УДК 004.415.2

РАЗРАБОТКА РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ СБОРА И АНАЛИЗА ДАННЫХ НА БАЗЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ

© 2016 Ю.С. Артамонов, С.В. Востокин

Самарский национальный исследовательский университет имени академика С.П. Королёва

Статья поступила в редакцию 11.11.2016

В статье рассматривается применение микросервисной архитектуры в области вычислительных наук. Микросервисная архитектура в основном применяется для распределенных приложений в промышленных системах и не является распространенной в области научных вычислений и анализа данных. Однако ее применение в настоящее время приобретает все большую актуальность. Декомпозиция научных приложений, связанных с вычислениями и обработкой данных, на микросервисы имеет потенциальные преимущества, известные в области промышленных систем: увеличение гибкости системы, снижение трудоемкости ее сопровождения, упрощение масштабирования и другие. Декомпозиция систем на микросервисы является сложной задачей и существует несколько методик, позволяющих выделить сервисы из монолитной системы. В промышленных системах пока нет единого подхода к декомпозиции систем и в каждой системе нужно выбирать методику, основываясь на особенностях предметной области, связанности подсистем и других параметрах системы. В работе обсуждаются основные подходы к реализации систем научных вычислений на базе микросервисной архитектуры, приводятся отличия научных систем от промышленных и выделяются аспекты, позволяющие проще, чем в промышленных системах, применять микросервисную архитектуру в научных приложениях. На примере реализованной авторами системы сбора и анализа данных о загрузке вычислительного кластера «Сергей Королёв» Самарского университета демонстрируются принципы определения границ сервисов при декомпозиции первоначальной монолитной системы. Процесс миграции существующего приложения на микросервисную архитектуру для каждой системы индивидуален и не может быть выполнен за одну итерацию. В качестве примера такой миграции показано преобразование архитектуры облачного сервиса Templet Web от монолитной версии системы с 3-х слойной архитектурой к микросервисной версии системы с выделенным сервисом Service Discovery / API Gateway. При миграции сильно связанные части системы не были разбиты на микросервисы, чтобы не усложнять дизайн системы. Авторами даются практические рекомендации решения проблем, возникающих при эксплуатации системы на базе микросервисов. В заключении приводятся важнейшие отличия микросервисной версии системы от монолитной и преимущества, позволяющие решать больший, по сравнению с монолитными системами, спектр задач.

Ключевые слова: распределенные системы, вычислительная наука, микросервисы, архитектура систем, обмен сообщениями, интеграция систем.

Работа выполнена при государственной поддержке Министерства образования и науки РФ в рамках реализации мероприятий Программы повышения конкурентоспособности Самарского национального исследовательского университета имени академика С.П. Королева среди ведущих мировых научно-образовательных центров на 2013-2020 годы.
Работа частично поддержана грантом РФФИ № 15-08-05934 А.

ВВЕДЕНИЕ

Декомпозиция системы на микросервисы – подход к построению приложений в виде набора небольших сервисов, каждый из которых исполняется как отдельный процесс и связывается с другими при помощи легковесных механизмов, зачастую по протоколу HTTP. Эти сервисы строятся исходя из потребностей и особенностей предметной области так, что каждый из них инкапсулирует все аспекты какой-либо возможности системы. Все сервисы могут быть развернуты при помощи полностью автоматических механизмов системы. Выполнение этих требований позволяет реализовать каждый из сервисов, используя различные языки программирования и различные технологии хранения данных [1].

В этой работе мы будем рассматривать приложения, которые используются при Machine-To-Machine (M2M) взаимодействии [2], то есть функционируют без участия человека, получая данные, производя вычисления и передавая данные далее по конвейеру вычислений или же просто записывая их на диск. Графические приложения

Артамонов Юрий Сергеевич, ассистент кафедры информационных систем и технологий.
E-mail: artamonov@about.me
Востокин Сергей Владимирович, доктор технических наук, профессор кафедры информационных систем и технологий. E-mail: easts@mail.ru

и инструменты для проведения исследований не являются предметом обсуждения в этой статье. Почти все научные приложения, связанные с автоматизированными вычислениями, реализованы как монолитные системы, спроектированные для решения всего одной задачи, чаще всего проведением вычислений в определенной научной области. Это довольно сильно отличает их от промышленных, где одним из основных требований является совместимость с другими системами, протоколами и форматами данных.

Рассмотрим другие особенности приложений научных вычислений применительно к архитектуре систем:

- алгоритмы и приложения для анализа данных разрабатываются на разных языках, с использованием различных библиотек и инструментов, в зависимости от того, какой набор средств принят в конкретном коллективе разработчиков;

- каждое из приложений является монолитным и выполняет одну или несколько вычислительных задач в конкретной области знаний, то есть каждое приложение узко специализировано;

- приложения могут зависеть от аппаратных средств конкретного окружения, для которого было разработано приложение, от структуры сети, систем хранения и специализированного оборудования.

Промышленные монолитные приложения имеют несколько значительных недостатков [3]:

- сложное масштабирование;

- необходимость полного обновления системы при изменении незначительных деталей реализации;

- если приложение выходит из строя, то весь функционал недоступен для пользователей;

- сложно или практически невозможно изменить технологический стек приложения по мере развития.

Часть этих недостатков проявляется в вычислительных задачах и в приложениях для сбора и обработки данных, и их можно устранить, если изменить подход к проектированию даже самых небольших утилит и систем.

Некоторые проблемы монолитных систем не имеют значения для приложений, связанных с вычислениями, поскольку в них можно пренебречь некоторыми ограничениями:

- обычно не требуется транзакционность при записи результатов вычислений. Основная операция над важными данными – чтение, что позволяет не использовать транзакции вовсе, например, задействовать файлы или NoSQL СУБД [4];

- основной способ взаимодействия систем анализа данных – простой файловый обмен или HTTP запросы для получения данных;

- не требуется обеспечивать безопасность и разграничение прав доступа при взаимодействии систем;

- большая часть подсистем получает данные и возвращает результат, не сохраняя состояние [5] или сохраняя его в своем собственном формате для целей кэширования или повторного использования, не требуя специального контроля консистентности.

Все выше перечисленные факторы упрощают применение микросервисного подхода в системах, связанных с вычислениями и анализом данных.

Мы обратили внимание на микросервисную архитектуру, поскольку она хорошо себя зарекомендовала в промышленных системах, в том числе при замещении старых (legacy) систем в таких компаниях как Ebay, Netflix и Amazon [6]. Цель этой работы продемонстрировать, какие шаги нужно предпринять, чтобы преобразовать монолитную прикладную систему в систему на базе микросервисной архитектуры и какие преимущества может дать такое преобразование.

МИКРОСЕРВИСНЫЙ ПОДХОД ДЛЯ СИСТЕМ, СВЯЗАННЫХ С ВЫЧИСЛЕНИЯМИ

Микросервисная архитектура – подход к разработке приложений в виде набора небольших независимых сервисов, каждый из которых исполняется в виде отдельного самостоятельного процесса. Сервисы связываются друг с другом посредством легковесных механизмов, обычно по протоколу HTTP, что позволяет реализовать каждый из сервисов, используя различный набор библиотек и языков программирования.

Подходы, применяемые для разработки микросервисов очень близки к философии Unix. Они сформулированы в виде четырех основных правил[7]:

1. сервисы малы и достаточны для реализации одной функции системы;

2. в разработке поощряется автоматизация развертывания и тестирования;

3. отказы систем и оборудования учитываются в дизайне систем как регулярное событие жизненного цикла;

4. все сервисы масштабируемы и сочетаемы.

Правило сочетаемости и масштабируемости сервисов приобретает большое значение в задачах сбора и анализа данных, поскольку зачастую вычислительные задачи или эксперименты с данными не проектируются с расчетом на совместимость с распространенными форматами данных и протоколами взаимодействия систем. Пример промышленного приложения, построенного на базе микросервисной архитектуры приведен на рис. 1.

Архитектура такого примера приложения для онлайн покупок включает в себя несколько сервисов с конкретной областью ответственности: продукты, заказы, и т.д. При этом клиенты, взаимодействующие с системой через веб-браузер или мобильное приложение, видят перед собой одну цельную систему.

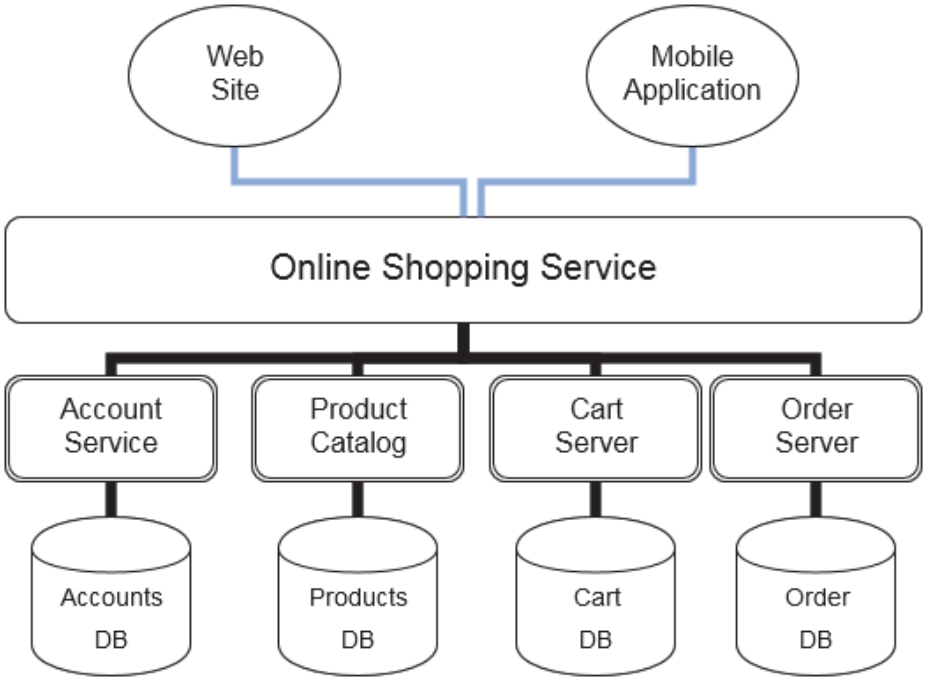

Рис. 1. Пример приложения на базе микросервисной архитектуры

Особенности реализации микросервисного подхода порождают новые проблемы, которые требуется решить при проектировании систем:

- необходимо учитывать сетевые задержки;
- сервисы должны использовать общий формат сообщений и протоколы взаимодействия;
- приложение сложнее разворачивать и тестировать;
- сложнее обеспечить отказоустойчивость и балансировку нагрузки распределенной системы;
- в сумме требуется больше вычислительных ресурсов для функционирования множества сервисов.

Недостатки микросервисной архитектуры действительно существенны для бизнес-приложений, но ими можно частично или полностью пренебречь при построении приложений распределенных вычислений.

ДЕКОМПОЗИЦИИ СИСТЕМЫ УПРАВЛЕНИЯ ВЫЧИСЛИТЕЛЬНЫМИ ЗАДАЧАМИ КЛАСТЕРА «СЕРГЕЙ КОРОЛЁВ»

Авторами разработан веб-сервис Templet Web [8] для автоматизации параллельных вычислений на суперкомпьютере «Сергей Королёв» Самарского университета, который позволяет запускать задачи на кластере, собирает статистику использования и прогнозирует загрузку узлов кластера. Основные подсистемы первой монолитной версии приложения представлены на рис. 2:

- Security– отвечает за проверку прав доступа к записям в системе на основе роли пользователя;
- Remote Connector – выполняет соединение по SSH протоколу с удаленным сервером и передает ему пакет команд;
- Task Deployer– использует Remote Connector для запуска задач на исполнение;
- Task Monitoring – проверяет статусы задач при помощи Remote Connector;
- Statistic Collector– собирает статистику использования ресурсов кластера, выполняет подключение к кластеру, используя RemoteConnector;
- Load Forecasting – на основе последней доступной статистики, собранной Statistic Collector, прогнозирует загрузку узлов суперкомпьютера.

В монолитном варианте система была очень сложна в обслуживании, и обновление отдельных подсистем вело к недоступности всей системы. Поскольку компоненты системы имеют строго разграниченную область ответственности, и мы знаем, как они связаны друг с другом, то мы можем выполнить декомпозицию этой монолитной системы на микросервисы.

Задача разбиения монолитной системы на отдельные микросервисы сама по себе является довольно сложной и не может существовать универсального алгоритма для такого изменения архитектуры. Один из тривиальных подходов к декомпозиции монолитных систем – это разделение по вариантам использования. Некоторые авторы [9] упоминают другой подход – по глаголам (verbs), например, подсистема входа (Loginsub-system), подсистема журналирования (Loggingsub-system). Еще одним распространенным подходом является декомпозиция по именам сущностей/ресурсов (nouns/resources) [10], и в этом случае в системе выделяются сервисы, владеющие некоторым ресурсом, например, сервис платежей (Paymentsservice) или сервис товаров (Goodsservice). В идеальном случае декомпозиция должна дать сервисы с очень маленькой областью ответственности.

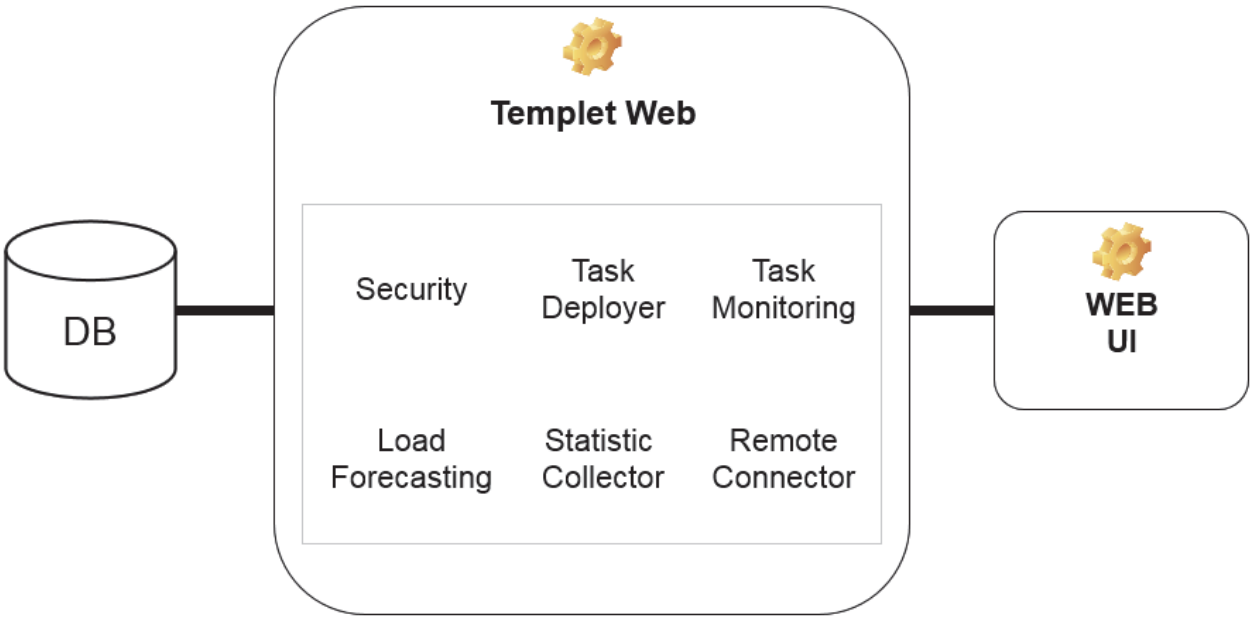

Рис. 2. Монолитная система запуска задач и мониторинга кластера

Если мы говорим о декомпозиции приложений, связанных с вычислениями, то можно также проводить декомпозицию по виду преобразования данных и по типам предоставляемых сервисом данных. В качестве основной стратегии разбиения нами была выбрана такая декомпозиция по типу предоставляемых данных. Часть системы, которая не была затронута этим типом декомпозиции, мы оставили в виде большой подсистемы. Любая декомпозиция должна происходить итеративно, поскольку велик риск выделить слишком большое количество сервисов, которые станет трудно поддерживать.

Результат декомпозиции монолитной системы представлен на рис. 3. Синим цветом обозначены связи сервисов со специальным

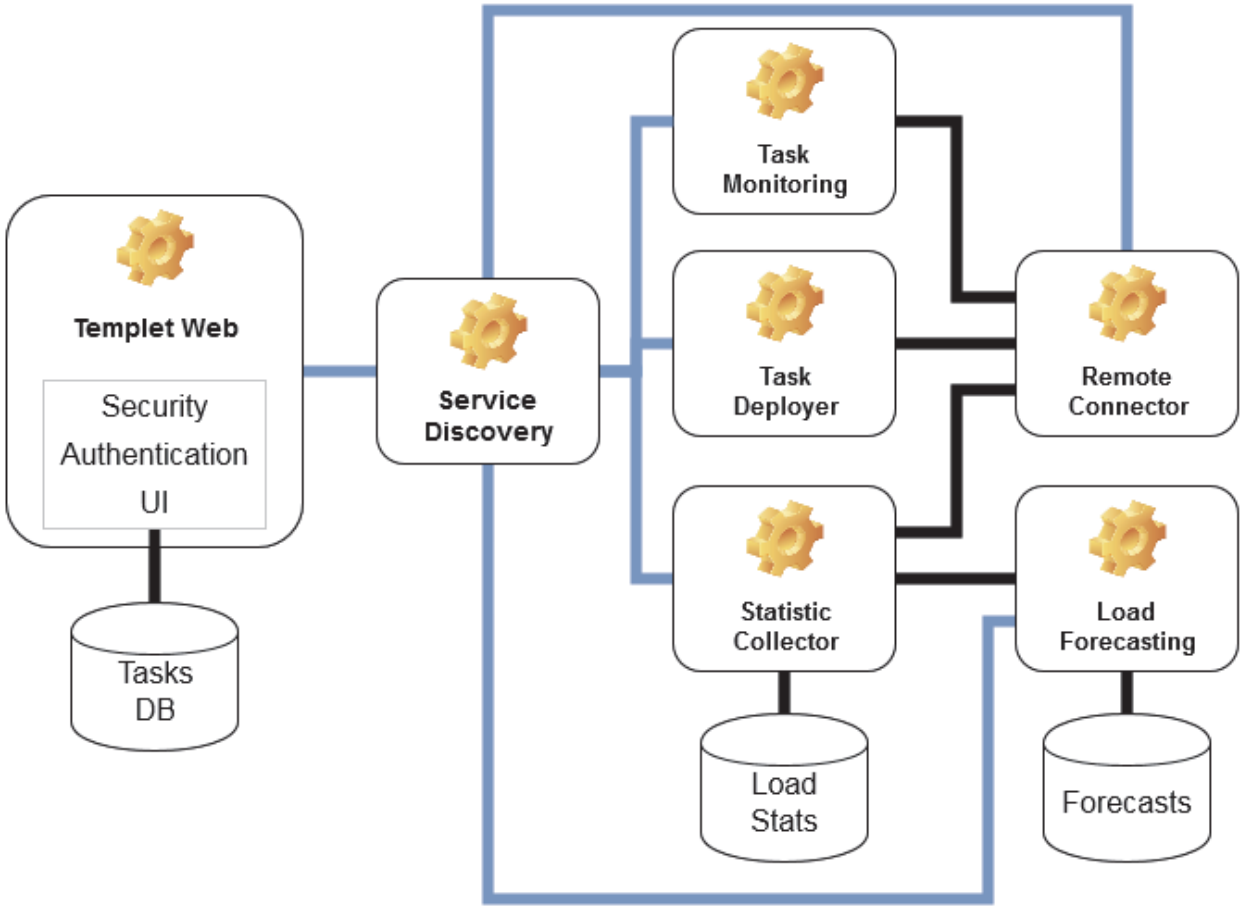

Рис. 3. Декомпозиция системы Templet на микросервисы по данным и варианту использования

сервисомServiceDiscovery. В микросервисном подходе этот сервис выполняет роль маршрутизатора и реестра сервисов, позволяя клиентам узнавать адрес сервиса в сети и затем обращаться к нему напрямую [11]. Кроме того, в нашей системе этот сервис выполняет роль APIGateway [12], через который веб-приложение, реализующее пользовательский интерфейс, обращается к выделенным микросервисам.

В этом вспомогательном сервисе заложены возможности по управлению всей системой, например, для ее масштабирования. Так, если мы захотим запускать параллельное развертывание задач на кластере, мы можем развернуть несколько экземпляров сервиса TaskDeployer, и ServiceDiscovery будет выдавать разный адрес сервиса TaskDeployer по одной из известных стратегий балансировки нагрузки, например, RoundRobin.

Следует отметить, что в получившейся распределенной системе каждый из сервисов использует свое постоянное хранилище для своих данных, а остальные сервисы могут получить эти данные, только обратившись к сервису по протоколу HTTP. В качестве транспортного протокола для обмена данными между сервисами мы выбрали REST [13] с форматом данных JSON. На практике при декомпозиции старых систем стоит выбирать те форматы, которые приняты в конкретной области знаний, и не изменять их вместе с декомпозицией системы.

Получившийся вариант распределенной системы не является конечным, и в нем могут быть выделены новые микросервисы из существующих больших подсистем для наиболее специфичных областей ответственности, таких как управление задачами, правами доступа и др.

Отдельно стоит отметить, что любая распределенная система больше подвержена проблемам и ошибкам чем монолитная, поскольку есть ненадежный транспортный уровень – сеть передачи данных. Учитывая это, в промышленных системах обязательно используются сервисы мониторинга и консолидации журналов (log-файлов), которые позволяют проводить централизованный мониторинг работы всего распределенного приложения.

Учитывая наш опыт миграции системы с монолитной архитектуры на микросервисную, мы можем дать несколько рекомендаций по эксплуатации микросервисов:

- каждый из сервисов должен предоставлять телеметрическую информацию (healthcheck) [14] о своем функционировании в виде простого HTTP адреса;

- сервисы должны использовать для маршрутизации DNS имена друг друга, а не IP адреса, чтобы повысить гибкость и масштабируемость;

- мониторинг и журналирование в приложении должно быть выполнено в виде отдельного сервиса, чтобы позволить обслуживать эту подсистему так же, как остальные;

- сервисы должны использовать стандартный для системы способ запуска и, по возможности, использовать один стек технологий, чтобы не усложнять общий дизайн системы;

- чтобы упростить развертывание и дать возможность легкой миграции сервисов на другие серверы, рекомендуется использовать одну из легковесных систем изоляции и виртуализации ресурсов: Docker, LXC и др. [15].

ЗАКЛЮЧЕНИЕ

Выработка новых подходов к разработке приложений для обработки и анализа данных имеет большое значение, поскольку в этой области существует определенный недостаток стандартизации, и применение микросервисной архитектуры с заложенными возможностями интеграции и масштабируемости может стать одним из таких стандартов. Микросервисный подход показывает себя как наиболее перспективный, и он уже зарекомендовал себя в промышленных системах, что подтверждается нашим опытом декомпозиции прикладного веб-сервиса для управления задачами кластера “Сергей Королёв”. Применение микросервисной архитектуры позволило более эффективно обслуживать и обновлять систему, затрагивая минимальный набор подсистем, а также дало возможность прозрачно масштабировать отдельные подсистемы, не изменяя логику работы клиентов, использующих эти подсистемы. Многие проблемы использования микросервисов в промышленном ПО слабо проявляются в научных приложениях, что еще больше расширяет сферу применения это архитектуры и позволяет мигрировать на нее старые (legacy) системы поэтапно.

СПИСОК ЛИТЕРАТУРЫ

1. *Fowler M., Lewis J.* Microservices URL: http://martinfowler.com/articles/microservices.html(дата обращения 05.09.2016)
2. *Namiot D., Sneps-Sneppe M.* On M2M Software Platforms // International Journal of Open Information Technologies, 2(8). 2014. С. 29-33.
3. Service-oriented e-learning platforms: From monolithic systems to flexible services / *D. Dagger, A. O’Connor, S. Lawless, E. Walsh, V. Wade* // IEEE Internet Computing. Т. 11. №. 3. 2007.С. 28-35.
4. The Database-is-the-Service Pattern for Microservice Architectures / *A. Messina, R. Rizzo, P. Storniolo, M. Tripiciano, A. Urso*// International Conference on Information Technology in Bio-and Medical Informatics. Springer International Publishing, 2016. С. 223-233.
5. A purely functional approach to packet processing/ *N. Bonelli, S. Giordano, G. Procissi, L. Abeni* // Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems. ACM, 2014. С. 219-230.

6. *Villamizar M.* Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures // 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). IEEE, 2016. С. 179-182.
7. *Namiot D., Sneps-Sneppe M.*On micro-services architecture // International Journal of Open Information Technologies 2(9). 2014. С. 24-27.
8. *Артамонов Ю.С., Востокин С.В.* Инструментальное программное обеспечение для разработки и поддержки исполнения приложений научных вычислений в кластерных системах // Вестник Самарского государственного технического университета. Сер. Физ.-мат. науки, 19:4. 2015. С. 785-798.
9. *Hassan M., Zhao W., Yang J.*Provisioningwebservicesfromresourceconstrainedmobiledevice // InCloudComputing (CLOUD), 2010 IEEE 3rd International Conference on. IEEE.С. 490–497.
10. Microservices URL: http://microservices.io/patterns/microservices.html (датаобращения05.09.2016)
11. *Balalaie A., Heydarnoori A., Jamshidi P.* Microservices migration patterns // Tech. Rep. TR-SUTCE-ASE-2015-01, Automated Software Engineering Group, Sharif University of Technology, Tehran, Iran, 2015.
12. *Newman S.* Building Microservices. O'Reilly Media, 2015. 280 с.
13. *Fernández J., Iglesias C., M. Garijo* Microservices: Lightweight service descriptions for REST architectural style // ICAART 2010 – Proceedings of the International Conference on Agents and Artificial Intelligence, Volume 1 – Artificial Intelligence, Spain, 2010.С. 186-189.
14. An architecture for self-managing microservices / *G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, A. Edmonds* // Proceedings of the 1st International Workshop on Automated Incident Management in Cloud. ACM, 2015. С. 19-24.
15. Performance Evaluation of Microservices Architectures using Containers / *M. Amaral, J. Polo, D. Carrera, M. Steinder* // Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on. IEEE, 2015. С. 27-34.

DEVELOPMENT OF DISTRIBUTED APPLICATIONS FOR DATA COLLECTION AND ANALYSIS ON THE BASIS OF A MICROSERVICE ARCHITECTURE

© 2016 Y.S. Artamonov, S.V. Vostokin

Samara National Research University named after Academician S.P. Korolev

The article discusses the use of the microservice architecture in the field of computational science. Microservice architecture is mainly used for distributed applications in industrial systems and is not quite common in the field of scientific computing and data analysis. However, its use is now becoming increasingly important. Decomposition of scientific applications related to computing and data processing to the level of microservices has potential benefits, known in the field of industrial systems: increased flexibility, reduced support complexity, simplification of application scaling, etc. Decomposition of a system to microservices is challenging and there are several techniques that enable extraction of the services from a monolithic system. There is no uniform approach to system decomposition in industrial systems; for each system you need to choose a methodology, based on the application domain features, the connectivity of subsystems and other system properties. The paper provides an overview of the main approaches to the implementation of scientific computing systems based on the microservice architecture, given the differences of scientific systems and industrial systems, and highlights the aspects that enable easier application of the microservice architecture in scientific applications in comparison with industrial systems. Based on the example of the system for data collection and analysis of the Sergey Korolev computing cluster load, we demonstrate the principles of determining the service boundaries during the initial decomposition of a monolithic system. The migration of an existing application to the microservice architecture is a unique process for each system and cannot be performed in a single iteration. As an example of such migration, we demonstrate the transformation of the Templet Web cloud service from a monolithic version of the system with 3-tier architecture to the microservice version of the system with a dedicated Service Discovery / API Gateway service. The tightly integrated parts of the system have not been broken down into microservices in order to not complicate the system design. The authors provide practical guidance solutions for problems arising from the operation of the system based on microservices. In conclusion, the major differences between the microservice version of the system and the monolithic are provided, as well as the benefits that enable the microservice architecture to address the larger range of tasks comparedtomonolithicsystems.

Keywords: distributed systems, computational science, microservices, systems architecture, messaging, system integration.

Yuriy Artamonov, Assistant Lecturer at the Information Systems and Technologies Department.
E-mail: artamonov@about.me
Sergey Vostokin, of Technics, Professor at the Information Systems and Technologies Department.
E-mail: easts@mail.ru