ПРОЕКТИРОВАНИЕ УНИВЕРСАЛЬНОЙ СИСТЕМЫ БРОНИРОВАНИЯ НА БАЗЕ БЕССЕРВЕРНЫХ ВЫЧИСЛЕНИЙ

О.Н. Цесько
Научный руководитель: И.А. Ботыгин
Томский политехнический университет
E-mail: oleg.tsesko@gmail.com

Введение

Бессерверные вычисления – естественная для облака архитектура, которая позволяет передать большую часть операционной ответственности облачному провайдеру и тем самым получить больше гибкости и инновационных возможностей [1]. Бессерверные вычисления позволяют создавать и запускать приложения и сервисы, не беспокоясь о серверах. Они устраняют необходимость заниматься вопросами управления инфраструктурой – такими, например, как выделение серверов или кластеров, необходимых ресурсов, а также установка исправлений и обслуживание операционной системы. Их можно использовать практически для любого типа приложений или сервисов серверной части, при этом все, что требуется для запуска и масштабирования приложения с высокой доступностью, выполняется без вмешательства пользователя [2].

Разрабатываемый проект является онлайн системой для бронирования абстрактных сущностей событийного характера. Разберём детальнее в данном контексте термины «абстрактный» и «событийный». Событийный характер означает, что сущности в системе связаны с определёнными событиями во времени. Простым примером события может быть концерт, перелёт на самолёте, поездка на поезде и т.д. У всех этих примеров есть момент времени начала и завершения. Именно для сущностей подобного рода работает данная система. Система абстрактная – это значит, что система может предоставлять услуги для бронирования любых событийных сущностей, не привязываясь к конкретному контексту. На базе одного и того же программного решения можно реализовать бронирование сущностей из совершенно разных областей: авиация, железные дороги, концерты, грузовые перевозки и так далее.

Структура и архитектура системы бронирования

Универсальная система бронирования – это составной проект. Он состоит из нескольких слоёв функциональных элементов (нод). Каждый слой может быть расширен как горизонтально, так и вертикально, порождая новые слои. Сложность и разнонаправленность системы бронирования увеличивается с числом слоёв. Самым нижним слоем является слой базовых нод – Base Node Layer (BNL). Базовые ноды (BN) являются атомарными, т.е. неделимыми. Этот слой является слоем «рабочих лошадок», определяющий состояние запрашиваемых и резервируемых ресурсов. Всё многообразие абстракций данной системы бронирования базируется на комбинациях нод этого слоя.

На базе проведенного анализа большинства возможных способов реализации аналогов данной системы бронирования, а также посредством упрощения представления взаимодействия слоёв системы, была разработана инновационная иерархическая схема делегирования полномочий по выделению и резервированию ресурсов (см. рисунок).

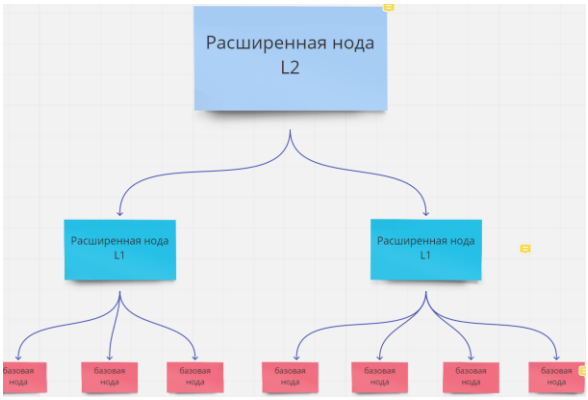

Рис. Схема по делегированию сложности функциональности бронирования

Работает данная схема следующим образом. Например, необходимо поддерживать сложную сущность для бронирования, имеющую множество различных классов мест, а также составные элементы. В конкретной реализации ярким примером может быть авиарейс с одной пересадкой, имеющий два класса мест. Для этой реализации понадобится четыре базовых ноды (под каждый класс-участок), две ноды уровня L1 (для объединения участков), и одна нода L2 (для организации требуемого многообразия). Ноды L1 и L2 делегируют сложную функциональность на базовые ноды и между друг другом. Важная особенность системы – делегируют только сверху вниз, но никак не наоборот.

Выбор инструментов для реализации системы бронирования

Площадок с облачными функциями (бессерверными вычислениями) – большое множество. У всех свои плюсы и минусы, у всех

свои преимущества и нюансы, особенности и правила. Для данной разработки, выберем Firebase от корпорации Google. Firebase управляется одним облачным аккаунтом Google, и включает в себя не только облачные функции, но и множество других инструментов. В эксперименте для реализации этого проекта также используется облачная база данных от Firebase – Firestore. Благодаря HTTP-интерфейсу облака Google, мы можем не вдаваться в подробности реализации сетевых взаимодействий.

Реализация системы бронирования

Давайте посмотрим на схему, и убедимся в том, что здесь прослеживается рекурсия, начиная со слоя L1 и вверх до бесконечного слоя. Слой базовых нод (L0) не подпадает под общие правила рекурсии для верхних слоёв. Как было указано ранее, это единственный слой с «рабочими лошадками», и описывается он совершенно другими функциональными особенностями – слой является фундаментом всей системы бронирования.

Таким образом, нужно будет описать функции базового слоя, функции верхних слоёв и возможность для рекурсии, чтобы можно было включать до бесконечности количество слоёв в приложение.

Для начала разработаем базовый слой. Для него понадобятся следующие функции: «создать сущность на базовом уровне», «забронировать слоты», «разбронировать слоты», «проверить наличие свободных слотов». Также важно реализовать ответ, в случае ошибок и в случае успешного занятия мест.

После того, как базовый слой разработан, проектируем функции для верхних слоёв: «создать высокоуровневую сущность из нод нижележащего уровня», «забронировать места в высокоуровневой сущности», «разбронировать места в высокоуровневой сущности», «получить состояние о местах». Как видно, функциональность похожа на функциональность базового слоя, но, есть большое различие – это всё на более высоком уровне, включающем множество базовых нод. Тем самым предоставляется сложная сущность для бронирования. Если понадобится произвести бронирование сложной сущности, можно обращаться к высшим уровням, которые, в свою очередь, передают команду на бронирование каждой ноде своего нижележащего уровня. Нижележащий уровень, получив команду бронирования от вышележащего, также передаёт команду, но уже своим нижележащим нодам. И так далее до тех пор, пока вызов не дойдет до самого нижнего уровня, который и произведёт реальное физическое занятие мест в системе. Также важно разработать возможность ответа в случае ошибок и успешного занятия мест, а также ожидание ответа от нижележащих уровней, как базового, так и нижних по рекурсии. Когда любая из функций дождётся ответа от нижнего уровня, пересылаем ответ на более высокий уровень, и так далее по цепочке, пока не дойдём до уровня-инициатора бронирования

Далее разработаем возможность рекурсии и делегирования между уровнями, это можно определить в каждой функции верхних уровней, где мы проверим, является ли нижележащий уровень базовым. Если да, то передадим управление на базовые функции, а если нет, то передаём управление в эту же функцию рекурсивно, но с дочерними данными.

Теперь, если мы отправим запрос какой-нибудь ноде верхнего уровня, например, на L2, алгоритм проверит, на каком уровне находятся «подчинённые» данной ноды. В данном случае – не на базовом. Значит алгоритм направит схожий запрос всем своим нижележащим нодам из уровня L1. Каждая нода L1, получив команду, проверит свои нижележащие ноды. В данном случае, это уровень базовых нод. Значит. L1 направит запрос уже не по рекурсии, а на функции базовых уровней.

Заключение

В докладе продемонстрирована возможность использования облачных функций (бессерверных вычислений) с целью реализации сложной многоуровневой системы бронирования. Система описана вкратце, в сжатой форме. Возможности по её расширению и улучшению – безграничны. Мы можем наращивать различные функции для прикладных целей как вертикально, так и горизонтально.

С помощью простого доступа к облачным функциям по HTTP-протоколу, можно использовать комбинированные вызовы функций для организации более сложного многосоставного прикладного использования.

Разработка многосоставной архитектуры, наращивающей свою структурную сложность композицией простых составляющих, является хорошим решением вкупе с применением облачных функций.

Список использованных источников

1. Официальная документация AWS // Amazon. URL: https://www.aws.amazon.com (дата обращения: 14.01.2020).
2. Информационный портал Habr // URL: https://www.habr.com (дата обращения 14.01.2020)
3. Официальная документация Firebase // Google Firebase. URL: https://firebase.google.com (дата обращения: 14.01.2020).
4. Miro // URL: https://www.miro.com (дата обращения: 14.01.2020).