Методология тестирования микросервисных облачных приложений*

Д.И. Савченко, Г.И. Радченко

Южно-Уральский Государственный Университет

Микросервисный подход к организации приложений подразумевает, что архитектура облачного приложения представляется в виде набора небольших независимых сервисов, каждый из которых имеет свою зону ответственности и функциональность. Использование такого подхода подразумевает необходимость использования методов непрерывной интеграции для развертывания системы, что, в свою очередь, требует методологии и инструментария для автоматического тестирования как отдельных микросервисов так и микросервисных систем в общем. В данной статье предлагается методология тестирования микросервисных систем на всех этапах их разработки и функционирования, начиная с разработки отдельных микросервисов, заканчивая непрерывным тестированием стабильности микросервисных систем. Также, представлены подходы реализации данной методологии на основе прототипа микросервисной облачной платформы Mjolnirr.

1. Введение

Микросервисная архитектура – это паттерн проектирования облачных приложений, подразумевающий, что сложное приложение разделяется на ряд небольших независимых сервисов, взаимодействующих друг с другом посредством кроссплатформенного API. Эти сервисы могут быть развернуты независимо друг от друга, в автоматическом режиме [14]. Микросервисы могут быть рассмотрены как мета-процессы мета-ОС, они независимы и могут взаимодействовать друг с другом посредством передачи сообщений, они могут быть продублированы, приостановлены и перезапущены, а также перемещены на любой другой вычислительный узел. На данный момент существует несколько примеров реализации

Рис. 1: Монолитная и микросервисная архитектуры

*Работа выполнена при финансовой поддержке Совета по грантам Президента Российской Федерации (номер проекта МК-7524.2015.9) и Российского фонда фундаментальных научных исследований (грант № 14-07-00420)

платформ, построенных в соответствии с микросервисным подходом. Такие компании, как Netflix [15, 20] и SoundCloud [3] используют концепцию микросервисов в своей архитектуре, в то время, как Amazon [21] предлагает набор сервисов, которые могут быть также рассмотрены как примеры микросервисов.

Микросервисный архитектурный подход может быть противопоставлен с монолитным подходом к проектированию архитектуры приложения, при котором приложение представляет собой единый логический блок (см. рис. 1). Серверная часть монолитного приложения принимает HTTP-запросы, следит за бизнес-логикой приложения, взаимодействует с базой данных, а также обеспечивает отображение интерфейса пользователя для всего приложения. Изменение одного элемента такого приложения может спровоцировать пересборку и переразвертывание всего приложения, что, в свою очередь, может вызвать простои в работе пользователей приложения. Также, при необходимости масштабирования, приложение такого типа тяжело разбить на независимые компоненты, в следствии чего приходится производить дублирование всего монолита на все вычислительные узлы. Для того, чтобы избежать таких последствий, необходимо неукоснительно следовать парадигме построения модульных приложений, что, однако, требует больших затрат на проектирование [14].

При проектировании системы в соответствии с микросервисным подходом (см. рис. 1), проектируемая система делится на независимые компоненты (микросервисы), которые управляются независимо. Такое разделение полезно для высоконагруженных систем, части которых подвергаются разной нагрузке и должны масштабироваться независимо. Микросервисный подход имеет много общего с сервис-ориентированным подходом [12]. Netflix, разработчик одной из первых микросервисных платформ, называет микросервисную архитектуру "мелкомодульной COA"("a fine grained Service Oriented Architecture") [13]. Однако, микросервисная архитектура имеет и свои характерные особенности [14]:

- микросервис должен использовать механизм обмена сообщениями для коммуникации с другими микросервисами (например, HTTP);
- каждый микросервис должен реализовывать отдельный, независимый от других микросервисов, блок бизнес-логики приложения;
- микросервисы должны поддерживать возможности автоматизированного управления жизненным циклом;
- микросервисная архитектура должна иметь минимум централизованных ресурсов, архитектура микросервисной системы должна поддерживать децентрализованное управление.

Однако микросервисный подход не является универсальным решением проблемы сложности ПО. В случае микросервисной архитектуры, сложность создания и управления ПО перекладывается на инфраструктуру [2]. По этой причине возникает необходимость использования специальных средств для управления сложностью, в том числе жизненным циклом микросервисов (запуск, остановка, пауза и дублирование экземпляра микросервиса), балансировкой нагрузки и непрерывной интеграции микросервисной платформы [19]. Такой набор сервисов может быть предоставлен PaaS-облаком, которое может взять на себя функции масштабирования микросервисного приложения, поддержания жизненного цикла, а также предоставить подсистему обмена сообщениями. Кроме того, многие PaaS-решения имеют встроенный механизм мониторинга состояния отдельных компонентов приложения, а также потока сообщений между этими компонентами [17].

Целью данного исследования является разработка методологии тестирования микросервисов и реализация решения, позволяющего производить автоматизированное тестирование микросервисных систем. Для достижения этой цели необходимо решить следующие задачи:

- провести обзор существующих подходов к тестированию облачных, мультиагентных и других распределенных программных систем с целью выявления особенностей тестирования, характерных для микросервисного подхода;
- разработать методологию тестирования микросервисов;
- разработать ПО, предоставляющее поддержку тестирования микросервисов и реализующее предложенную методологию.

Данная работа посвящена обзору современных решений, касающихся тестирования распределенных систем, анализу их применимости в области микросервисов. Кроме того, будут рассмотрены основные особенности тестирования микросервисных систем и предложен подход к их тестированию. Работа построена следующим образом: в части 2 рассматриваются современные подходы к тестированию распределенных вычислительных систем. В части 3 описаны отличительные особенности тестирования микросервисных вычислительных систем и предлагается методология тестирования таких систем. В части 4 предложен подход к тестированию микросервисов на примере платформы Mjolnirr. В части 5 подведены результаты работы.

2. Обзор современных подходов к тестированию распределенных систем

2.1. Микросервисные системы

Микросервисный подход является сравнительно новым направлением в мире распределенных вычислительных систем, потому количество работ в этой области не очень велико. Основная информация о микросервисном подходе изложена в статье Джеймса Льюиса и Мартина Фаулера [14]. В этой статье описывается концепция микросервиса, а также характерные особенности микросервисной архитектуры. Тоби Клемсон в своей статье [6] уделяет особое внимание вопросам тестирования микросервисов, включая компонентное тестирование, интеграционное тестирование и тестирование контракта микросервиса. Нил Форд [7] также рассматривает микросервисные архитектуры, описывая основные идеи, касающиеся их разработки, мониторинга и тестирования. Кроме того, некоторые коммерческие компании успешно перешли на микросервисную архитектуру и описали результаты такого перехода. Например, в статье [15] рассказывается о переходе компании Netflix на микросервисную архитектуру и о том, чего это потребовало и какие плюсы принесло. В предыдущей работе описывается создание платформы распределенных вычислений под названием Mjolnirr [18]. Эта система работает с изолированными компонентами, предоставляя возможность обмениваться сообщениями в соответствии с паттерном "издатель-подписчик". На базе этой платформы были реализованы механизмы автоматического распределения нагрузки между независимыми и изолированными программными компонентами и планирования их размещения на доступных вычислительных ресурсах. [16]. На данный момент законченных коробочных продуктов, однозначно ориентированных на создание микросервисных приложений, все еще не разработано. Платформа Vamp [4], которая должна будет соответствовать всем требованиям к микросервисной платформе, выдвинутым Мартином Фаулером, на данный момент находится в стадии альфа-версии и поддерживает только JVM-ориентированные микросервисы.

Серия стандартов ISO/IEC 25010 – Software Product Quality [8] и ISO/IEC 29119 – Software Testing [9] служат отправной точкой этой работы. Тестирование в общем случае определяется как совокупность верификации и валидации [9]. В этой работе тестирование микросервисов будет рассматриваться как процесс валидации, так как невозможно применить методы статического анализа кода к уже скомпилированному микросервису в общем случае.

Как упомянуто выше, микросервисы являются подмножеством сервисов в смысле сервис-ориентированной архитектуры (SOA). Следовательно, для тестирования микросервисов возможно использовать те же приемы, что и при тестировании SOA-сервисов. Некоторые из существующих подходов используют BPEL-модель данных предметной области для тестирования на основе ограничений [11], другие предлагают тестирование на основе концепции белого ящика [1] для определения тестового покрытия сервиса.

Для качественного тестирования сервиса необходимо решить, какая качественная характеристика важна для тестируемого продукта. В терминах стандарта ISO/IEC 25011 [10] качество определяется как степень, в которой сервис может быть использован определенными пользователями для эффективного и безрискового удовлетворения своих потребностей и для достижения конкретных целей с учетом соглашения об уровне услуг (SLA) [9]. Стандарт ISO/IEC 25011 декларирует 8 качественных характеристик сервиса: Применимость сервиса (Service Suitability), Удобность сервиса (Service Usability), Безопасность сервиса (Service Security), Надежность сервиса (Service Reliability), Понятность сервиса (Service Tangibility), Отзывчивость сервиса (Service Responsiveness), Доступность сервиса (Service Adaptability) и Сопровождаемость сервиса (Service Maintainability). Все эти качественные характеристики (кроме Понятности сервиса) могут быть применены и к микросервисным вычислительным системам. Кроме того, эти качественные характеристики не регламентируют эффективность работы сервиса – эта задача целиком ложится на разработчика сервиса. Однако стабильность системы, время отклика, отсутствие рисков и соответствие SLA являются теми факторами, которые должны быть протестированы в рамках микросервисной системы.

2.2. Тестирование и распределенные системы

Микросервисные системы, мультиагентные системы и системы акторов роднит единая глобальная идея – во всех трех используется набор условно независимых сущностей, совместно работающих над решением общей задачи. Соответственно, многие подходы к тестированию таких систем могут оказаться применимы и в случае микросервисов. Например, подход, использующий онтологии для генерации тестовых случаев, применим и для микросервисных систем [22]. Подход предполагает использование онтологий для описания входных и выходных данных каждого агента (или микросервиса), а также дальнейшей автоматической генерация тестовых случаев, соответствующих входных данным и проверяющих все граничные условия входных и выходных данных.

Однако некоторые аспекты работы этих систем отличаются друг от друга. Например, в работе [23] тестирование программы, работающей в соответствии с акторным подходом, заключается в моделировании графа взаимодействий и порождения одних акторов другими. Такой анализ может быть проведен только для акторных систем и не может использоваться при тестировании микросервисов, так как микросервисы могут быть созданы только менеджером ресурсов, и одни микросервисы не могут произвольно порождать другие.

Таким образом, некоторые из существующих систем тестирования мультиагентных и акторных систем могут быть использованы для тестирования микросервисных систем, однако микросервисы конструктивно отличаются от акторов и агентов, поэтому те способы тестирования таких систем, которые учитывают характерные особенности агентов и акторов (например, возможность порождения одних акторов другими), неприменимы в тестировании микросервисных систем.

Согласно стандарту ISO/IEC 29119-4, разработка и реализация тестов и тестовых процедур при разработке ПО состоит из 6 шагов – определение набора особенностей (TD1), выделение тестовых условий (TD2), выделение тестового покрытия (TD3), выделение тестовых случаев (TD4), создание наборов тестов (TD5) и определение тестовых процедур (TD6) (см. рис. 2) [9]. Для описания процесса тестирования микросервисов будут рассмотрены этапы TD2, TD3, TD4, TD5 на TD6.

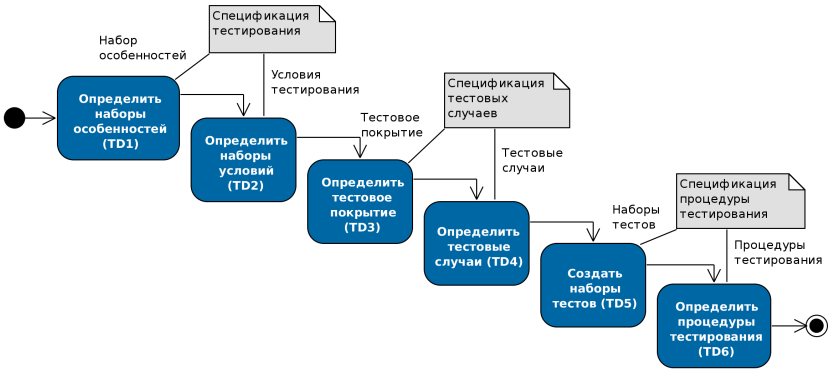

Рис. 2: ISO/IEC/IEEE29119-2 Разработка и реализация тестов

3. Методология тестирования микросервисов

В этой статье микросервис рассматривается как сервис-ориентированную сущность, которая отвечает следующим требованиям:

- Изолированность – микросервис изолирован в рамках определенного контейнера как от других микросервисов, так и от аппаратной платформы;
- Автономность – микросервис может быть развернут, уничтожен, перемещен или дублирован независимо от других микросервисов (следовательно, микросервисы не могут непосредственно ссылаться на локальные ресурсы системы, на которой развернуты);
- Микросервис должен предоставлять открытый и стандартизованный интерфейс взаимодействия для всех вариантов использования (как программный интерфейс так, возможно, и графический пользовательский интерфейс);
- Микросервисы мелькомодульны, то есть каждый микросервис несет полную ответственность за свою часть проблемной области общего приложения.

Рассмотрим основные шаги тестирования, характерные как для ПО в целом, так и для микросервисных систем, и определим те из них, которые имеют особенности в рамках микросервисной архитектуры. Под компонентным тестированием будем понимать тестирование отдельного микросервиса, под интеграционным тестированием – тестирование взаимодействия между отдельными микросервисами, под системным тестированием – тестирование функционирования системы в целом (см. таблицу 1).

3.1. Компонентное тестирование микросервисов

Функциональное компонентное тестирование является проверкой отдельного микросервиса на соответствие функциональным требованиям. Каждый микросервис имеет набор функциональных требований. Для того, чтобы произвести компонентное тестирование отдельного микросервиса, совершенно не обязательно загружать его в облачное окружение – этот тип тестирования имеет дело с микросервисом как с изолированным компонентом.

Разработка ПО в соответствии с микросервисным подходом – это процесс разработки системы как набора независимых микросервисов, взаимодействующих только путем передачи сообщений. Следовательно, каждый микросервис представляет собой независимый программный продукт, который должен быть разработан и протестирован независимо от всех остальных микросервисов. Функциональное компонентное тестирование микросервиса может быть разделена на два этапа:

- стандартное юнит-тестирование классов, входящих в микросервис (базовый уровень);

Таблица 1: Типы тестирования, имеющие особенности реализации в рамках микросервисной архитектуры

	Компонентное	Интеграционное	Системное
Функциональное	+	+	-
Нагрузочное	+	+	-
Безопасности	+	+	-

– – тип тестирования не имеет особенностей в терминах микросервисов

\+ – тип тестирования имеет особенности в терминах микросервисов

- самотестирование микросервиса, то есть проверка микросервисом своего собственного интерфейса.

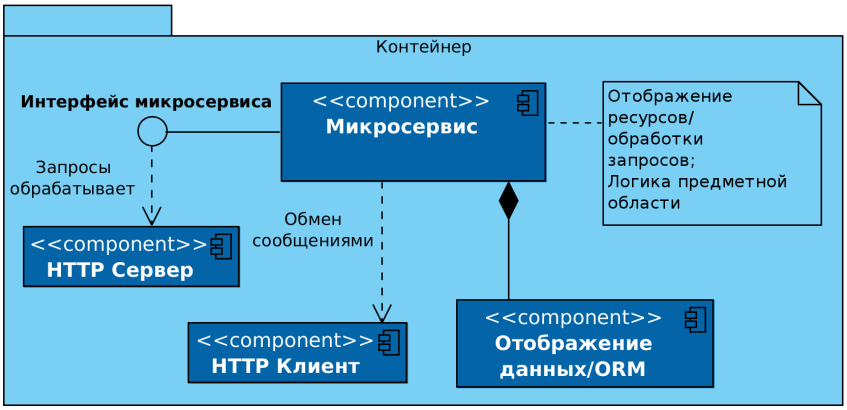

Рис. 3: Общая структура микросервиса

Микросервис является достаточно сложной сущностью, состоящей из нескольких компонентов (например, локального хранилища, веб-сервера и т. д.), упакованной в контейнер (см. рис. 3), и даже интеграция коробочного ПО в виртуальный контейнер должна сопровождаться дополнительно проверкой того, насколько корректно работает полученный микросервис. После этого необходимо проверить интерфейс собранного микросервиса на соответствие формальным требованиям к этому микросервису. На этом этапе микросервис самостоятельно тестирует собственный внешний интерфейс. Функциональное тестирование должно проводиться автоматически при каждом изменении конфигурации микросервиса.

Нагрузочное компонентное тестирование направлено на проверку работоспособности отдельного микросервиса при определенной нагрузке. Такая проверка также может осуществляться автоматически для каждого микросервиса в отдельности. Нагрузочное тестирование микросервиса позволяет не только ответить на вопрос, при какой нагрузке произойдет отказ микросервиса, но и выявить необходимое количество ресурсов для работы микросервиса.

Компонентное тестирование безопасности направлено на тестирование безопасности и изолированности отдельного микросервиса. Как и в случае с функциональным тестированием, безопасность микросервиса не заканчивается на безопасности отдельных его классов

– тестируется безопасность всего микросервиса в комплексе (например, на предмет типичных уязвимостей веб-серверов, инъекций и т.д.). Этот этап тестирования производится без участия прикладного программиста.

3.2. Интеграционное тестирование микросервисов

Интеграционное тестирование микросервисной системы включает в себя тестирование всех коммуникаций между микросервисами, то есть тестирование протоколов коммуникации, форматов обмена, разрешения взаимных блокировок, использования разделяемых ресурсов и последовательности передачи сообщений. Для реализации всех перечисленных возможностей нам необходимо отслеживать все сообщения, передающиеся от микросервиса к микросервису, и хранить пути их передачи, а также знать всех возможных получателей того или иного сообщения. Эта информация должна быть представлена в контракте микросервиса.

Функциональное интеграционное тестирование направлена на проверку корректности отправки и получения сообщений отдельным микросервисом, последовательности взаимодействия, а также оркестрации микросервисов. Зная последовательность взаимодействия, мы можем генерировать тестовые случаи автоматически, но эта информация должна быть предоставлена разработчиком микросервиса. Используя эту информацию, мы можем заранее построить граф взаимодействия и тестировать систему с его помощью.

Нагрузочное интеграционное тестирование направлено на проверку корректности работы микросервисов в условиях автоматического развертывания, дублирования и переноса их, а также оркестрация и хореография микросервисов.

Интеграционное тестирование безопасности направлено на тестирование безопасности коммуникаций между микросервисами, а также проверку на возможность перехвата сообщений извне или внутри системы кем-либо, помимо получателя сообщения. Микросервисная специфика этого уровня тестирования точно также должна учитывать особенности среды, в которой работают микросервисы.

3.3. Общая методология тестирования микросервисных систем

Используя полученные данные, возможно предложить методологию тестирования микросервисной системы, позволяющей гарантировать стабильность системы при любом изменении в отдельном микросервисе (см. рис. 4).

После того, как завершается кодирование микросервиса, его исходные коды должны быть подвергнуты юнит-тестированию для того, чтобы убедиться в соответствии написанного кода функциональным требованиям к разрабатываемому микросервису. Если юнит-тестирование прошло успешно, то микросервис упаковывается в контейнер, после чего выполняется самотестирование скомпилированного микросервиса. Если самотестирование прошло успешно, то производится нагрузочное юнит-тестирование и компонентное тестирование безопасности. Эти два этапа могут выполняться одновременно. Если они выполняются успешно, то микросервис разворачивается в тестовом окружении, где производятся функциональное интеграционное тестирование, интеграционное нагрузочное тестирование и интеграционное тестирование безопасности. Если этот этап проходит успешно, то микросервис разворачивается в рабочем окружении, где происходит непрерывное тестирование стабильности в соответствии с методологией намеренного провоцирования случайных сбоев работы системы (реализуемого, например, сервисом Chaos Monkey [5]).

4. Тестирование микросервисов в платформе Mjolnirr

В статьях [16, 18] была описана разработка прототипа микросервисной платформы Mjolnirr. Эта платформа поддерживает микросервисы, разработанные на базе JVM. Ин-

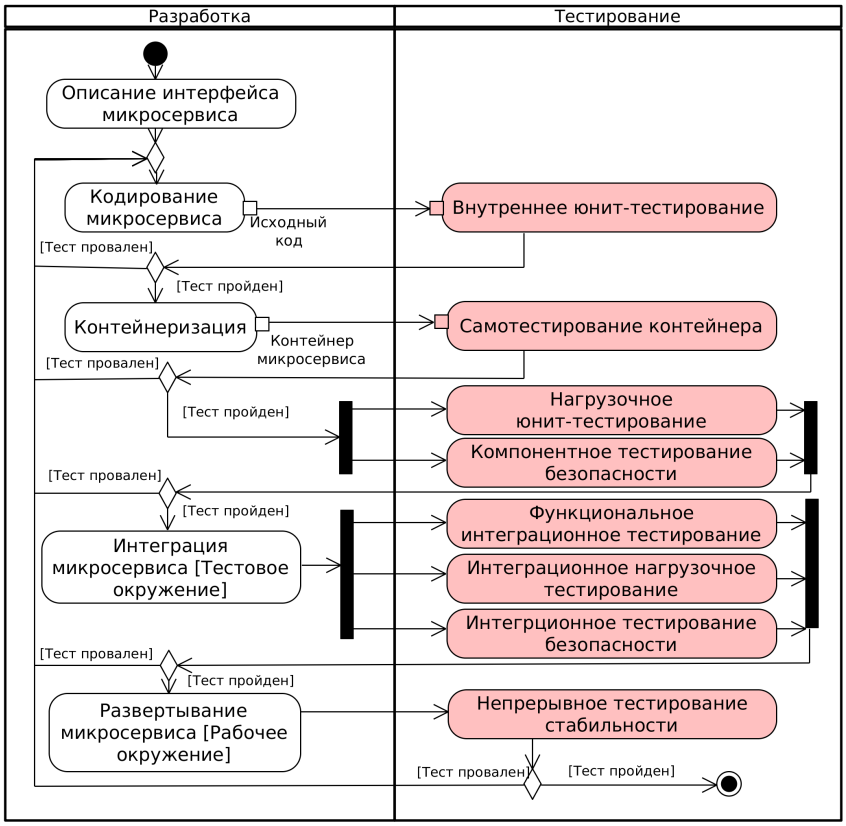

Рис. 4: Методология тестирования микросервисной системы

теграция с Mjolnirr будет первым шагом реализации системы для тестирования микросервисов. На этом этапе будут использоваться стандартные инструменты тестирования для платформы Java (например, jUnit для юнит-тестирования). Опишем то, как должен быть реализована такая система, основываясь на методологии, предложенной в разделе 3. Описание будем строить в соответствии с шагами разработки и реализации тестов и тестовых процедур, регламентированными стандартом ISO/IEC 29119-4 (см. раздел 2).

TD2 (Выделение тестовых условий) описывает то, какие аспекты микросервисов подлежит тестированию. В случае платформы Mjolnirr отдельные микросервисы и обращения к ним могут рассматриваться как тестовые случаи. Как сказано в главе 3, каждый микросервис является изолированной и независимой сущностью, и, следовательно, его тестирование может быть проведено в отдельной JVM. Микросервисы обмениваются информацией при помощи сообщений, следовательно, мы можем выделить следующие условия работы микросервисной системы: полная изоляция на компонентном уровне, обмен сообщениями и инкапсуляция логики.

TD3 (Выделение тестового покрытия) описывает атрибуты, выделенные из условий тестирования. Тестовыми условиями являются микросервисы и сообщения между ними, следовательно, элементами тестового покрытия будут методы и сообщения. Каждый метод микросервиса и каждый канал коммуникации с другими микросервисами должен быть покрыт тестами, сгенерированными автоматически либо написанными разработчиком приложения.

TD4 (Выделение тестовых случаев) описывает процесс генерации тестовых случаев. Исходя из выбранных характеристик качества и стандарта ISO/IEC 29119-4, можно решить, какие способы разработки тестов мы должны использовать. Для тестирования на компонентном уровне возможно использовать только методы, основанные на спецификации микросервиса (например, анализ граничных значений, эквивалентное разбиение, тестиро-

вание на основе вариантов использования и т. д.). Кроме того, на компонентном уровне возможно анализировать микросервис на предмет типичных уязвимостей на основе знаний об используемом ПО.

TD6 (Определение тестовых процедур) описывает процедуры проведения тестирования. В случае Mjolnirr используются как автоматические, так и написанные вручную тестовые случаи, следовательно, необходимо покрыть все элементы покрытия микросервиса. Тестовые случаи могут модифицировать состояние микросервиса и его окружение, следовательно, тестирование микросервиса должно происходить в изолированном окружении, состояние которого легко можно вернуть к первоначальному. Как показано на рисунке 5,

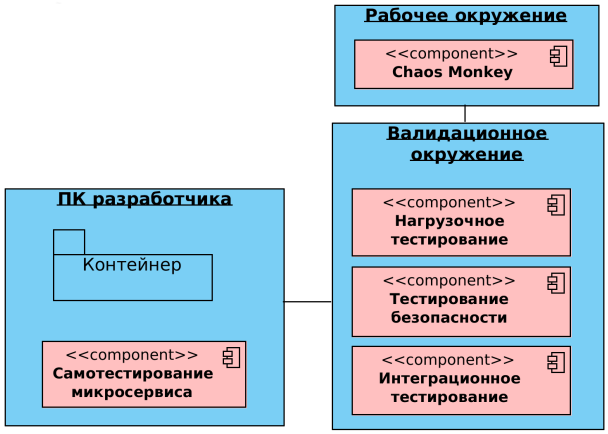

Рис. 5: Размещение компонентов системы тестирования на платформе Mjolnirr

Mjolnirr имеет 3 варианта окружения – специальный изолированный контейнер для тестирования на компонентном уровне, тестовое окружение для тестирования на интеграционном уровне и рабочее окружение для конечных пользователей. Тестовое и рабочее окружения имеют встроенный сервис Chaos Monkey [5] для постоянной проверки стабильности системы. Каждый микросервис должен последовательно проходить через эти 3 типа окружений перед тем, как быть развернутым в рабочем окружении.

Для того, чтобы использовать вышеперечисленные техники, необходимо иметь детальное описание интерфейса микросервиса, включая его входные и выходные данные, их ограничения и синтаксис. Используя эту информацию, мы можем производить тестирование каждого микросервиса по методу черного ящика в дополнение к юнит-тестам, написанным разработчиком. Пример интерфейса микросервиса представлен на рисунке 6.

В случае интеграционного тестирования известна внутренняя структура тестируемой системы. Следовательно, в дополнение в вышеописанным техникам написания тестов, возможно использовать структурные техники, например, тестирование по потоку данных и т.д. Как показано на рисунке, интерфейс микросервиса также описывает формат входных и выходных сообщений, следовательно, система может построить граф, вершинами которого будут являться микросервисы, а ребрами – пути передачи сообщений между двумя отдельными микросервисами, и тестировать систему в соответствии с построенным графом.

Тестирование на интеграционном уровне не может быть реализован на уровне контейнера, поэтому в платформе Mjolnirr эта задача решается на уровне всей платформы модулем Proxy. Proxy хранит граф вызовов и сравнивает его с графом, построенным на основе интерфейсов микросервисов. Кроме того, Proxy может использовать различные дополнительные техники интеграционного тестирования – например, Chaos Monkey и доставку случайного сообщения случайному получателю.

```
{
  "name": "test_microservice",
  "inputs": [
    {
      "type": "integer",
      "min": 0,
      "max": 10
    },
    {
      "type": "string",
      "syntax": "he.?lo"
    }
  ],
  "outputs": [
    {
      "type": "integer",
      "min": 0,
      "max": 10
    }
  ],
  "input_connections": ["test_microservice2"],
  "output_connections": ["test_microservice3"]
}
```


Рис. 6: Интерфейс микросервиса

5. Заключение

В этой статье были описаны особенности тестирования микросервисов и техники, применимые для решения этой задачи. Была представлена методология тестирования микросервисных платформ, разработанная на основе серии стандартов ISO/IEC 25010. Данная методология покрывает процесс разработки микросервисной системы, начиная с кодирования отдельного микросервиса и заканчивая непрерывным тестированием стабильности рабочего окружения. На основе разработанной методологии нами было предложено решение, обеспечивающее тестирование микросервисных систем, развернутых на базе платформы Mjolnirr.

Дальнейшим направлением развития данного проекта будет являться проектирование и реализация программных компонентов, обеспечивающих поддержку предложенной методологии тестирования микросервисных систем в виде автоматизированных сервисов. Данные сервисы обеспечат процесс автоматизированного тестирования разрабатываемых микросервисов и поддержку непрерывной интеграции микросервисных систем.

Литература

1. Bartolini C., Bertolino A., Elbaum S., Marchetti E. Whitening SOA testing Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. 161—170.

2. Brooks F.P. No Silver Bullet. Essence and accidents of software engineering IEEE computer 20(4); 1987: 10-19.

3. Calchado P. Building Products at SoundCloud—Part III: Microservices in Scala and Finagle [Online] Available at: https:developers.soundcloud.comblogbuilding-products-at-soundcloud-part-3-microservices-in-scala-and-finagle [accessed 01.05.2015].

4. VAMP [Online] Available at: https:github.commagneticiovamp [accessed 12.06.2015]

5. Chaos Monkey [Online] Available at: https:github.comNetflixSimianArmywikiChaos-Monkey [accessed 12.01.2015]

6. Clemson, T. Testing Strategies in a Microservice Architecture. 2014. [Online] Available at: http:martinfowler.comarticlesmicroservice-testing [Accessed 10.01.2015].

7. Ford N. Building Microservice Architectures. [Online] Available from: http:nealford.comdownloadsBuilding_microservice_Architectures(Neal_Ford).pdf [accessed 13.02.2015]

8. International Organization for Standardization (2014) 25010. Service Quality Requirements and Evaluation (SquaRE) – System and software Quality Model, 2014, Canada.

9. International Organization for Standardization (2014) 29119. Software Testing, 2014, Canada.

10. International Organization for Standardization (2015) 25011. Service Quality Requirements and Evaluation (SquaRE) – Service Quality Model, 2015, China.

11. Jehan S., Pill I., Wotawa F. Functional SOA Testing Based on Constraints. Proceedings of the 8th International Workshop on Automation of Software Test. 33—39.

12. Jones S. Microservices is SOA, for those who know what SOA is, 2014. [Online]. Available at: http:service-architecture.blogspot.ru201403microservices-is-soa-for-those-who-know.html. [Accessed: 17.11.2014].

13. Kant N., Tonse T. Karyon: The nucleus of a Composable Web Service. [Online] Available at: http:techblog.netflix.com201303karyon-nucleus-of-composable-web-service.html [accessed 17.11.2014].

14. Lewis J., Fowler M. Microservices. [Online] Available at: http:martinfowler.comarticlesmicroservices.html [accessed 17.11.2014].

15. Prana: A Sidecar for your Netflix PaaS based Applications and Services. [Online] Available from: http:techblog.netflix.com201411prana-sidecar-for-your-netflix-paas.html [accessed 13.02.2015]

16. Radchenko G., Mikhailov P., Savchenko D., Shamakina A., Sokolinsky L. Component-based development of cloud applications: a case study of the Mjolnirr platform. Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR '14). Article 6, 10 pages.

17. Rhoton J, Haukioja R. Cloud computing architected. [Tunbridge Wells, Kent]: Recursive Press; 2011.

18. Savchenko D., Radchenko G. Mjolnirr: A Hybrid Approach to Distributed Computing Architecture and Implementation CLOSER 2014. Proceedings of the 4th International Conference on Cloud Computing and Services Science (Barcelona, Spain 3-5 April, 2014), 2014. P. 445-450.

19. Thones J. Microservices. Software, IEEE, Volume 24 Issue 3, 2015. P. 116-116.

20. Tonse S. Microservices at Netflix. [Online] Available from: http:www.slideshare.netstonsemicroservices-at-netflix [accessed 6.02.2015].

21. Varia J. Cloud architectures. White Paper of Amazon Web Services, 2008. [Online] Available at: https:media.amazonwebservices.comAWS_Cloud_Architectures.pdf [accessed 17.11.2014].

22. Nguyen C.D., Perini A., Tonella P. Ontology-based Test Generation for MultiAgent Systems (Short Paper) Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3. P. 1315-1320.

23. Tasharofi S., Karmani R.K., Lauterburg S., Legay A., Marinov D., Agha G. TransDPOR: A Novel Dynamic Partial-Order Reduction Technique for Testing Actor Programs. In FMOODS/FORTE, volume 7273 of Lecture Notes in Computer Science, 219–234. Springer, 2012.

Microservices cloud applications testing approach

Dmitry Savchenko and Gleb Radchenko

Keywords: Cloud computing, Microservices, SOA, Fine-grained SOA

Microservice architecture is a cloud application design pattern that implies that the application is divided into a number of small independent services, each of which is responsible for implementing of a certain feature. The need for continuous integration of developed and/or modified microservices in the existing system requires a comprehensive validation of individual microservices and their co-operation as an ensemble with other microservices. In this paper, we would provide an analysis of existing methods of cloud applications testing and identify features that are specific to the microservice architecture. Based on this analysis, we will try to propose a validation methodology of the microservice systems.