Разработка облачного сервиса для тестирования микросервисных приложений*

Н.А. Ашихмин, Д.И. Савченко, Г.И. Радченко

Южно-Уральский Государственный Университет

Микросервисы - современный подход к созданию облачных приложений. Он обладает многими преимуществами, но имеет и недостатки. К ним, в частности, относится сложность тестирования. В данной статье описываются создание облачного сервиса для тестирования микросервисных приложений. Проводится анализ существующих решений и методологий, описываются варианты использования данной системы. Представлено описание архитектуры разрабатываемого облачного сервиса, аспекты реализации отдельных компонентов.

Ключевые слова: тестирование как сервис, микросервисные приложения, облачное тестирование.

1. Введение

В настоящее время, выделяют два концептуальных подхода к разработке и развертыванию облачных приложений: монолитный и микросервисный [14]. Архитектура монолитных приложений является наследием и развитием трехзвенной клиент-серверной архитектуры, в рамках которой все компоненты, реализующие бизнес-логику приложения собраны в рамках единого сервера приложений, отдельно от сервера баз данных, отвечающего за хранение данных всех бизнес-сущностей.

В рамках микросервисного подхода, серверная часть приложения разбивается на отдельные, изолированные компоненты – микросервисы, обеспечивающие прозрачный веб-доступ к своим функциональным возможностям и реализующие определенную роль в бизнес-логике приложения. Микросервисы хранят собственное состояние в отдельных независимых базах данных и взаимодействуют с другими микросервисами с помощью открытого веб-протокола (например, в соответствии со стилем REST [7]). Для функционирования такого облачного приложения необходимо обеспечить совместную работу (оркестрацию) множества микросервисов. Микросервисный подход обладает рядом существенных преимуществ по сравнению с монолитным подходом: сервисы имеют четкую границу ответственности, легко масштабируются могут быть написаны на разных языках программирования и разработаны разными командами [14]. В связи с высокой степенью масштабируемости и слабосвязанности облачных приложений, в настоящее время микросервисный подход к проектированию архитектуры ПО пользуется большой популярностью. Примерами использования данного подхода являются решения, предлагаемые такими компаниями как Amazon, eBay, Netfix и др. [15].

Автоматизированное тестирование играет очень важную роль в современном процессе разработки ПО. Оно позволяет всегда поддерживать работоспособность продукта, вносить изменения и проводить рефакторинг. Микросервисный подход вносит свои сложности в вопрос тестирования, так как нужно проверять корректное взаимодействие микросервисов в различных условиях работы. Для полного тестирования микросервисов необходимо разворачивать сложное тестирующие окружение на специальных серверах, что требует больших человеческих и материальных затрат.

* 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)

Многим компаниям сложно и дорого развертывать на своих серверах систему для тестирования своих микросервисных приложений, в стремлении снизить издержки они обращают внимания на средства облачного тестирования. Облачные приложения могут предоставить удобный сервис для качественной проверки приложений пользователей за небольшую плату.

Цель данной работы - описать архитектуру решения для тестирования микросервисных приложений.

Для достижения этой цели нужно решить следующие задачи:

1) проанализировать существующие решения и методологии в области облачного тестирования, существующие средства для тестирования распределенных систем;
2) определить требования к облачному сервису для тестирования микросервисных приложений;
3) разработать архитектуру облачного сервиса для тестирования микросервисных приложений, и на основе нее описать детали реализации отдельных модулей разрабатываемого решения.

Работа состоит из следующих частей: в части 2 описаны существующие методы тестирования, публикации, коммерческие решения и их особенности, в части 3 предложена архитектура разрабатываемого нами приложения.

2. Анализ решений в области облачного тестирования

Применение облачных технологий для тестирования приложений развивается начиная с конца 2000-х годов. Термин «Testing as a Service» (TaaS) был впервые предложен компанией Tieto в 2009 году [10]. TaaS – это модель использования внешних ресурсов (аутсорсинга), при которой задачи тестирования приложений решаются с помощью привлечения сторонней компании [16]. Облачное тестирование - это разновидность TaaS, при которой тестирование приложений реализуется на основе облачных вычислительных ресурсов [11]. Применение облачного тестирования вместо классических методов тестирования приложений обладает такими преимуществами как высокая масштабируемость тестового окружения и снижение затрат на тестирование.

Дискуссия по определению понятия и области применения облачного тестирования представлена в статье [12]. В частности, в статье затронуты такие темы, как:

1) приложения, подходящие для облачного тестирования;
2) критерии качества тестирования в облаке;
3) распределение мощностей для тестирующих приложений в многопользовательской среде;
4) онлайновые тестирующие решения для корпоративных приложений.

Облачное тестирование имеет свои риски и затраты и подходит не для всех пользователей и типов программного обеспечения. В статье [12] приводятся следующие критерии, которым должны удовлетворять приложения, чтобы они подходили для облачного тестирования:

1) тестовые случаи должны быть независимы друг от друга;
2) приложение должно разворачиваться в одной из стандартных ОС;
3) программный интерфейс приложения должен подходить для автоматизированного тестирования.

Основываясь на этом, самыми подходящими случаями для тестирования являются следующие:

1) юнит-тестирования и регрессивное тестирование;
2) большое количество автоматизированных тестов;
3) нагрузочное тестирование.

Авторами статьи [5] выделяются три случая использования облачного тестирования:

1) TaaS для разработчиков – вид тестирования, когда разработчики пишут автоматизированные тесты и используют их для непрерывной интеграции;
2) TaaS для конечных пользователей – вид тестирования, которое необходимо обычным пользователям, сюда входят эмуляторы телефонов для проверки работоспособности приложений, сервисы анализа компьютера на предмет соответствия системным требованиям приложения и т.д.;

3) TaaS для сертификационных центров – виде тестирования, используемый для сертификации приложений, например, для публикации в AppStore. Этот вид тестирования анализирует приложение на наличие дефектов и для каждого найденного дефекта предоставляет доказательства. На основе количества дефектов выставляется рейтинг тестирования. Чаще всего такое тестирование сводится к тестированию производительности и надежности.

В статье [9] авторы описывают следующие преимущества облачного тестирования:

- быстрота развертывания;
- экономия средств на содержания машин для тестирования;
- высокая производительность облачного тестирования;
- возможность проверки специфичных характеристик веб-приложений таких, как масштабируемость и избычность.

Все эти преимущества позволяют снизить затраты на тестирование на 10-20% и повысить его качество.

Аппаратное обеспечение облачных систем включает большое количество процессоров, жестких дисков, памяти и множества другого оборудования. Различное оборудование приводит к отказам, поэтому появляется задача тестирования восстановления приложений после сбоев. Вариант решения этой проблемы предложен в статье [8], в виде платформ FATE (Failure Tasting Service) и DESTINI (Declarative Testing Specifications). FATE с некоторой периодичностью имитирует отказ оборудования, тем самым тестируя систему на автоматическое восстановление. DESTINI создан для удобного описания процедуры атематического восстановления после сбоев.

В статьях [1, 13] была предложена методология тестирования микросервисных приложений,. Она базируется на трех характеристиках качества микросервисного приложения: стабильность системы, время отклика и безопасность. Исходя из этого, предлагается следующая последовательность тестирования:

1. В соответствии с особенностью языка проведение unit тестов, для отдельных компонентов микросервисного приложения для проверки корректности работы отдельных модулей.
2. После завершения первого тапа тестирования микросервис упаковывается в контейнер и происходит тестирование интерфейса сервиса, т.е. проверка правильности выполнения REST запросов к сервису.
3. Если самотестирование произошло успешно, то микросервисные разворачиваются в общем тестовом окружении, где производится функциональное интеграционное тестирование, нагрузочное интеграционное тестирование и тестирование безопасности.
4. Если все тесты прошли удачно, то приложение разворачивается на рабочих серверах. Для тестирования непрерывной интеграции случайным образом, согласно сценарию, возникают отказы оборудования, с которыми система должна справляться самостоятельно.

Одним из предлагаемых средств для тестирования распределенных и параллельных систем является D-Cloud, предложенная в статье [4]. Эта система позволяет создавать различные конфигурации виртуальных машин, с различным объемом памяти, количеством процессоров и прочее. Так же для тестирования отказов, она эмулирует сбои в работе дисков, памяти, сети и прочего, как часть тестового сценария. Другим подходом к тестированию облачных приложений является Cloud9 [6] – облачный сервис для тестирования, использующий для своей работы ресурсы Amazon EC2, что позволяет хорошо распараллеливать этапы тестирования. В статье [6] показана эффективность данной методики для регрессивных тестов.

Отдельно необходимо рассмотреть существующие коммерческие предложения для организации облачного тестирования. Компанией IBM представлена платформа «Integrated Development and Test Environment for Cloud», которая позволяет интегрировать средства тестирования в облачную инфраструктуру [17], включающая следующие сервисы:

1) IBM Testing Services for Cloud – сервис, предоставляющий виртуализацию приложений и средства тестирования производительности;
2) IBM Rational Load Testing - сервис, позволяющий производить интеграционное тестирование производительности приложения;
3) IBM Defect Analysis Starter (DAS) – сервис, позволяющий оценить слабые места в приложении, в плане производительности и безопасности;

4) IBM Defect Analysis Starter for Environment – сервис, расширяющий DAS, и позволяющий проводить оценку среды облачных приложений.

Также, существует ряд решений в области облачного тестирования от компаний Hewlett-Packard [18], SOASTA [19] предоставляющие возможности регрессивного, нагрузочного тестирования и тестирования безопасности облачных приложений.

Отдельно можно отметить систему Load Impact [20] – облачный сервис, предоставляющий возможности нагрузочного тестирования веб-сайтов. Он позволяет симулировать до 10 тысяч пользователей одновременно. Page Analyzer tool позволяет эмулировать различные виды браузеров и оценивать различные метрики такие, как скорость загрузки, время отклика, время в очереди, в них.

Интересным решением для тестирования работы приложений является New Relic [21]. New Relic — это инструмент для разработчиков, который осуществляет мониторинг рабочих приложений и предоставляет подробные данные об их производительности и надежности. Таким образом, он позволяет быстрее выявлять и диагностировать возникающие проблемы с производительностью, а также предоставляет в ваше распоряжение данные, необходимые для решения этих проблем.

Инструмент New Relic отслеживает время загрузки и пропускную способность веб-транзакций как через сервер, так и через браузеры пользователей. Он позволяет увидеть, как долго вы пользуетесь базой данных, анализирует медленные запросы и веб-запросы, обеспечивает мониторинг бесперебойной работы и отображает соответствующие предупреждения, отслеживает исключения в приложениях, а также поддерживает множество других функций [2].

Анализ источников показывает, что на данный момент существует большой интерес к области облачного тестирования. Авторами платформ TaaS предоставляются различные подходы к тестированию, и пользователю трудно выбрать какой-то конкретный сервис, обеспечивающий наиболее полное тестирование для облачного приложения. К тому же многие решения не ориентированы на тестирование микросервисных приложений. Поэтому существует необходимость создания платформы, которая обеспечивала бы поддержку различных методов тестирования на различных этапах разработки, внедрения и поддержки микросервисных облачных приложений. Это даст пользователю возможность выбирать те, методы тестирования, которые необходимы для его продукта, что обеспечит качественное тестирование приложений.

3. Облачный сервис для тестирования микросервисных приложений

Для решения задачи тестирования микросервисных облачных приложений, нами предлагается решение на основе веб-сервиса, обеспечивающего возможность организации автоматизированного тестирования микросервисных облачных приложений на всех этапах разработки, начиная от тестирования исходных кодов отдельных микросервисов, заканчивая автоматизированным тестированием стабильности. Пользователь системы сможет применить интересующие его методы тестирования для собственного облачного приложения, указав методы доступа к тем или иным ресурсам микросервисного приложения (репозитории исходного кода, репозитории виртуальных машин, адреса облачных серверов и др.).

3.1 Основные определения

Введем определения основных сущностей нашей системы.

Проектом тестирования будем называть логически связанный пакет целей тестирования, имеющий общие настройки. Проект тестирования конфигурируется пользователем для достижения всестороннего (в соответствии с пониманием пользователя) тестирования определенного микросервисного приложения.

Цель тестирования – это сущность, представляющая процесс тестирования компонента либо части микросервисного приложения. Цель тестирования представляет собой последовательность определенных действий для тестирования приложения и\или его части. Цель тестирования характеризуется не только последовательностью шагов (методов тестирования), но и совокупностью типов, к которым относится тестируемое приложение или его часть.

Типом приложения мы будем называть определенную характеристику, определяющую специфику тестируемого приложения или компонента. Например, в виде типа приложения может выступать язык, на котором написано приложение (Java, C#, Python), используемый фреймворк (Django, ASP.NET, Ruby on Rails и т.д.), тип базы данных (SQL, NoSQL), особенности архитектуры приложения (веб-сервис, веб-сайт) и т.д. С каждым типом приложения связан определенный набор возможных методов тестирования.

Классификатор – коллекция всех зарегистрированных в системе типов приложений.

Метод тестирования – это атомарный шаг тестирования, направленный на проверку определенного аспекта системы (например, функциональное или нагрузочное тестирование будет представлять собой отдельный метод).

3.2 Общее описание системы

Облачный сервис для тестирования микросервисных приложений позволит автоматизировать и вывести в облако следующие виды тестирования:

- компонентное тестирование классов, входящих в микросервис на основе тестирования исходного кода;
- самотестирование микросервисов после их сборки, то есть проверка микросервисом своего собственного интерфейса;
- нагрузочное компонентное тестирование, то есть проверка работоспособности отдельных микросервисов в условиях пиковых нагрузок;
- компонентное тестирование безопасности, направленное на проверку безопасности и изолированности отдельных микросервисов;
- функциональное интеграционное тестирование, направленное на проверку правильности взаимодействия микросервисов;
- нагрузочное интеграционное тестирование, направленное на проверку корректности взаимодействия микросервисов в условиях пиковых нагрузок на облачное приложение.
- интеграционное тестирование безопасности, направленное на безопасность коммуникаций между микросервисами, а также на возможность перехвата сообщений извне или внутри системы кем-либо в отличии от получателя сообщений.

Каждый отдельный метод тестирования будет реализован в виде независимого сервиса тестирования. Таким образом, система будет поддерживать интеграцию существующих облачных методов тестирования, а также расширение функционала за счет возможности интеграции новых методов тестирования.

3.3 Варианты использования системы

Система представляется в виде веб-приложения, позволяющего тестировать микросервисные приложения пользователей в автоматическом режиме. Она позволяет настраивать, создавать методы тестирования и предлагать их пользователям, в соответствии с указанными ими типами приложения.

Определим основных актеров, взаимодействующих с системой.

Тестировщик – это пользователь системы, который использует сервис тестирования для тестирования какого-либо облачного микросервисного приложения.

Разработчик методов тестирования – это пользователь системы, который создает и добавляет в сервис тестирования новые методы тестирования. Он может создать свой метод тестирования в виде отдельного сервиса, принимающего параметры тестируемого приложения (адреса конечных точек, репозитории исходного кода и др.), а на выход отдает результат пройденных тестов.

Микросервисное приложение – это тестируемая система.

Диаграмма сервиса автоматизированного тестирования микросервисных приложений приведена на рис. 1. В системе существует 3 актера: тестировщик, разработчик методов тестирования, микросервисное приложение.

Разработчик методов тестирования создает методы тестирования и связывает их с типами приложений.

Тестировщик может:

– создать проект тестирования– тестировщик создает проект, который будет содержать все цели тестирования;

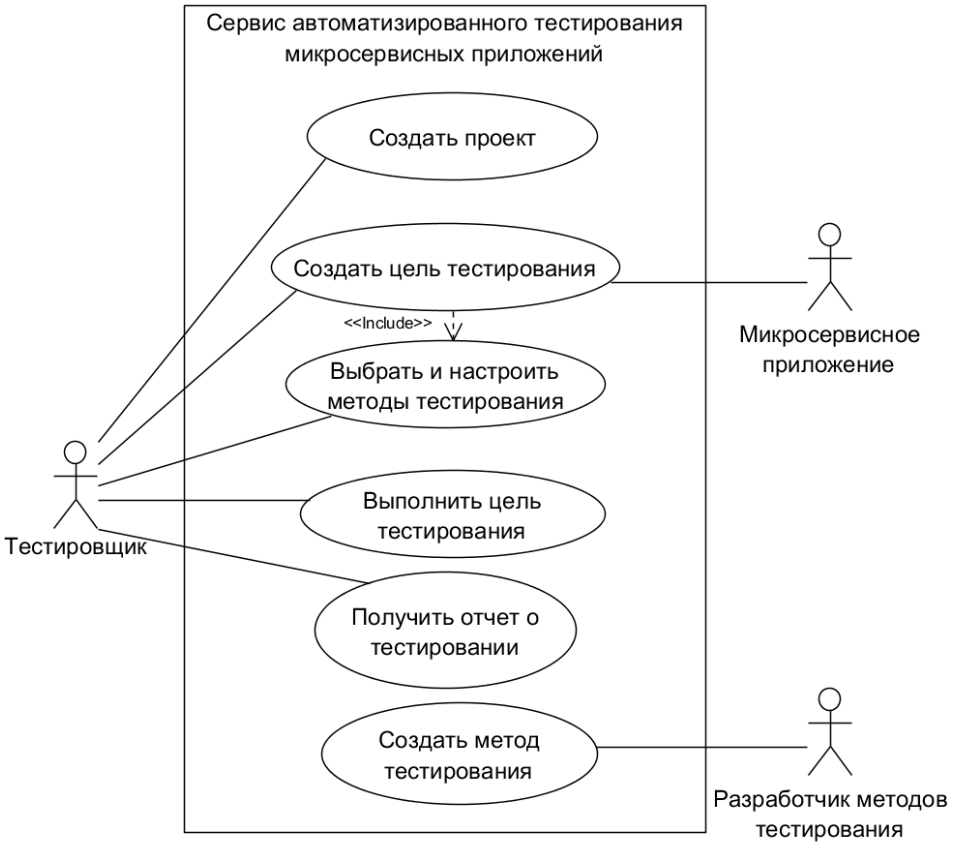

Рис. 1 Диаграмма вариантов использования сервиса для тестирования микросервисных приложений

– создать цель тестирования – тестировщик задает цель тестирования, где указывает название, описание цели, и типы приложения; целью может быть тестирование как отдельных компонентов микросервисного приложения, так и некоторых аспектов функциональности (тестирование нагрузки на жесткие диски, нагрузки на сеть, и т.д.);
– выбрать и настроить методы тестирования – тестировщик выбирает из предложенных методов тестирования те, которые ему нужны, и указывает необходимые входные данные для них.
– выполнить цель тестирования – тестировщик запускает цель тестирования, либо назначает ее выполнение на какое-либо время, это может быть, как выполнение последовательности тестов, так и режим постоянного мониторинга и тестирования работающего приложения, в зависимости от указанных методов тестирования в цели тестирования;
– получать отчет о тестировании – тестировщик получает от системы отчет о результатах работы цели тестирования.

3.4 Архитектура системы

Архитектура предлагаемого решения изображена на рис. 2. Приложение предлагается как облачный веб-сервис с микросервисной архитектурой. Оно состоит из 3 основных микросервисов и подключаемых методов тестирования, реализуемых в виде отдельных микросервисов. Рассмотрим архитектуру подробней.

Микросервис аутентификации пользователей – микросервис, отвечающий за идентификацию пользователей. Он позволяет аутентифицировать пользователя с помощью логина и пароля, открытого SSH ключа, а также с помощью аккаунтов из социальных сетей.

Микросервис проектов пользователя – микросервис, реализующий взаимодействие с пользователем. Он агрегирует проекты пользователей и предоставляет внешние интерфейсы как в виде веб-интерфейса, так и в виде программного интерфейса на основе REST запросов. Сервис хранит данные о проектах, целях и конфигурациях методов тестирования всех пользователей. Для идентификации тестировщиков сервис взаимодействует с микросервисом аутентификации.

Микросервис методов тестирования и типов приложений – микросервис, ответственный за хранение данных о типах приложения и зарегистрированных в системе методах тестирования. Включает в себя классификатор типов приложений и сервис методов тестирования. При запросах по типу приложения выдает подходящие методы тестирования и требуемых параметры для них.

Микросервис метода тестирования – отдельный микросервис, принимающий на вход конфигурацию метода тестирования и возвращающий результат. Методы тестирования интегрируются в систему с помощью регистрации в микросервисе методов тестирования и типов приложений.

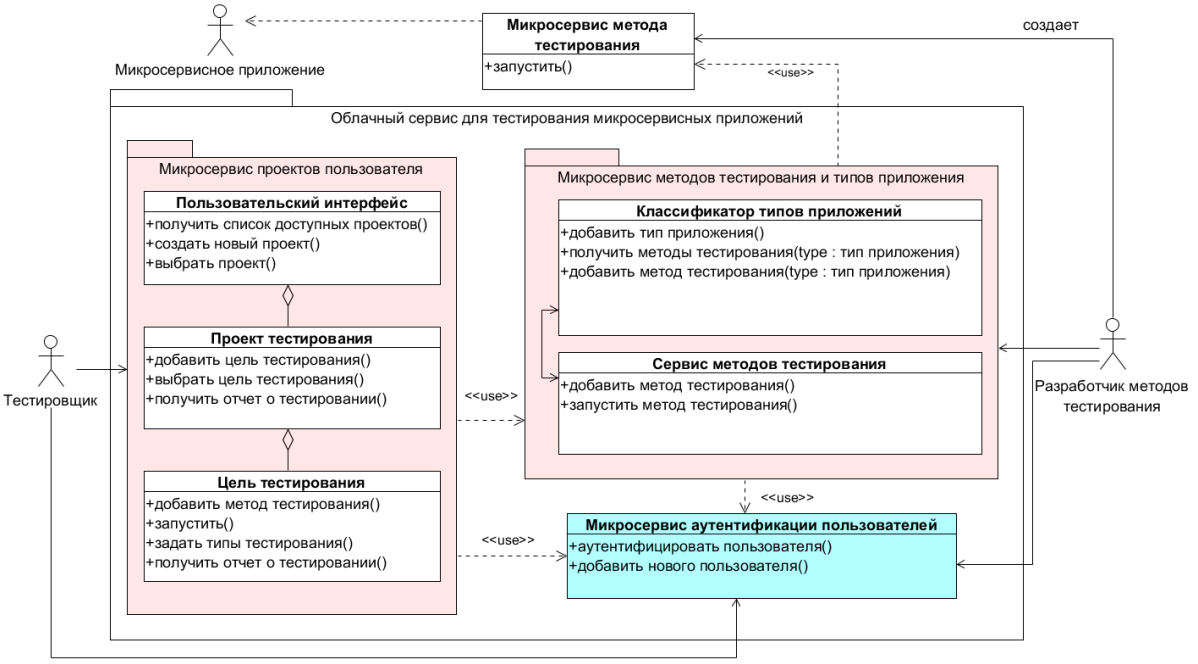

Рис. 2 Архитектура сервиса для тестирования микросервисных приложений

Определим процесс создания проекта тестирования с настройкой в нем целей тестирования представлен (см. рис. 3):

1) тестировщик аутентифицируется в системе с помощью микросервиса аутентификации пользователей, вводя свой логин и пароль;
2) дальше он создает новый проект тестирования, указывая его название;
3) после этого тестировщик создает цель тестирования указывая ее имя и описание;
4) существующие в системе типы приложения запрашиваются у микросервиса методов тестирования и типов приложения и возвращаются микросервису проектов пользователя;
5) тестировщик указывает типы приложения, которые подходят для его цели тестирования;
6) на основе выбранных типов микросервис проектов пользователя обращается к микросервису методов тестирования и типов приложения и запрашивает подходящие методы тестирования для данных типов приложения;
7) пользователь выбирает и специфицирует все методы тестирования, указывая необходимые параметры для них, такие как путь к ресурсам, файлы исходного кода, ограничение по времени работы и т.д.

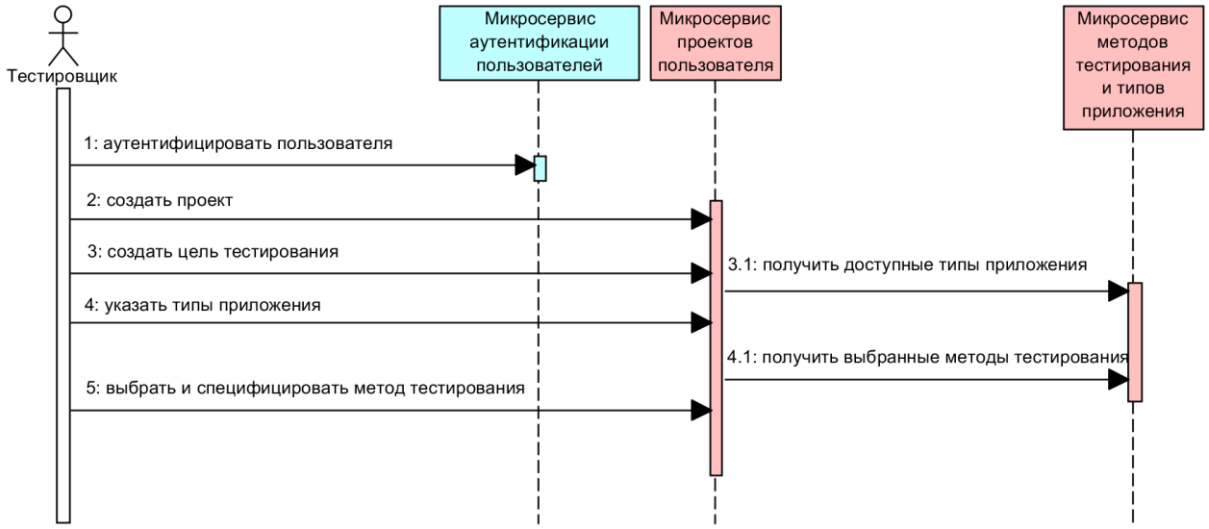

Рис. 3 Диаграмма последовательности сценария создания проекта тестирования с настройкой целей тестирования

Также, опишем стандартный алгоритм действия тестировщика по тестированию приложения (см. рис. 4):

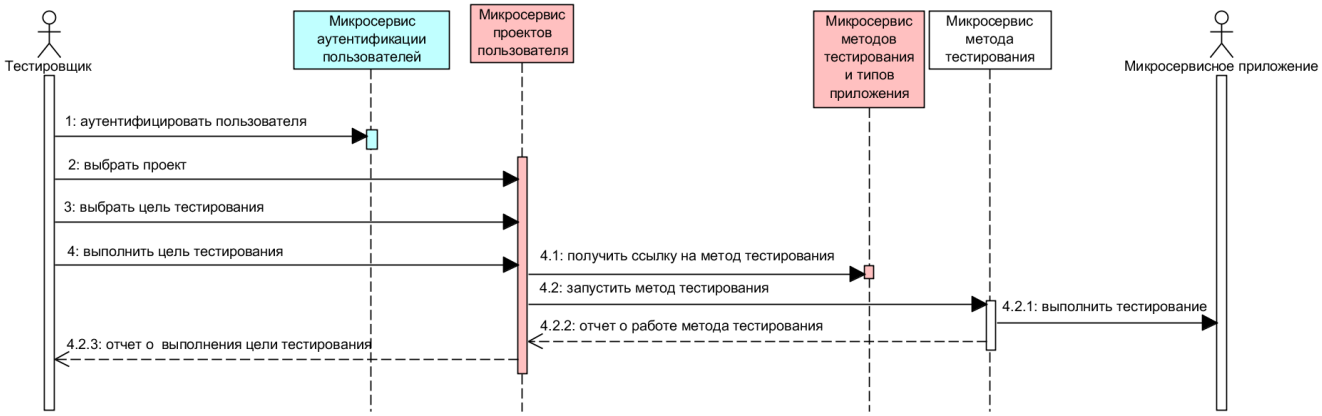

Рис. 4 Диаграмма последовательности сценария тестирования микросервисного приложения

1) тестировщик аутентифицируется в системе с помощью микросервиса аутентификации пользователей, вводя свой логин и пароль;
2) дальше он запрашивает необходимый проект от микросервиса проектов пользователя и выбирает нужную цель тестирования;
3) после чего тестировщик передает команду на выполнения цели тестирования микросервису проектов пользователя;
4) этот микросервис обращается к микросервису методов тестирования и запрашивает ссылки на методы тестирования, указанные в цели тестирования;
5) получив данные, микросервис проектов пользователя запускает микросервисы методов тестирования в порядке, указанном в цели тестирования, передавая им необходимые данные;
6) микросервис метода тестирования, извлекает полученные данные и либо разворачивает у себя в окружении тестируемое приложение, либо подключается к уже запущенному, либо выполняет какое-либо другое действие (к примеру, просматривает файлы, для статического анализа исходного кода) в зависимости от предназначения метода тестирования. После этого он производит тестирование и возвращает отчет о результатах своей работы микросервису проектов пользователя;
7) тот же в свою очередь формирует общий отчет от всех методов и передает его тестировщику.

3.5 Интерфейсы системы

Для интеграции системы тестирования микросервисов с системами непрерывной интеграции и другими внешними сервисами в ней предусмотрен как пользовательский графический интерфейс, так и JSON REST API. При помощи графического интерфейс происходит конфигурация проекта, его целей, методов; REST API позволяет запускать цели тестирования, передавая туда необходимые параметры, а также запрашивать статус тестирования, лог исполнения и ошибок. Все действия, инициируемые при помощи API, возможно исполнять также и при помощи графического интерфейса.

На рис. 5 представлен интерфейс конфигурации новой цели тестирования. Помимо очевидных полей, от пользователя требуется предоставить информацию о том, к каким типам относится приложение. После этого система может предложить ряд методов тестирования, характерных для объединения указанных типов приложений.

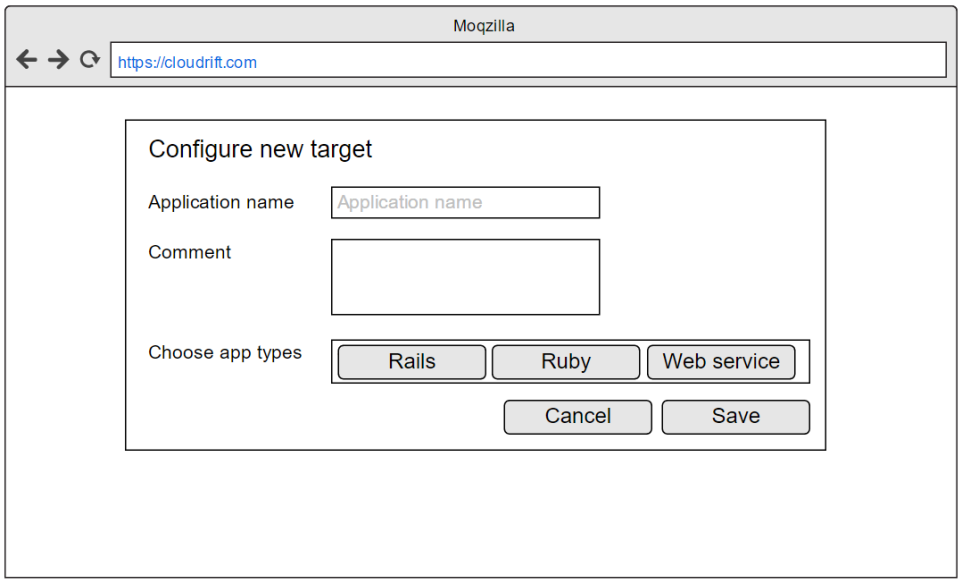

Рис. 5 Конфигурация новой цели тестирования

После создания цели можно задать ряд параметров, не изменяющихся от запуска к запуску, а также сконфигурировать последовательность и логику работы методов тестирования (должна ли ошибка в одном из методов приводить к аварийному завершению всей цели). Сконфигурированную цель можно запустить при помощи POST-запроса, пример которого можно увидеть на рис. 6.

```
POST /api/v1/targets/my_netty_target

{
  source: "git@github.com:skayred/netty-upload-example.git",
  methods: "all",
  framework: "maven",
  endpoint: "https://netty-upload.herokuapp.com"
}
```

Рис. 6 Пример JSON-запроса на запуск цели тестирования

Поля JSON-запроса содержат информацию, требуемую методами тестирования, входящими в цель. В данном запросе предоставляется расположение исходного кода приложения, список методов (в данном случае запускаются все методы тестирования), фреймворк (требуется методом юнит-тестирования), а также расположение развернутого веб-сервиса для запуска нагрузочных тестов, тестов безопасности, а также, возможно, функциональных тестов, использующих внешний интерфейс сервиса и реализованных пользователем.

3.6 Реализация отдельных методов тестирования

Изначально разрабатываемый сервис поддерживает только микросервисные приложения на Java. В дальнейшем, при развитии проекта будет добавляться поддержка новых языков и фреймворков, таких как Django (Python), Ruby on Rails и ASP.NET. В качестве средства контейнеризации используется Docker [22]. Все методы тестирования, реализовываются, как отдельные микросервисы.

Наш сервис изначально позволяет производить функциональное тестирование, тестирование безопасности и нагрузочное тестирования.

3.5.1 Функциональное тестирование

Функциональное тестирование предоставляется в двух видах, с развертыванием тестового окружения на наших серверах и с подключением к уже развернутым сервисам в интернете, так называемые дымовые тесты [3]. Первый тип тестирования позволяет убедиться в корректности работы программы, второй же позволяет убедиться, что на production серверах все работает должным образом.

Функциональное тестирование с развертыванием тестового окружения реализуется 3 методами тестирования.

Метод юнит тестирования отдельных модулей – метод, позволяющий производить юнит тестирование компонентов сервиса. Он принимает на вход скомпилированное приложение и Junit тесты. Производит обычное юнит-тестирование без разворачивания приложения в отельный контейнер.

Метод тестирование отдельных сервисов – метод, позволяющий убедиться в корректности работы отдельных сервисов. Он принимает на вход Docker контейнер с тестируемым сервисом и скрипты для тестирования. Для эмуляции взаимодействующих сервисов используется Mountebank, сервис, позволяющий легко создавать http заглушки [23].

Метод сквозного тестирования – метод, тестирующий взаимодействие нескольких сервисов. На вход принимаются Docker контейнеры с отдельными микросервисами, скрипты тестовых сценариев и необходимые настройки, для развертывания всего приложения. При запуске все сервисы разворачиваются, между ними устанавливается коммуникация и запускаются тестовые сценарии.

Функциональное тестирование с подключением к уже развернутым сервисам, реализуется *методом дымового тестирования*, принимающего на вход скрипты тестирования и адреса тестируемых сервисов.

3.5.2 Нагрузочное тестирование.

Метод нагрузочного тестирования позволяет проверить работу приложения под нагрузкой. Для тестирования используется два отдельных компонента – микросервис генератора нагрузки, находящийся снаружи и монитор ресурсов, разворачиваемый на хосте рядом с сервисом, тестируемым в данный момент.

Генератор нагрузки требует описание интерфейса тестируемого сервиса, а именно форматы запросов, а также задания длительности тестирования и количества запросов.

Монитор ресурсов производит мониторинг использования ресурсов хоста микросервиса. Информацию о степени загрузки сервиса, времени ответа на запросы и случаях нетипичного поведения (нелинейных скачков загрузки, отказов и т.д.) монитор передает методу нагрузочного тестирования.

В качестве входных параметров метод нагрузочного тестирования принимает следующие параметры:

- местонахождение сервиса, его URL;
- пути, по которым доступна тестируемая функциональность (необходима для мониторов);
- структура корректных запросов для каждого монитора;
- описание интерфейсов тестируемым сервисов для генератора (адреса, порты, типы корректных запросов);
- ограничения по времени отклика;

– установка агента-монитора на сервер.

3.5.3 Тестирование безопасности

Метод тестирования безопасности позволяет проводить проверку микросервисного приложение на наличие SQL инъекций и тестировать на возможность проникновения. Этот метод так же содержит два модуля: генератор запросов и внутренний агент.

Генератор необходим для генерации заведомо ошибочных запросов, а также попыток внедрить SQL инъекцию. Роль агента - фиксация изменений в базе данных и установление факта взлома.

Набор информации, необходимый для тестирования безопасности, похож на набор из нагрузочного тестирования и состоит из следующих параметров:

– местонахождение сервиса, его URL;
– пути, по которым доступна тестируемая функциональность (необходима для агентов);
– описание интерфейсов тестируемым сервисов для генератора (адреса, порты, типы корректных запросов);
– описание тестируемых SQL инъекций и эксплойтов;
– установка агента-монитора на сервер.

4. Заключение

В этой статье были описаны существующие решения и методологии в области облачного тестирования, имеющиеся средства для тестирования распределенных систем. Были определены требования к разрабатываемой платформе. Разработана архитектура тестирующего приложения и описана реализация отдельных методов тестирование нами приложения.

Дальнейшим направлением развития будет выпуск данной системы, и разработка новых методов тестирования, позволяющих в дальнейшем производить облачное тестирование не только микросервисных, а также десктопных, мобильных и прочих приложений.

Литература

1. Савченко Д.И., Радченко Г.И. Методология тестирования микросервисных облачных приложений // Суперкомпьютерные дни в России: Труды международной конференции (28-29 сентября 2015 г., г. Москва). 2015. С. 245-256.
2. Веб-приложение .NET в службе приложений Azure с управлением производительностью приложения от New Relic. URL: https://azure.microsoft.com/ru-ru/documentation/articles/store-new-relic-web-sites-dotnet-application-performance-management/ (дата обращения: 25.01.2016).
3. Дымовое тестирование или Smoke Testing. URL: http://www.protesting.ru/testing/types/smoke.html (дата обращения: 30.01.2015).
4. Banzai T. D-Cloud: Design of a Software Testing Environment for Reliable Distributed Systems Using Cloud Computing Technology // 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (Melbourne, Australia, 17-20 May 2010), 2010, P. 631-636.
5. Candea G., Bucur S., Zamfir C. Automated software testing as a service // Proceedings of the 1st ACM Symposium on Cloud Computing (Indianapolis, USA, 10-11 June 2010), SoCC '10. 2010, P. 155–160.
6. Ciortea L. Cloud9 : A Software Testing Service // ACM SIGOPS Operating Systems Review. 2010. Vol. 4, P. 5–10.
7. Fielding R.T., Taylor R.N. Principled design of the modern web architecture // ACM Transactions on Internet Technology (TOIT). 2002. Vol. 2, P. 115-150.
8. Haryadi S.G. FATE and DESTINI: a framework for cloud recovery testing // Proceedings of the 8th USENIX conference on Networked systems design and implementation (San Jose, USA, 25-27 April 2011). 2011. P. 238–252.

9. Janani V., Krishnamoorty K. Cloud Testing as a Service (CTaaS) – Analysis , Design and Implementation // International Journal of Applied Engineering Research (IJAER). 2015. Vol. 10, No. 12, P. 30393–30405.
10. Pardeshi S.N. Study on Testing as a Service on Cloud // International Journal of Advanced Computer Research. 2013. No. 1, P. 1–4.
11. Pardeshi S.N., Choure V. Testing as a Service on Cloud : A Review // International Journal on Recent and Innovation Trends in Computing and Communication. 2014. Vol. 2, P. 188–193.
12. Riungu L.M., Taipale O., Smolander K. Research Issues for Software Testing in the Cloud // Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on (Indianapolis, USA, 25 November – 3 December 2010). 2010. P. 557–564
13. Savchenko D.I., Radchenko G.I., Taipale O. Microservices validation : Mjolnirr platform case study // Proceedings of the 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO'2015 (Opatija, Croatia, 25 May - 29 May 2015). 2015. P. 248–253.
14. Microservices: a definition of this new architectural term. URL: http://martinfowler.com/articles/microservices.html (дата обращения: 25.12.2015).
15. Pattern: Microservices Architecture. URL: http://microservices.io/patterns/microservices.html (дата обращения: 25.12.2015).
16. Testing as a Service (TaaS) definition. URL: http://searchcloudapplications.techtarget.com/definition/Testing-as-a-Service-TaaS (дата обращения: 13.01.2016).
17. IBM Extends Development and Test to the IBM Cloud. URL: http://www-03.ibm.com/press/us/en/pressrelease/29685.wss (дата обращения: 01.12.2015).
18. Test and Develop with HP Cloud. URL: http://www.hpcloud.com/solutions/test-develop (дата обращения: 01.12.2015).
19. SOASTA. URL: http://www.soasta.com/ (дата обращения: 01.12.2015).
20. Load Impact. URL: https://loadimpact.com/ (дата обращения: 01.12.2015).
21. New Relic. URL: http://newrelic.com/ (дата обращения: 25.01.2016).
22. Docker - Build, Ship, and Run Any App, Anywhere. URL: https://www.docker.com/ (дата обращения: 25.01.2016).
23. Mountebank - over the wire test doubles. URL: http://www.mbtest.org/ (дата обращения: 30.01.2015).

The development of cloud service for testing microservice applications

N.A. Ashikhmin, D.I. Savchenko, G. I. Radchenko

South Ural State University

Microservices is a modern approach to building cloud applications. It has many advantages, but it also has shortcomings. These include, in particular, the complexity of testing. This article describes development of a cloud service for microservice applications testing. We provide analysis of existing solutions and methodologies. On the basis of the analysis, we define the main requirements and use cases for the system. We describe the architecture of developed cloud service for microservice applications testing and the implementation details of its individual components.

Keywords: testing as a service, microservice applications, cloud testing.

References

1. Savchenko, D., Radchenko, G. Microservices cloud applications testing approach // 1st Russian Conference on Supercomputing Days 2015, RuSCDays 2015 (Moscow, Russian Federation, 28-29 September 2015). CEUR Workshop Proceedings. 2015. Vol. 1482, P. 245-256.
2. Veb-prilozhenie .NET v sluzhbe prilozheniy Azure s upravleniem proizvoditel'nost'yu prilozheniya ot New Relic [.NET Web application in the service of Azure applications with performance management applications from New Relic]. URL:https://azure.microsoft.com/ru-ru/documentation/articles/store-new-relic-web-sites-dotnet-application-performance-management/ (accessed: 01.25.2016).
3. Dymovoe testirovanie ili Smoke Testing [Flue testing or Smoke Testing]. URL: http://www.protesting.ru/testing/types/smoke.html (accessed: 01.30.2015).
4. Banzai T. D-Cloud: Design of a Software Testing Environment for Reliable Distributed Systems Using Cloud Computing Technology // Cluster, Cloud and Grid Computing (Melbourne, Australia, 17-20 May 2010). 10th IEEE/ACM International Conference on, 2010, P. 631 - 636.
5. Candea G., Bucur S., Zamfir C. Automated software testing as a service // Proceedings of the 1st ACM Symposium on Cloud Computing (Indianapolis, USA, 10-11 June 2010), SoCC '10. 2010, P. 155–160.
6. Ciortea L. Cloud9 : A Software Testing Service // ACM SIGOPS Operating Systems Review. 2010. Vol. 4, P. 5–10.
7. Fielding R.T., Taylor R.N. Principled design of the modern web architecture // ACM Transactions on Internet Technology (TOIT). 2002. Vol. 2, P. 115-150.
8. Haryadi S.G. FATE and DESTINI: a framework for cloud recovery testing // Proceedings of the 8th USENIX conference on Networked systems design and implementation (San Jose, USA, 25-27 April 2011). 2011. P. 238–252.
9. Janani V., Krishnamoorty K. Cloud Testing as a Service (CTaaS) – Analysis , Design and Implementation // International Journal of Applied Engineering Research (IJAER). 2015. Vol. 10, No. 12, P. 30393–30405.
10. Pardeshi S.N. Study on Testing as a Service on Cloud // International Journal of Advanced Computer Research. 2013. No. 1, P. 1–4.
11. Pardeshi S.N., Choure V. Testing as a Service on Cloud : A Review // International Journal on Recent and Innovation Trends in Computing and Communication. 2014. Vol. 2, P. 188–193.

12. Riungu L.M., Taipale O., Smolander K. Research Issues for Software Testing in the Cloud // Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on (Indianapolis, USA, 25 November – 3 December 2010). 2010. P. 557 – 564
13. Savchenko D.I., Radchenko G.I., Taipale O. Microservices validation : Mjolnirr platform case study // Proceedings of the 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO'2015 (Opatija, Croatia, 25 May - 29 May 2015). 2015. P. 248–253.
14. Microservices: a definition of this new architectural term. URL: http://martinfowler.com/articles/microservices.html (accessed: 25.12.2015).
15. Pattern: Microservices Architecture. URL: http://microservices.io/patterns/microservices.html (accessed: 25.12.2015).
16. Testing as a Service (TaaS) definition. URL: http://searchcloudapplications.techtarget.com/definition/Testing-as-a-Service-TaaS (accessed: 13.01.2016).
17. IBM Extends Development and Test to the IBM Cloud. URL: http://www-03.ibm.com/press/us/en/pressrelease/29685.wss (accessed: 01.12.2015).
18. Test and Develop with HP Cloud. URL: http://www.hpcloud.com/solutions/test-develop (accessed: 01.12.2015).
19. SOASTA. URL: http://www.soasta.com/ (accessed: 01.12.2015).
20. Load Impact. URL: https://loadimpact.com/ (accessed: 01.12.2015).
21. New Relic. URL: http://newrelic.com/ (accessed: 25.01.2016).
22. Docker - Build, Ship, and Run Any App, Anywhere. URL: https://www.docker.com/ (accessed: 25.01.2016).
23. Mountebank - over the wire test doubles. URL: http://www.mbtest.org/ (accessed: 30.01.2015).