Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

Методические указания
для проведения практических занятий

Томск 2025

УДК 004.021
ББК 32.973.4
Б39

Рецензент:
Сенченко П.В., кафедра АОИ ТУСУР, к.т.н., доцент

Автор: И.В. Безходарнов

Безходарнов, Илья Владимирович
Б39 Микросервисная архитектура: метод. указания для проведения практических занятий / И.В. Безходарнов – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2025. – 15 с.

Методические указания предназначены для проведения практических занятий по дисциплине "Микросервисная архитектура". В ходе занятий предлагается на учебных системах разобраться с несколькими базовыми принципами построения систем с микросервисной архитектурой, а в заключительном занятии обобщить и расширить знания путем сопоставления материала, изученного на практических занятиях с теоретическим материалом, изученном на лекциях.

Одобрено на заседании ПИШ, протокол № 9 от 03.09.2024 г.

УДК 004.021
ББК 32.973.4

Оглавление

Введение

Данные указания для практических занятий помогут вам изучить и понять несколько базовых принципов разработки систем с микросервисной архитектурой.

В темы практических занятий заложена идея, что при создании учебной системы, не обремененной бизнес-логикой реальных приложений легче разобраться с базовыми, основополагающими подходами, методиками, попробовать их на практики и осознать, как они могут быть использованы в реальных системах.

Ограниченный объем часов, отведенных на изучение дисциплины не позволяет включить в него все многообразие методов по созданию подобных систем, что отчасти компенсируется завершающим практическим занятием. В нем предлагается суммировать ваши знания, полученные и в ходе выполнения практических работ и в ходе лекционных занятий и осознать, к каким конкретным пунктам во всем разнообразии методологий микросервисных систем относятся вопросы, рассмотренные в ходе выполнения практик.

Тема №1 Изучение различий между монолитной и микросервисной архитектурами

Цель: изучить на практике и понять отличия между монолитной архитектурой и микросервисной архитектурой

Задачи:

1. Создание учебной системы в монолитной архитектуре.
2. Переделка этой же учебной системы в микросервисной архитектуре.
3. Автоматизировать процесс развертывания обоих вариантов реализации.

Теоретическая часть

Описание логики работы учебной системы:

4. Авторизация пользователя по логину и паролю.
5. Выполнение набора действий, например, генерация случайных чисел по заданным параметрам.
6. Выполнение арифметических операций над полученными числами.
7. Возврат результата.

Конкретные варианты параметров для генерации чисел и операций над ними приведены в практической части.

Данный вариант системы является учебным примером. По предварительному согласованию с преподавателем вы можете предложить свой вариант логики работы учебной системы для чего нужно: 1) сделать описание логики работы; 2) представить его преподавателю и получить подтверждение.

Практическая часть предусматривает работу с использованием командного интерпретатора bash. Это обусловлено в основном его повсеместной доступностью и простотой организации рабочих мест для проведения практических занятий. Официальный сайт с полной документацией: https://www.gnu.org/software/bash/.

По предварительному согласованию с преподавателем можно использовать другую технологию создания приложений, включая веб, для чего нужно его представить преподавателю и показать наличие технической возможности использовать данный вариант.

На последнем этапе практики нужно будет разложить созданные скрипты в разные репозитарии, поэтому в случае, когда вы согласовали свой вариант выполнения работы, это тоже нужно будет сделать, учтите данную информацию при работе над заданиями.

Практическая часть

Реализация в виде монолита

Создаем bash скрипт с логикой авторизации пользователей. Параметры Имя и Пароль берутся из командной строки. На этом этапе нужно добиться, чтобы скрипт проверял наличие параметров, переданных при запуске и, если их недостаточно, или больше, чем нужно, то принудительно завершал выполнение. Для выполнения авторизации достаточно прямо в коде указать трех возможных пользователей с их паролями и проверить входные параметры на соответствие. Если валидная комбинация пользователь:пароль не найдена, то скрипт принудительно завершается.

Примечание: для чтения параметров командной строки в bash можно использовать встроенные переменные ${N}, например $1 – первый параметр, $2 – второй параметр. А для определения сколько параметров было передано переменную $#.

Добавляем в скрипт еще один параметр, он будет отвечать за диапазон, в котором генерируются случайные числа, и его нужно будет указывать третьим по счету в командной строке:

1 – от 0 до 10.
2 – от 0 до 100.
3 – от 0 до 1000.

Примечание: для получения случайного числа в bash можно использовать переменную $RANDOM, которая меняет свое значение случайным образом при каждом ее чтении, а диапазон выдаваемых значений колеблется от 0 до 32767.

Добавляем в скрипт четвертый параметр, который будет отвечать за операции над случайными числами. Например:

1 – умножить полученное число на 2.
2 – получить два случайных числа (в одинаковом диапазоне) и выдать их сумму.
3 – получить два случайных числа и выдать их разность.

Таким образом, у нас получился скрипт, который выполняет следующие действия:

1. При старте проверяет количество переданных параметров и, если оно не равно 4, принудительно завершается.

2. Интерпретирует первые два параметра как имя пользователя и пароль, ищет в заранее заданных вариантах их полное совпадение и в случае неуспеха принудительно завершается.

3. В зависимости от параметров 3 и 4 генерирует случайные числа в определенных диапазонах и производит над ними определенные арифметические действия.

4. Выводит полученный результат.

На этом первый шаг – создание "монолитной системы" завершен.

Реализация в виде микросервисной архитектуры

Давайте разделим получившийся скрипт на несколько маленьких и организуем в дополнительном скрипте их последовательный запуск для выполнения тех же функций, которые реализованы в первом варианте.

Например, первый скрипт будет выполнять только авторизацию. Значит ему на вход будет подаваться всего два параметра Имя и Пароль. Его задачами будет: проверить количество переданных параметров, проверить соответствие переданных параметров одному из заранее заданных вариантов и возвратить статус завершения Успех или Ошибка в зависимости от полученного результата.

Второй скрипт пусть занимается только тем, что возвращает случайное число в заданном диапазоне. Он также должен проверять корректность переданного ему на вход параметра и возвращать статус ошибки или успеха. Кроме того, в случае успеха он должен вернуть случайное число.

Третий скрипт будет аналогом внешнего эндпоинта (иногда его называют шлюзом - gateway), в реальной микросервисной системе. Он будет принимать внешние команды и дальше запуская внутренние скрипты получать конечный вариант и передавать его пользователю.

Его последовательность действий примерно следующая: проверить количество переданных параметров, вызвать скрипт авторизации и проверить статус его завершения, вызвать скрипт получения случайного числа один, или два раза в зависимости от параметров, выполнить нужную арифметическую операцию в зависимости от параметров и вернуть полученный результат пользователю.

Проделав все предложенные действия, вы получите три скрипта, выполняющих ту же самую функцию что и первый скрипт. Сделайте это и проверьте правильность его работы.

Примечание: для возврата значений из вызываемых скриптов используйте механизм конвейеров (pipeline) а для возврата статуса завершения встроенную переменную $?.

Создание инфраструктуры для разработки и функционирования систем

Для первого варианта - монолитной учебной системы нужно сделать отдельный каталог, скопировать в него наш первый скрипт и инициализировать в этом каталоге git репозитарий. Получится аналог среды разработки реальной системы.

Теперь сделайте отдельный, пустой каталог, который будет служить аналогом среды развертывания, т.е. тем местом, где система будет эксплуатироваться. Изначально этот новый каталог – пустой.

В репозитарии, где расположена монолитная система нужно создать дополнительный скрипт, задачей которого будет являться простое копирование главного скрипта в каталог среды развертывания. Этот дополнительный скрипт также закомитьте в репозитарий.

Проверьте, что при запуске скрипта развертывания в каталоге развертывания появляется главный скрипт. Смысл этого действия - разделить среду разработки и среду эксплуатации системы. Т.е. разработчик меняет, отлаживает, тестирует новые версии в своем рабочем каталоге, а когда новая версия готова, запускает скрипт развертывания, и пользователь получает новую версию в эксплуатацию.

Для второго варианта – микросервисной учебной системы все делается аналогично, но придется повторить это несколько раз, для каждого микросервиса отдельно. Создаем для каждого микросервиса отдельный каталог разработки, в каждом из них инициируем git репозитарий, в каждом создаем скрипт для развертывания и заливаем в свой репозитарий. Каталог для среды исполнения - будет только один, все скрипты микросервисов должны копироваться в него. Таким образом получилось, что каждый из микросервисов – это отдельный проект, он разрабатывается, отлаживается, тестируется независимо от остальных и может быть запущен в живую систему в любой момент индивидуально. Т.е. процесс развертывания каждого микросервиса не нужно синхронизировать с другими микросервисами, что позволяет в реальных проектах менять такую систему независимо, по частям, не останавливая всей работы как в случае с монолитной архитектурой.

Полученные результаты

В ходе практического занятия должны получится следующие артефакты:

1. Для монолитной системы каталог для разработки, который находится под управлением системы контроля версий исходного кода git. В этом каталоге два скрипта,

первый — это наш рабочий скрипт, второй - скрипт для развертывания. Рабочий каталог системы, куда скрипт развертывания копирует рабочий скрипт.

2. Для микросервисной системы несколько каталогов для разработки, по одному на каждый микросервис, каждый из которых находится под управлением системы контроля версий git. В каждом из этих каталогов по два скрипта - рабочий и скрипт развертывания. Рабочий каталог системы, куда скрипты развертывания копируют все микросервисы.

Тема №2 Изучение принципов работы брокеров сообщений для систем с микросервисной архитектурой

Цель: изучить методологию построения систем в микросервисной архитектуре с применением брокеров сообщений в логике очередь-подписка

Задачи:

1. Создание учебного варианта брокера сообщений, реализующего логику очередей-подписок.
2. Модификация учебной системы, полученной в предыдущем занятии на вариант общения микросервисов через брокер сообщений в логике очередей.

Теоретическая часть

Существует два основных шаблона обмена сообщениями в микросервисной архитектуре. Первый – это обмен точка-точка, когда каждый микросервис знает к какому микросервису нужно обратиться, чтобы получить определенный результат.

Второй – это обмен через модель подписки на очереди сообщений, когда отправитель не знает кому попадет его сообщение, он лишь снабжает это сообщение идентификатором очереди, в которую оно должно попасть, а остальные микросервисы, которые имеют интерес получить такие сообщения, заранее подписываются на определенные очереди.

Во втором случае нужно, чтобы отдельная система, называемая в микросервисной архитектуре "Брокер сообщений" реализовала функционал очередей.

В данном занятии предлагается самостоятельно разработать учебный брокер сообщений и воспользоваться им, чтобы модифицировать систему, полученную в ходе первого занятия к варианту обмена информацией между микросервисами через очереди.

Брокер сообщений в нашем случае будет представлять собой отдельный скрипт. Микросервисы нашей системы будут запускать его для того, чтобы передать сообщение другим микросервисам. Таким образом брокер сообщений должен будет, получив на вход имя очереди и параметры сообщения, запустить все микросервисы, подписанные на данную очередь, и передать им изначально полученные входные параметры. Соответственно, наш брокер сообщений должен уметь определять какие микросервисы надо запустить по имени очереди.

Практическая часть

Разработку брокера следует вести в отдельном каталоге для разработки, а в каталог рабочей системы размещать с помощью скрипта развертывания. Для отладки брокера

сообщений можно создать дополнительные тестовые скрипты, которые должны находится в каталоге для разработки, а в каталог рабочей системы не попадать.

Сначала предлагается разработать способ, по которому брокер сообщений будет сопоставлять имя очереди и набор микросервисов, которые надо запустить. Так как в общем случае мы не знаем какие будут очереди и какие будут микросервисы, нужно сделать этот механизм универсальным и конфигурируемым. Можно, например, воспользоваться конфигурационным файлом, в котором прописать строки следующего вида: "имя очереди: имяСервиса1, имяСервиса2, …". Синтаксис этого файла можно выбрать любым, в зависимости от того каким образом вы будете разбирать эти строки в скрипте брокера сообщений.

Завершите разработку этой части отладкой системы, которая будет по входным параметрам "имяОчереди параметр1 параметр2…" выводить на экран список всех сервисов, которые надо запустить на основании информации из конфигурационного файла. При выводе на экран за именем микросервиса должны следовать все параметры, первоначально переданные брокеру сообщений. Файл настройки является атрибутом рабочей системы, поэтому в каталоге для разработки брокера будет лежать своя версия файла конфигурации очередей, а в рабочем каталоге системы, своя версия файла.

Следующим шагом создайте и отладьте код, который будет запускать реальные скрипты. Для отладки воспользуйтесь каким алгоритмом визуализации происходящего. Например, выводом каждого шага на экран или в лог файл. В качестве микросервисов на этом шаге используйте простейшие модельные скрипты без какой-либо логики кроме той, которая обеспечит вам диагностику происходящего. В процессе отладки и тестирования проверьте ситуацию, когда два или более сервисов передают друг другу сообщения "по кругу". Когда будете уверены, что ваш брокер сообщений корректно реализует всю требуемую логику, можно переходить к модификации системы, полученной не предыдущем шаге.

Модификация учебной системы, перевод ее на вариант взаимодействия через брокер сообщений

Сформируйте имена очередей для системы и набор параметров для каждого сообщения. Учтите, что в этом методе обмена информацией нужно обеспечить взаимодействие типа запрос – ответ. Например, для получения случайного числа один микросервис должен его запросить, другой вернуть, но механизм очередей в нашей реализации не сохранит отправителя. Значит, первоначальный запрос нужно снабдить параметром, по которому его можно будет идентифицировать при получении ответа.

Дальше нужно модифицировать и отладить микросервисы. Будет плюсом, если при этом вы будете разрабатывать новую версию микросервиса в каталоге для разработки и релизить его в рабочую систему через скрипт развертывания. Для того, чтобы упростить процесс, можно дополнительно к каждому микросервису разработать тестирующий его скрипт. Тестирующий скрипт должен будет запустить микросервис с различными комбинациями параметров и проверить результат выполнения. Тестирующий скрипт разместите в репозитарии соответствующего микросервиса.

Действуйте пошагово, постепенно усложняя логику работы системы и доведите ее до состояния, когда система по функционалу станет полностью аналогична, той, что была создана на предыдущем занятии.

Полученные результаты

В ходе практического занятия должны получится следующие артефакты:

1. Каталог для разработки брокера сообщений, находящийся под управлением системы контроля версий исходного кода, со скриптом развертывания, файлом конфигурации очередей и возможно со скриптами для тестирования.
2. Набор микросервисов, каждый в своем каталоге для разработки, там же должен находиться скрипт развертывания микросервиса и возможно скрипты для его тестирования.
3. Рабочий каталог системы, в котором расположены готовые микросервисы, работающие совместно, брокер сообщений и файл настройки очередей.

Тема №3 Обобщение знаний, полученных в ходе изучения предмета

Цель: сопоставить результаты, полученные в ходе практических занятий, с теоретическими знаниями, полученными в ходе лекций

Задачи:

1. Изучение литературы.
2. Подготовка отчета.

Теоретическая часть

На двух предыдущих занятиях были созданы собственная учебная система и собственный брокер сообщений. В процессе этого занятия нужно понять, что это лишь малая часть подходов, которые используются в системах с микросервисной архитектурой, и задач, которые приходится решать при их создании. Кроме того, существует множество инструментов, которые реализуют эти подходы и сильно облегчают разработку, отладку и эксплуатацию подобных систем. Для выполнения практической части воспользуйтесь уже полученными знаниями и источниками, которые помогут получить дополнительные сведения. В качестве источников можно использовать приведенные в данных методических указаниях, а также любые, доступные и достоверные сведения о системах с микросервисной архитектурой.

Практическая часть

Дайте ответы, описание, ваше мнение относительно следующих пунктов:

1. Критерии, по которым принимается решение о выборе архитектуры между монолитной и микросервисной для конкретной системы.
2. Особенности решения задачи авторизации в микросервисной архитектуре, сравнение с монолитной.
3. Особенности решения задачи логирования в микросервисной архитектуре, сравнение с монолитной.
4. Особенности решения задачи развертывания в микросервисной архитектуре, сравнение с монолитной.
5. Возможности конфигурирования системы, построенной на микросервисной архитектуре в процессе ее функционирования, без перезапуска сервисов.
6. Приведите примеры продуктов с указанием их предназначения и кратким описанием, предназначенных для применения в разработке и эксплуатации систем с микросервисной архитектурой.

Полученные результаты

1. Отчет в свободной форме.

Заключение

Системы, построенные на микросервисной архитектуре, уже получили большую популярность и распространение. Не менее широким является и набор инструментов, помогающих создавать и эксплуатировать такие системы. Благодаря этому набору, а также методологическим основам, заложенным в данную архитектуру, изначальный ее недостаток – большая, по сравнению с монолитными системами, сложность инфраструктуры для разработки и эксплуатации была нивелирована и превратилась в преимущество. Методологии и инструменты, предназначенные для создания и эксплуатации микросервисных систем, бурно развиваются, и нужно использовать эти достижения на практике. Поэтому получив понимание базовых принципов, будьте открыты для расширения своих знаний и самостоятельного изучения новых и уже существующих инструментов, без применения которых на практике не получится использовать все, заложенные в данную методологию возможности.

Список рекомендуемой литературы

1. Microservice Architecture : сайт / microservices. – URL: https://microservices.io. – Режим доступа: свободный (дата обращения: 15.08.2024).

2. GNU Bash : сайт / gnu. – URL: https://www.gnu.org/software/bash/. – Режим доступа: свободный (дата обращения: 16.08.2024).

3. ZeroMQ – An open-source universal messaging library : сайт / zeromq. – URL: https://zeromq.org. – Режим доступа: свободный (дата обращения: 18.08.2024).

4. Apache Kafka : сайт / kafka.apache. – URL: https://kafka.apache.org/. – Режим доступа: свободный (дата обращения: 29.08.2024).