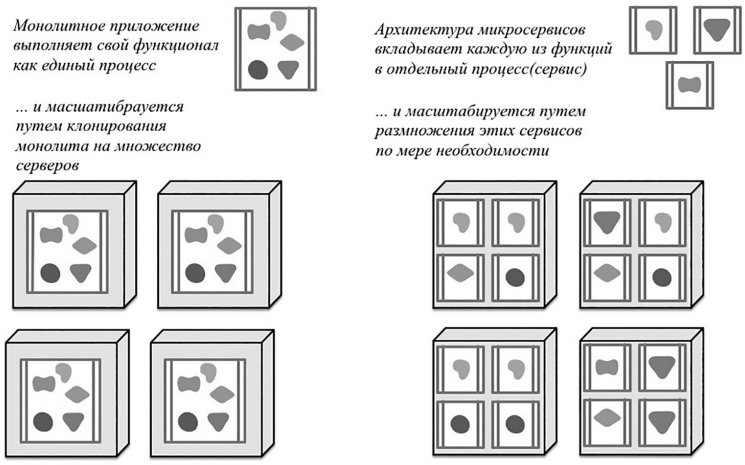
Причины для использования микросервисов

Традиционно программисты разрабатывают продукт как единую систему – монолит. Это знакомо и мне тоже, потому что в рамках изучения объектноориентированного программирования, будучи студентом бакалавриата, я писал несложные программы. Как правило, их структура была разбита на отдельные классы, каждый из которых выполнял свою определенную функцию. Тем не менее это было единое монолитное приложение. Будучи оператором технической поддержки в хостинг-компании, я убедился, что примером монолитного приложения в Интернете является почти любой из сайтов. Он, как правило, всегда имеет пользовательский интерфейс в виде HTML-страниц, серверную часть и, почти всегда, базу данных. Данный подход довольно очевиден, ведь приложение само обрабатывает HTTP-запросы, обновляет и запрашивает информацию в базе данных, направляет в ответ клиенту HTML-страницу с необходимой информацией. Однако при внесении любых изменений в код приложения, его необходимо заново пересобрать и развернуть на сервере [9].

В наше время все большее распространение получают облачные технологии. Крупные компании не всегда готовы потратить большие деньги на то, чтобы развернуть собственные серверные мощности, поэтому они обращаются к облачным провайдерам, которые специализируются на их аренде. Причин этого несколько: 1) пропадает необходимость в самостоятельной модернизации и обновлении технического оборудования; 2) удобная среда для тестирования нового ПО, которую можно легко изменить с помощью специалистов провайдера; 3) более эффективное расходование средств, которое достигается за счет того, что у провайдера серверные мощности задействованы примерно на 80%, тогда как в частной ИТ-инфраструктуре, как правило, задействуется лишь 10–15% от максимума. Получается, что за меньшие деньги мы получаем те же серверные мощности [3].

Не удивительно, что с переходом компаний к облачным технологиям монолитные приложения тоже переместились в облако. Через некоторое время оказалось, что поддерживать их работу там не очень удобно. Как было сказано ранее, из-за своей монолитной архитектуры внесение изменений в приложение требует его полной пересборки и загрузки на сервер, т.е. в облако. Это очень неудобно, если изменение требуется лишь в одном из модулей приложения. Еще одним минусом стало масштабирование монолита, так как для этого создаются его полные копии, между которыми затем распределяются входящие запросы. Из-за этого объем необходимых для работы приложения ресурсов вырастает в разы.

Для решения этих проблем подошел механизм микросервисов. Разбиение монолитного приложения на отдельные самостоятельные сервисы позволило четко разграничить область взаимодействия основных модулей между собой. Обновление отдельного модуля происходит быстрее, чем обновление всего приложения. Наглядное изображение преимущества микросервисов над монолитами в своем блоге представил Мартин Фоулер (рис.).



Сравнение масштабирования монолитов и микросервисов

Реализация микросервисов с помощью контейнеров

Архитектура микросервисов – это хорошо, но как ее реализовать на практике? Ответ на этот вопрос частично можно найти в принципах проектирования, использованных в Unix. Один из них гласил, что за каждую функцию в системе должен отвечать отдельный процесс [5]. С микросервисом та же ситуация: необходимо поместить часть приложения в некую изолированную среду, чтобы она могла выполнять внутри нее свою функцию. Одним из первых механизмов, позволявших выполнить подобную изоляцию, стало изменение корневого каталога в UNIX-подобных операционных системах с помощью операции *chroot*. Программа, запускаемая с измененным корневым каталогом, имела доступ только к тем файлам, которые находятся в этом каталоге [2]. Подобный подход изоляции клиентского доступа на общем сервере используется в хостинг-компаниях до сих пор. В этом я убедился, будучи оператором технической поддержки в одной из них. Хотя данный механизм до сих пор используют, сфера его применения довольно ограничена, из-за этого он не особенно популярен среди основной массы разработчиков. Еще одним аргументом не в пользу *chroot* является то, что он был создан в далеком 1982 г. С тех пор в нем нашли несколько серьезных программных уязвимостей.

Всплеск популярности микросервисов произошел в 2013 г., когда компания Google представила обществу разработчиков исходный код платформы Docker. В основу данной технологии легла существовавшая в Linux технология контейнеров. Контейнеры представляют собой средства инкапсуляции приложения вместе с его зависимостями. В отличие от виртуальных машин они имеют следующие преимущества:

- контейнеры совместно используют ресурсы основной ОС, что делает их на порядок более эффективными;
- переносимость контейнеров обеспечивает потенциальную возможность устранения целого класса программных ошибок, вызываемых незначительными изменениями рабочей среды, – лишается обоснования древний довод разработчика: «но это работает на моем компьютере»;

• упрощенная сущность контейнера означает, что разработчики могут одновременно запускать десятки контейнеров. Инженеры по эксплуатации могут запустить на одном хосте намного больше контейнеров, чем при использовании отдельных виртуальных машин;

• контейнеры предоставляют преимущества конечным пользователям и разработчикам без необходимости развертывания приложения в облаке. Пользователи могут загружать и запускать сложные приложения без многочасовой возни с конфигурированием и проблемами при установке и при этом не беспокоиться о каких-либо изменениях в их локальных системах. В свою очередь, разработчики подобных приложений могут избежать проблем, связанных с различиями в конфигурациях пользовательских сред и с доступностью зависимостей для этих приложений [4].

Простота и высокая скорость работы контейнеров позволяют считать их наиболее подходящими компонентами для реализации архитектуры микросервисов. С этим трудно не согласиться, ведь по сравнению с виртуальными машинами контейнеры намного меньше по размерам и гораздо быстрее разворачиваются. Данное преимущество я хорошо прочувствовал на своем примере. Раньше для эмуляции изолированной среды для запуска приложений я использовал средство виртуализации Virtual Box. Выделение ресурсов, установка ОС, ее настройка занимали много времени – около 3–4 часов. Для контейнера же достаточно скачать его образ, установить Docker и запустить его. Время выполнения будет зависеть только от скорости подключения к Интернету или локальному хранилищу. В моем случае это время, как правило, составляло всего 10–15 минут. Разница во времени колоссальна! Благодаря этому же свойству архитектура микросервисов позволяет использовать минимум ресурсов и быстро реагировать на требуемые в ней изменения.

Открытое сообщество уже сформировало огромную библиотеку готовых образов контейнеров, которые может скачать для ознакомления или использования любой желающий. Данное хранилище называется Docker Hub [7].

Управление контейнерами

Для управления контейнерами используются так называемые оркестраторы. Полина Тараненко в своей статье «Почему Kubernetes стал настолько популярен» приводит следующую информацию: «… Контейнеры в разработке используются уже около десяти лет, и, как уже было сказано, в последние несколько лет их популярность начала резко расти. На рынке есть множество решений для запуска контейнеров и их оркестрации, однако более 4/5 контейнеров запускается в среде Docker, а для оркестрации более половины пользователей используют Kubernetes. Так сложилось исторически.

Kubernetes не заменяет Docker: он серьезно расширяет его возможности, упрощая управление развертыванием, сетевой маршрутизацией, расходом ресурсов, балансировкой нагрузки и отказоустойчивостью запускаемых приложений… » [6].

Предложения по применению контейнеров

Опираясь на вышесказанное, я пришел к очевидному выводу, что контейнеры – это невероятно удобная среда для тестирования программного продукта. Обусловлено это двумя причинами: первая из них – все необходимые для работы программы модули, библиотеки и утилиты уже помещены внутри контейнера. У системного администратора, или тестировщика, больше нет необходимости разбираться, почему программа не запускается на его компьютере, в то время как на рабочей машине разработчика проблем не наблюдается.

Вторая причина заключается в том, что рабочая среда контейнера полностью изолирована от основной операционной системы. Благодаря этому свойству внутри запущенного контейнера без каких-либо опасений можно делать что угодно: скачивать и запускать вирусы или небезопасное ПО; посмотреть, что произойдет, если удалить корневой каталог системы и др. – выполненные действия никак не повлияют на основную операционную систему, где был запущен контейнер. Более того, чтобы откатить изменения внутри контейнера, достаточно его просто перезапустить из первоначального образа.

Ранее в качестве одного из преимуществ использования микросервисов я упомянул их масштабируемость. Как уже было сказано ранее, крупную монолитную программу можно разбить на отдельные компоненты в виде микросервисов и поместить их в контейнеры. Допустим, программа работает в публичном облаке, имеет веб-интерфейс и предполагает онлайн-взаимодействие по обработке файлов с ее помощью. Иногда может возникать ситуация, когда число одновременных пользователей достигает такого значения, что какой-либо из компонентов перестает справляться с возникшей нагрузкой. Чтобы исправить эту ситуацию, надо как-то повысить производительность. Выполняется это путем горизонтального масштабирования, то есть запускается еще несколько экземпляров контейнеров с необходимыми компонентами, между которыми можно сбалансировать нагрузку. Например, если увеличилось число обращений к сайту, то можно запустить дополнительные контейнеры с веб-сервером, который отвечает за отображение веб-интерфейса. Если же увеличился объем файлов, отправленных на обработку, то запускается еще несколько контейнеров, содержащих этот обработчик.

Опираясь на вышеуказанные свойства, я полагаю, что основными пользователями технологии контейнеризации должны стать: 1) производители программного обеспечения, нуждающиеся в быстром разворачивании сред тестирования; 2) разработчики антивирусных программ, так как изучением новых вирусов лучше всего заниматься в изолированной среде; 3) онлайн-сервисы потокового вещания, у которых число пользователей может резко увеличиваться из-за каких-либо факторов, например, онлайн-кинотеатры или платформы музыкального стриминга.

Для демонстрации эффективного использования технологии контейнеризации хочется привести следующий пример. В 2017 г. компания Adidas пришла к выводу, что их инфраструктура нуждается в модернизации. Обусловлено это было тем, что в компании разрабатывалось свое программное обеспечение. Из-за крупного объема файлов сайта обновление его компонентов происходило лишь раз в 4–6 недель. Даже процесс выделения виртуальной машины для обычного разработчика мог занимать целую неделю, что не есть хорошо. В результате совместной работы с компанией Giant Swarm, специализирующейся на внедрении и обслуживании Kubernetes-кластеров, в течение 6 месяцев был запущен сайт электронной коммерции, который на 100% работал на Kubernetes. Это позволило снизить время отклика сайта вдвое, а релизы обновлений стали происходить по 3–4 раза за день [10].

Предложения по подготовке специалистов

Первые контейнеры были реализованы на Linux подобных операционных системах. Этот фактор непосредственно влияет на те знания, которыми должен обладать пользователь для успешного использования контейнеров. Специалист должен понимать, как устроена операционная система Linux, владеть навыками работы без графического ин-

терфейса, т.е. с помощью команд в консоли терминала, способен выявлять и устранять неполадки, разбираться в принципах построения и настройки компьютерных сетей. Требуемый объем знаний довольно велик, поэтому порог «входа» при трудоустройстве в данной сфере довольно высок. Это объясняет то, что число системных администраторов и разработчиков, которые научились пользоваться ПО Docker и оркестраторами вроде Kubernetes значительно ниже, чем число обычных сисадминов.

Мне кажется, что подготовка квалифицированного специалиста должна включать несколько этапов: 1) он должен иметь высшее или среднее профессиональное образование технического профиля, чтобы иметь базовые специальные знания; 2) получить опыт работы, будучи системным администратором Linux подобных систем; 3) изучить специальные курсы по работе с технологией контейнеризации.

Заключение

Архитектура микросервисов в последние годы стала особенно популярна благодаря значительному усовершенствованию технологии контейнеризации. Мощным толчком для использования контейнеров стало размещение в открытом доступе компанией dot Cloud своей технологии Docker [8]. Ежедневно в геометрической прогрессии увеличивается число компаний, использующих контейнеры в своей рабочей инфраструктуре.

Относительная новизна технологии контейнеризации объясняет малое число специалистов, владеющих необходимым объемом знаний для ее использования. Тем не менее это заметно повысило ценность подобных сотрудников на рынке труда. Не удивительно, что размер заработной платы у специалистов, обладающих навыками использования контейнеров, заметно выше, чем у других их коллег [1]. Данный факт значительно повышает привлекательность профессии, поэтому число специалистов с каждым днем растет.

Широкое и повсеместное использование микросервисов в связке с контейнерами, развитие автоматизированных систем по управлению ими, рост числа квалифицированных специалистов – все эти факторы явно дают понять, что технология активно развивается. Контейнеризация – очень перспективное направление развития для компаний сферы IT-технологий, поэтому ее, безусловно, ожидает большое будущее.

Литература

1. Интересна карьера в DevOps? Подробно о профессии // BitDegree. URL: https://ru.bitdegree.org/rukovodstvo/devops/ (дата обращения: 05.06.2020).
2. *Колисниченко Д.Н.* Linux. От новичка к профессионалу. 6-е изд. СПб.: БХВ-Петербург, 2018. 672 с.
3. *Кречетов А.* Пять главных причин перехода в облако // Orange Business Services. URL: https://www.orange-business.com/ru/blogs/get-ready/oblachnye-vychisleniya/pyat-glavnyh-prichin-perehoda-v-oblako (дата обращения: 03.06.2020).
4. *Моуэт Э.* Использование Docker / пер. с англ. А.В. Снастина; науч. ред. А.А. Маркелов. М.: ДМК Пресс, 2017. 354 с.
5. *Робачевский А.М., Немнюгин С.А., Стесик О.Л.* Операционная система UNIX. 2-е изд. СПб.: БХВ-Петербург, 2010. 656 с.
6. *Тараненко П.* Почему Kubernetes стал настолько популярен // Vc.ru. URL: https://vc.ru/dev/107126-pochemu-kubernetes-stal-nastolko-populyaren (дата обращения: 03.06.2020).
7. Docker HUB. URL: https://hub.docker.com (date of the application: 03.06.2020).
8. Docker: Automated and Consistent Software Deployments // InfoQ. URL: https://www.infoq.com/news/2013/03/Docker/ (date of the application: 05.06.2020).

9. Microservices // Martin Fowler. URL: https://martinfowler.com/articles/microservices.html (date of the application: 03.06.2020).
10. Staying True to Its Culture, Adidas Got 40% of Its Most Impactful Systems Running on Kubernetes in a Year // Cloud Native Computing Foundation. URL: https://www.cncf.io/case-study/adidas/ (date of the application: 03.06.2020).

Literatura

1. Interesna kar'era v DevOps? Podrobno o professii // BitDegree. URL: https://ru.bitdegree.org/rukovodstvo/devops/ (data obrashcheniya: 05.06.2020).
2. *Kolisnichenko D.N.* Linux. Ot novichka k professionalu. 6-e izd. SPb.: BKHV-Peterburg, 2018. 672 s.
3. *Krechetov A.* Pyat' glavnykh prichin perekhoda v oblako // Orange Business Services. URL: https://www.orange-business.com/ru/blogs/get-ready/oblachnye-vychisleniya/pyat-glavnyh-prichin-perehoda-v-oblako (data obrashcheniya: 03.06.2020).
4. *Mouet E.* Ispol'zovanie Docker / per. s angl. A.V. Snastina; nauch. red. A.A. Markelov. M.: DMK Press, 2017. 354 s.
5. *Robachevskij A.M., Nemnyugin S.A., Stesik O.L.* Operatsionnaya sistema UNIX. 2-e izd. SPb.: BKhV-Peterburg, 2010. 656 s.
6. *Taranenko P.* Pochemu Kubernetes stal nastol'ko populyaren // Vc.ru. URL: https://vc.ru/dev/107126-pochemu-kubernetes-stal-nastolko-populyaren (data obrashcheniya: 03.06.2020).
7. Docker HUB. URL: https://hub.docker.com (date of the application: 03.06.2020).
8. Docker: Automated and Consistent Software Deployments // InfoQ. URL: https://www.infoq.com/news/2013/03/Docker/ (date of the application: 05.06.2020).
9. Microservices // Martin Fowler. URL: https://martinfowler.com/articles/microservices.html (date of the application: 03.06.2020).
10. Staying True to Its Culture, Adidas Got 40% of Its Most Impactful Systems Running on Kubernetes in a Year // Cloud Native Computing Foundation. URL: https://www.cncf.io/case-study/adidas/ (date of the application: 03.06.2020).

DOI: 10.25586/RNU.V9187.20.03.P.138 УДК 004.7

Т.Е. Черницкая, С.И. Макаренко, Д.В. Растягаев

АСПЕКТЫ АВТОМАТИЗАЦИИ ФУНКЦИЙ УПРАВЛЕНИЯ, ПРИНЯТИЯ РЕШЕНИЙ И СЕТЕВОГО ВЗАИМОДЕЙСТВИЯ В РАМКАХ ОЦЕНКИ ИНТЕРОПЕРАБЕЛЬНОСТИ СЕТЕЦЕНТРИЧЕСКИХ ИНФОРМАЦИОННО-УПРАВЛЯЮЩИХ СИСТЕМ*

В условиях перехода информационно-управляющих систем к сетецентрической архитектуре и созданию сетецентрических информационно-управляющих систем возрастает актуальность обеспечения интероперабельности в таких системах. Предложен подход к оценке аспектов автоматизации функций управления, принятия решений и сетевого взаимодействия в рамках разработки модели технической интероперабельности сетецентрических информационно-управляющих систем на основе ГОСТ Р 55062–2012. Показано, что аспекты технической интероперабельности для автоматизации процессов управления и принятия решений включают в себя параметры автоматизации принятия решений, управления, оценки адекватности принимаемых решений и вы-

* Исследование проводится в рамках проекта № 19-07-00774 Российского фонда фундаментальных исследований.