УДК 004.056.5 DOI: 10.18698/2541-8009-2019-09-527

ПОДХОД К ПРОЕКТИРОВАНИЮ МУЛЬТИТЕНАНТНОГО ПРИЛОЖЕНИЯ

В.В. Задорожный benzbmstu@yandex.ru
SPIN-код: 9937-2982

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Аннотация

Введено определение мультитенантности, приведен пример проектирования многопользовательского программного продукта. Графически показаны единицы, из которых строятся подобные приложения. В качестве примера предложена и описана структура web-приложения, полностью удовлетворяющая тематике работы. Для конкретного выделенного стека технологий отмечены некоторые проблемы реализации системных и инфраструктурных процедур, обеспечивающих целостность мультитенантной системы. Для их решения приведены некоторые алгоритмы с целью дальнейшей их автоматизации, а также минимизации ошибок при разработке приложения. Дан пример совместного использования двух серверных компьютеров для итеративной разработки конечного продукта, указано их функциональное значение.

Ключевые слова

Мультитенантность, многопользовательское приложение, SaaS, доменное имя, proxy-сервер, dns-сервер, база данных, backend, клиент, сервер, микросервис, multi-tenant, single-tenant

Поступила в редакцию 02.07.2019

Введение. Сегодня облачные сервисы все шире внедряются в повседневную жизнь пользователей. Особенно привлекательной является возможность круглосуточного использования сервисов с любых устройств и постоянная доступность хранимых там личных данных. Для таких типов сервисов ввели общее понятие SaaS (англ. *software as a service* — программное обеспечение (ПО) как услуга) — одна из форм облачных вычислений, модель обслуживания, при которой подписчикам предоставляется готовое прикладное ПО, полностью обслуживаемое провайдером [1]. Поставщик в этой модели самостоятельно управляет приложением, предоставляя заказчикам доступ к функциям с клиентских устройств, как правило, через мобильное приложение или web-браузер.

Однако у конечного пользователя нередко возникают вопросы относительно качества обслуживания и проверки достоверности личности при авторизации. Также важен вопрос изолированности личных данных от данных других пользователей. Все эти и многие другие принципы учитываются разработчиками при проектировании приложения.

В целом определение *SaaS* сильно обобщает облачные приложения и описывает принципы их построения. Можно говорить о выделенной и мультитенантной архитектурах приложения.

Выделенная архитектура (*single-tenant*) в основном реализуется на одном физическом сервере. Такое приложение можно масштабировать путем увеличения числа серверов — физических или виртуализированных. Этот подход очень неудобен, но иногда одно приложение предоставляет хорошие вычислительные возможности и надежность благодаря ориентации на работу только с одним клиентом, для которого выделяются собственные ресурсы, а падение сервера не затрагивает всех пользователей. Но развитие технологий в области программирования оставило этот вариант позади, и сегодня с SaaS ассоциируется мультитенантная архитектура (*multi-tenant*) [1, 2].

Новый подход в рамках одного сервера реализует многопользовательский доступ, позволяя изолировать ресурсы клиентов. При этом механизм изоляции может быть реализован на уровне инфраструктуры сервера (как в случае с выделенной архитектурой), на уровне данных, на уровне средств и услуг приложения, а также на уровне логики работы приложения. Рассмотрим один вариант из множества возможных реализаций подобного приложения.

Мультитенантная модель. Мультитенантная (многопользовательская) модель web-сервиса состоит из разных уровней, визуально показанных на рис. 1.

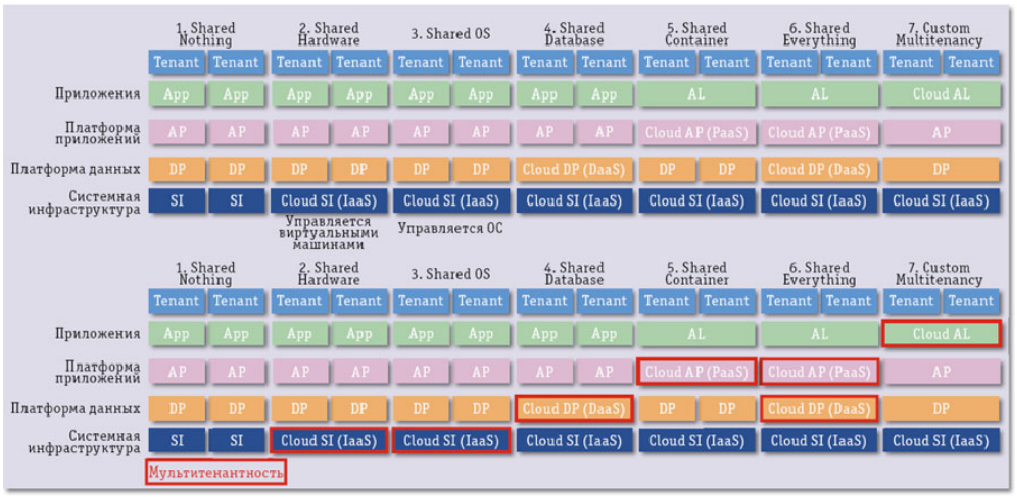

Рис. 1. Структура многопользовательского приложения и варианты организации уровней:

Shared Nothing — все ресурсы недоступны; Shared Hardware — общее оборудование; Shared OS — общая операционная система; Shared Database — общая база данных; Shared Container — общий контейнер; Shared Everythig – все ресурсы общие; Custom Multitenancy — специальная мультитенантность; Tenant — заказчик (клиент); App — приложение; Cloud — облачный (удаленный); AL — уровень приложения; AP — платформа приложения; DP — платформа данных; SI — системная инфраструктура

Важным достоинством такого подхода является возможность комбинирования различных слоев. Далее в статье будет рассмотрен пример реализации приложения с многопользовательским доступом.

Каждому клиенту-заказчику присваивается уникальное доменное имя в рамках приложения, а модель авторизации реализует уникальный ключ для

входа каждого пользователя. Метаданные записываются в одну базу данных, но в разные таблицы, которые создаются отдельно для каждого заказчика. Логика работы приложения реализует права доступа для пользователей в рамках единого рабочего пространства одного заказчика. Все медиаданные хранятся в системных каталогах с ограниченным к ним уровнем доступа на уровне файловой системы [3].

Схема инфраструктуры. Для каждого разрабатываемого приложения известны сценарии его использования и способы предоставления услуг. В данном случае финальным результатом будет являться web-приложение типа SaaS. В роли точки входа в систему выступает ссылка с уникальным ключом. После входа каждому пользователю доступна рабочая площадка, в которой можно хранить как различные модели деталей, полученные в САПР-приложениях, так и любые документы в разных форматах. Система прав доступа в рамках рабочего пространства каждого заказчика ограничивает возможность просмотра и редактирования файлов других членов этой организации. В итоге получаем информационную систему с возможностью изолированно обслуживать заказчиков и их пользователей в рамках одного SaaS-сервиса. Основным здесь является соблюдение изолированности подписчиков друг от друга. Действительно, клиенты не обрадуются, если данные, которые они хранят, будут доступны другим. Это явное нарушение приватности. Для реализации можно выбрать один сервер, одну операционную систему, одну базу данных. Механизм изоляции будет осуществляться с помощью особого хранения данных в таблицах базы данных (БД) и записью файлов в общую файловую систему, а также обработкой и перенаправлением web-запросов от клиентов на микросервисы [4–6]. Для достижения такой целей можно рассмотреть архитектуру сервера, включающего следующие функциональные уровни:

– DNS-сервер;
– Proxy-сервер;
– Backend;
– база данных;
– файловое хранилище;
– средства инфраструктуры.

Каждый уровень выполняет свою задачу. В совокупности с развитыми средствами автоматизации инфраструктуры они реализуют целостную структуру информационной системы приложения с изолированным многопользовательским доступом. Далее рассмотрим каждый из них.

Уровень DNS-сервера. Для начала введем важные определения. Доменное имя — имя компьютера в сети или символьный адрес, к которому можно привязать сетевые ресурсы web-приложения. Доменная зона – совокупность доменных имен определенного уровня. Зона необходима, чтобы обеспечивать управления доменными имена. Делегирование — передача контроля над частью доменной зоны следующей ответственной стороне [7]. На рис. 2 наглядно показано, как происходит процесс распространения доменных имен в сети.

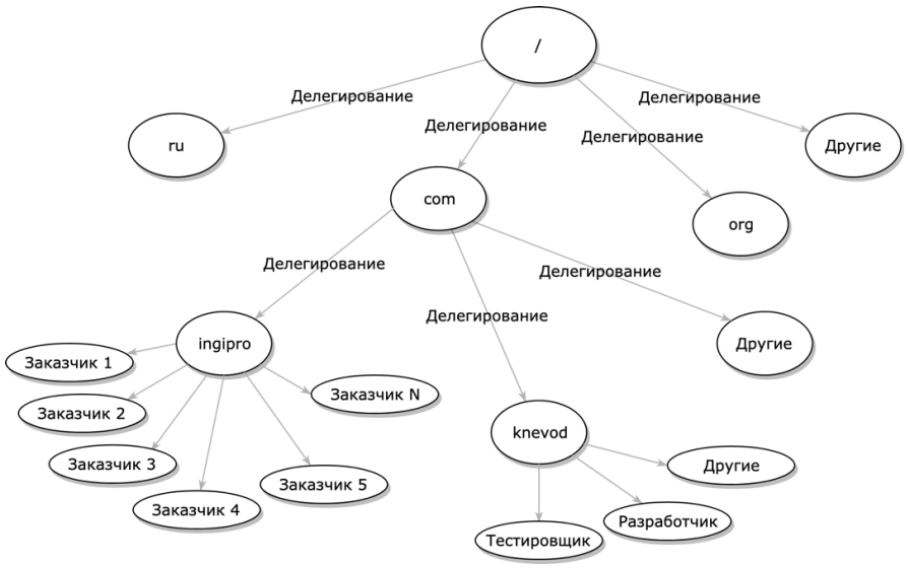

Рис. 2. Распространение доменных имен в виде семантической сети

DNS-сервер в данной ситуации играет важную роль. На уровне доменных имен выполняется первый этап разделения доступа к информационной системе приложения. Это самый верхний уровень, который помогает реализовать множество точек входа в систему. Для того чтобы зарегистрировать нового заказчика, необходимо добавить в доменную зону новое доменное имя заказчика. Так, если информационная система доступна по имени company.com, мы имеем право добавить имя customer в доменную зону. Получим адрес customer.company.com, которым сможет пользоваться заказчик. Остальной механизм авторизации осуществляется на следующих уровнях.

Уровень proxy-сервера. На роль proxy-сервера возлагается несколько задач. Среди них:

а) обработка запрашиваемого адреса с целью перенаправления на другой адрес;

б) фильтрация и перенаправление запросов на уровень *backend*;

в) выдача специальных ресурсов для запрашиваемого доменного уровня;

г) выдача статических ресурсов на сторону клиента.

На этом уровне web-сервер, выступающий в роли proxy-сервера, принимает запросы от клиентов по протоколу HTTP, перенаправляет их на необходимые микросервисы уровня backend, которые в свою очередь реализуют основную логику работы разрабатываемого приложения. Далее от web-сервера клиенту возвращается HTTP-ответ с необходимым содержимым для отображения графического дизайна и всех пользовательских данных в браузере [8, 9].

Уровень backend. Backend является основной логической частью веб-приложения. Его архитектура бывает монолитной и микросервисной. В текущем проекте реализуется микросервисная архитектура, состоящая из отдельных приложений, выполняющих специальные задачи. Так, существует микросервис, отвечающий за авторизацию пользователя в системе; микросервис, отвечающий

за подготовку передачи данных клиенту; микросервис, в задачу которого входит запись всех метаданных клиента в БД; микросервис, отвечающий за подготовку графических данных для отображения у клиента и т. д. Каждый микросервис работает на специально выделенном сетевом порту сервера. В зависимости от поступившего от клиента HTTP-запроса происходит перенаправление на конкретный микросервис через сетевой порт посредством proxy-сервера.

Уровень базы данных. Огромное количество информации пользователей хранится в базе данных. На этом уровне идеология мультитенантности достигается благодаря особому способу хранения информации пользователей. На верхнем уровне заказчики различаются по своим именам: customer1, customer2 и др. Для них в БД создаются уникальные таблицы, которые содержат имена заказчиков, например, attribute_customer1, attribute_customer2, entity_customer1, attribute_customer2. Таким образом можно изолировать данных всех заказчиков, используя только одну БД на одном сервере.

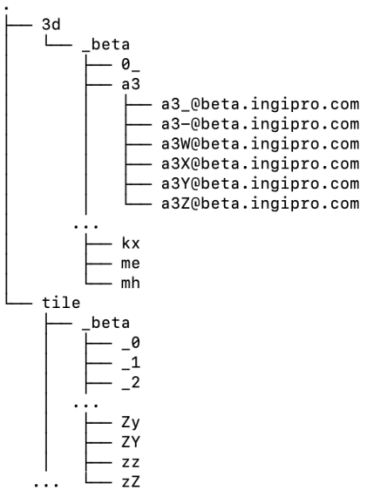

Рис. 3. Представление файлового хранилища в виде дерева

Уровень файлового хранилища. Уровень файлового хранилища представляет собой часть файловой системы, находящейся на локальном или удаленном диске. Здесь хранятся более тяжелые данные мультимедийного характера: документы, изображения, чертежи, аудио- и видеозаписи, а также другие файлы, загружаемые клиентом через браузер. Область файлового хранилища изолирована от системных пользователей на уровне операционной системы, доступ туда имеют только программы, участвующие в работе web-сервиса. Пример именования и хранения таких данных приведен на рис. 3.

На рис. 3 показана структура каталогов для некоторого домена *beta*. Для всех типов данных, хранящихся в такой структуре, права доступа файловой системы могут ограничивать возможность любой попытки прочитать или редактировать такие данные. При таком ограничении уровень файлового хранилища обеспечивает приватизацию данных, а реализация дерева каталогов обеспечивает принцип изолированности данных пользователей.

Средства инфраструктуры. Все описанные выше уровни рассматриваемой мультитенантной модели сервера выполняют специальные задачи. Все вместе они формируют программную систему, которая предлагается как конечный продукт. За реализацию каждого уровня на практике отвечает определенный разработчик или команда разработчиков. Они используют собственные средства, обычно не совместимые с продуктами других уровней. Это разные языки

программирования, сетевые интерфейсы, логики работы приложения, конфигурационные файлы. Часто для внедрения новой версии микросервиса требуется внесение изменений в настройку web или DNS-сервера. Все это требует внимания людей из разных зон ответственности.

Для решения подобных задач используют специальные скрипты и сервисы непрерывной доставки и интеграции [10]. Так, уровень инфраструктуры призван обеспечить систему набором процедур для быстрой и надежной переконфигурации и управления состоянием всех компонентов приложения. Главной целью средств инфраструктуры является автоматизация процессов обновления, останова и запуска. Это всегда уникальные решения, которые адаптированы для разрабатываемой системы.

Автоматизация процессов обслуживания. Под автоматизацией процессов обслуживания будем понимать комплекс мер по сборке программного продукта *(backend*, *frontend* [10]), а также действий по управлению всеми вспомогательными сервисами (останов, запуск, перезапуск) и действий по работе с конфигурационными файлами используемых программ. Все это обеспечивает быструю замену версий.

Причиной возникновения данного направления является ограниченное время, за которое необходимо успеть выполнить работы по обслуживанию. Клиенты не должны замечать перебои в работе приложения.

Сборка микросервисов. Множество действий по разработке ПО связано с постоянными изменениями в коде, его дополнением и тестированием, поэтому очень часто приходится пересобирать из исходного кода часть программного продукта. Особенно часто обращают на себя внимание уровень backend и его микросервисы.

Для развития *web*-приложения используют два сервера: отладочный и производственный. Цели отладочного сводятся непосредственно к разработке ПО, его отладке и тестированию. Финальную версию web-приложения выкладывают на производственный сервер, где клиенты пользуются сервисом, пока на отладочном происходит разработка очередной новой версии программного продукта. Ниже приведен пример алгоритма, который предстоит автоматизировать.

1. Проверка наличия и синтаксиса конфигурационного файла.
2. Обновление исходного кода из удаленного репозитория*:

а) проверка текущей ветки;

б) обновления копии локального репозитория из удаленной ветки.

3. Обновление и модификация фреймворка†:

а) сброс локального репозитория до первоначального состояния;

---

* Место, где хранятся и поддерживаются какие-либо данные. Чаще всего это исходный код и текстовые файлы.

† Шаблоны для программной платформы, определяющие архитектуру программной системы. Обычно это набор готовых библиотек, с помощью которых разработчик внедряется готовое решение в свое приложение.

б) обновление из официального репозитория;

в) наложение патчей[‡], подготовленных программистами микросервисов в отдельной локальной ветке.

4. Выполнение сборки микросервисов:

а) проверка разрешения на пересборку конкретного микросервиса;

б) выполнение сборки проекта микросервиса с помощью встроенного во фреймворк инструмента;

в) установка метки статуса сборки в скрытый временный файл;

г) сохранение старой версии микросервиса в файл в постфиксом .old в директорию с микросервисом.

5. Вызов модуля тестирования микросервисов:

а) проверка статуса сборки;

б) вызов модуля UNIT-тестов;

в) вызов модуля регрессионного тестирования.

6. Корректное завершение процессов:

а) проверка статусов тестирования;

б) проверка разрешения на перезапуск микросервиса;

в) чтение из конфигурационного файла переменной, содержащей максимальное время ожидания программного завершения микросервиса;

г) отправка процессу программного сигнала завершения SIGTERM и старт таймера отсчета;

д) ожидание завершения. Отправка сигнала SIGKILL, если процесс не успел завершиться за указанный интервал времени;

е) проверка закрытия сетевых портов;

ж) установка метки статуса сокета[§], на котором работал микросервис.

7. Запуск микросервисов и проверка их состояния.

а) проверка статуса сетевого порта;

б) запуск и демонизация[**] микросервиса;

в) проверка состояния процесса.

Перезапуск микросервисов. Для реализации перезапуска микросервисов может потребоваться отдельный скрипт, который отвечает за передачу сигнала прерывания в микросервис и обработку реакции останавливаемой программы на этот сигнал. Общий алгоритм останова и перезапуска микросервиса в автоматическом режиме имеет следующий вид:

а) проверка разрешения на перезапуск, отмеченный в конфигурационном файле;

б) проверка наличия рабочих процессов микросервиса в системе;

в) отправка сигнала SIGTERM на программное завершение;

[‡] Файл, содержащий информацию обо всех новых изменениях в файле.

[§] Программный интерфейс, с помощью которого реализуется общение между процессами в системе.

[**] Механизм работы программы в фоновом режиме без привязки к текущей рабочей сессии пользователя в ОС UNIX.

г) ожидание корректного завершения микросервиса и всех его дочерних процессов в течение указанного в конфигурационном файле времени. Если процесс не завершился самостоятельно, осуществляется отправка сигнала принудительного завершения SIGKILL.

д) ожидание закрытия сетевого порта микросервиса. Если порт не закрывается в течение указанного времени, например, 10 с, то автосборка отражает это состояние в параметре статуса порта;

е) проверка состояние сетевого порта по флагу. Если порт не был закрыт, то процесс автосборки завершается с определенным кодом.

Замена версии микросервисов. Процесс замены версий на отладочном и производственном серверах различается. Любая сборка проекта происходит на отладочном сервере. В данном случае, как и в случае с автоматизацией сборки и перезапуска микросервисов, учувствует специально реализованная программа. На отладочном сервере для замены версии микросервисов можно воспользоваться специальной утилитой для сборки, которая перезапишет локальные версии программ. С производственным сервером немного сложнее, так как нужно локально собрать ПО и доставить его.

Алгоритм замены может быть следующим:

а) проверка наличия и синтаксиса конфигурационного файла автосборки;

б) проверка существования бинарных файлов и формирование их списка;

в) установление ssh-соединения с отладочным сервером;

г) остановка микросервисов с использованием того же алгоритма автосборки;

д) создание копии восстановления каждого микросервиса;

г) перезапись старых бинарных файлов новыми;

д) запуск микросервисов;

г) разрыв ssh-соединения;

Наглядно процесс обновления можно изобразить на рис. 4.

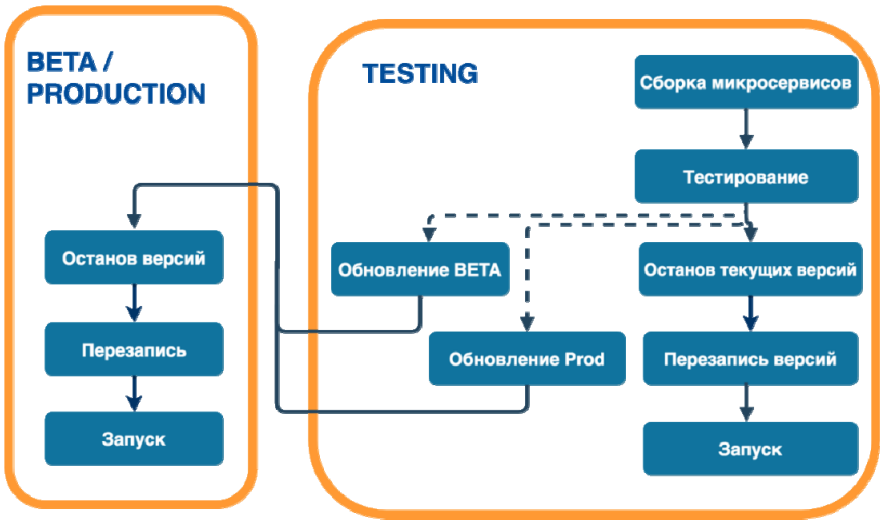

Рис. 4. Процесс обновления версии backend

На рис. 4 BETA, PRODUCTON, TESTING — это типы одновременно работающих копий уровня backend на двух отдельных машинах. BETA и PRODUCTION работают на производственном сервере, а BETA служит для финального тестирования продукта. Если все успешно работает, можно обновлять производственный слой, предназначенный для работы с конечным клиентом.

Регистрация доменного имени. Доменное имя является самым высокоуровневым атрибутом для клиента. Именно его клиент видит самым первым в строке своего браузера. Рассматривая перечисленные выше уровни многопользовательской модели, можно заметить, что доменное имя является уникальной единицей информации, вокруг которой строится вся мультитенантная модель web-приложения. Это имя используется на всех уровнях модели, а на уровне DNS-сервера обеспечивается возможность открытия страницы для входа в систему.

Необходимой процедурой является добавление нового доменного имени или удаление старого. Вспомним, что основным доменом web-приложения является домен второго уровня company.com, а домены третьего уровня такие как customer1, custormer2 и др., принадлежат заказчикам.

Для регистрации клиентских доменов необходимо выполнить следующие шаги:

а) развернуть и настроить DNS-сервер;

б) добавить доменное имя в файл зоны;

в) инкрементировать идентификационный номер файла доменной зоны;

г) перечитать измененный файл доменной зоны;

д) проверить доступность нового доменного имени.

Реализовывать данную процедуру вручную каждый раз неэффективно. Для исключения возможный ошибок предложенный сценарий можно запрограммировать, причем с дополнительными проверками синтаксиса файла доменной зоны.

Заключение. В заключение к данной статье можно сказать, что мультитенантная схема достаточно гибкая. Данный подход сильно превосходит однопользовательский режим и позволяет реализовать многопользовательское приложение с изоляцией данных на различных функциональных уровнях приложения. На рис. 1 показаны эти уровни, но каждый из них можно поделить в зависимости от нужд проектировщика. Рассмотренный в статье стек технологий является частным случаем такого разделения. В настоящее время он успешно используется в нескольких компаниях, предоставлявших услуги SaaS-сервиса.

Литература

[1] Bennet K. et al. Service-based software: the future for flexible software. *Proc. 7th APSEC,* 2000. DOI: 10.1109/APSEC.2000.896702 URL: https://ieeexplore.ieee.org/document/896702

[2] Traudt E., Konary A. Software as a service taxonomy and research guide. IDC Research Report, 2005.

[3] Familiar B. Microservices, IoT and Azure. Apress, 2015.

[4] Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. М., Вильямс, 2003.

[5] Richards M. Microservices vs. service-oriented architecture. O'Reilly Media, 2015.
[6] Newman. S. Building microservices. O'Reilly Media, 2015.
[7] Альбитц П., Ли К. DNS и BIND. СПб., Символ-Плюс, 2002.
[8] Hypertext Transfer Protocol — HTTP/1.1. URL: https://www.ietf.org/rfc/rfc2616.txt (дата обращения: 12.05.2019).
[9] Niu Y., Liu F., Li Z. Load balancing across microservices. IEEE INFOCOM, 2018. DOI: 10.1109/INFOCOM.2018.8486300 URL: https://ieeexplore.ieee.org/document/8486300
[10] Дюваль П.М., Матиас III С.М., Гловер Э. Непрерывная интеграция: улучшение качества программного обеспечения и снижение риска. М., Вильямс, 2008.

Задорожный Владислав Вадимович — студент кафедры «Системы автоматизированного проектирования», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Научный руководитель — Берчун Юрий Валерьевич, старший преподаватель кафедры «Системы автоматизированного проектирования», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Ссылку на эту статью просим оформлять следующим образом:
Задорожный В.В. Подход проектирования мультитенантного приложения. *Политехнический молодежный журнал*, 2019, № 9(38). http://dx.doi.org/10.18698/2541-8009-2019-9-527

MULTI-TENANT APPLICATION DESIGN APPROACH

V.V. Zadorozhnyi benzbmstu@yandex.ru
SPIN-code: 9937-2982

Bauman Moscow State Technical University, Moscow, Russian Federation

Abstract

The article introduces the definition of multi-tenancy, gives an example of designing a multi-user software product. The units from which such applications are built are graphically shown. As an example, the paper proposes and describes the structure of a web application that fully meets the theme of work. For a specific dedicated technology stack, some problems of implementing system and infrastructure procedures that ensure the integrity of a multi-tenant system are noted. To solve them, several algorithms are presented with the aim of further automating them, as well as minimizing errors during application development. The authors give an example of the joint use of two server computers for iterative development of the final product, and indicate their functional significance.

Keywords

Multi-tenancy, multi-user application, SaaS, domain name, proxy server, dns server, database, backend, client, server, microservice, multi-tenant, single-tenant

Received 02.07.2019

References

[1] Bennet K. et al. Service-based software: the future for flexible software. *Proc. 7th APSEC*, 2000. DOI: 10.1109/APSEC.2000.896702 URL: https://ieeexplore.ieee.org/document/896702

[2] Traudt E., Konary A. Software as a service taxonomy and research guide. IDC Research Report, 2005.

[3] Familiar B. Microservices, IoT and Azure. Apress, 2015.

[4] Connolly T., Begg C. Database systems: a practical approach to design, implementation, and management. Addison Wesley, 2001. (Russ. ed.: Bazy dannykh. Proektirovanie, realizatsiya i soprovozhdenie. Teoriya i praktika. Moscow, Vil'yams Publ., 2003.)

[5] Richards M. Microservices vs. service-oriented architecture. O'Reilly Media, 2015.

[6] Newman. S. Building microservices. O'Reilly Media, 2015.

[7] Liu C., Albitz P. DNS and BIND. O'Reilly Media, 2001. (Russ. ed.: DNS i BIND. Sankt-Petersburg, Simvol-Plyus Publ., 2002.)

[8] Hypertext Transfer Protocol — HTTP/1.1. URL: https://www.ietf.org/rfc/rfc2616.txt (accessed: 12.05.2019).

[9] Niu Y., Liu F., Li Z. Load balancing across microservices. *IEEE INFOCOM*, 2018. DOI: 10.1109/INFOCOM.2018.8486300 URL: https://ieeexplore.ieee.org/document/8486300

[10] Duvall P.M., Matyas S., Glover A. Continuous integration: improving software quality and reducing risk. Addison-Wesley Professional, 2007. (Russ. ed.: Nepreryvnaya integratsiya: uluchshenie kachestva programmnogo obespecheniya i snizhenie riska. Moscow, Vil'yams Publ., 2008.)

Zadorozhnyi V.V. — Student, Department of Computer-Aided Design Systems, Bauman Moscow State Technical University, Moscow, Russian Federation.

Scientific advisor — Berchun Yu.V., Senior Lecturer, Department of Computer-Aided Design Systems, Bauman Moscow State Technical University, Moscow, Russian Federation.

Please cite this article in English as:
Zadorozhnyi V.V. Multi-tenant application design approach. *Politekhnicheskiy molodezhnyy zhurnal* [Politechnical student journal], 2019, no. 9(38). http://dx.doi.org/10.18698/2541-8009-2019-9-527.html (in Russ.).