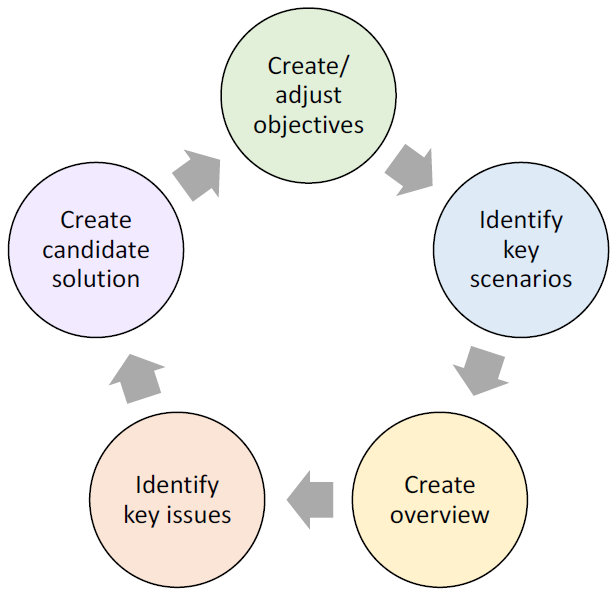
## Итеративен процес на проектиране на архитектура

Този метод включва итеративни цикли за усъвършенстване и разработване на подробни архитектури, подходящи за екипа за разработка. Всеки цикъл усъвършенства детайлите и точността на архитектурата. Използвайки този итеративен подход, архитектите могат да създават всеобхватни и усъвършенствани системни архитектури, които се развиват в тандем с изискванията и ограниченията на проекта.

Процеса се състои от пет основни стъпки. Всяка стъпка е от решаващо значение за гарантиране, че архитектурата е проектирана да отговаря на целите на проекта, да е мащабируема, стабилна и производителна.



Фиг. Process For Designing Architectures

Създаване на цели:

При започване на разработването на архитектурата на базирана в облак система за управление на поръчки, която е основна в управлението на бизнеса. Началната стъпка на „Създаване/настройване на цели“ придобива многостранни измерения. Първо, от решаващо значение е да се определи обхватът на архитектурата. Това включва разпознаване кои системи, като ERP, управление на автопаркове и различни системи за проследяване в различни страни, ще взаимодействат с централна платформа. Точната оценка на времето, необходимо за интегриране и внедряване, гарантира, че сроковете са реалистични и ресурсите са разпределени ефективно. Освен това, ясното разбиране на аудиторията – от професионалистите по веригата за доставки до крайните клиенти – е от основно значение за приспособяване на системните функционалности и интерфейси към техните нужди. Също така е от съществено значение да се разпознават всички ограничения за техническо използване и ограничения за внедряване. Те могат да бъдат продиктувани от регионално съответствие, стандарти за оперативна съвместимост или опасения за поверителността на данните. Чрез щателно определяне на цели около тези параметри в самото начало, може да се гарантира, че получената система за управление на поръчките е стабилна, удобна за потребителя и безпроблемно интегрирана в по-широката екосистема за управление на доставките.

Идентифицирайте ключови сценарии:

Неразделна стъпка в разработването на облачна система за управление на поръчки се крие в определянето на ключовите сценарии, които ще диктуват нейната функционалност и устойчивост. Ранното разпознаване на значими неизвестни или рискове позволява проактивни стратегии за смекчаване, осигурявайки плавни интеграции с ERP, управление на автопаркове и системи за проследяване в множество държави. Освен това е от съществено значение да се подчертаят най-значимите случаи на употреба, особено тези, които са критични за бизнеса и имат силно въздействие върху цялостната система за управление на доставките. Разбирането на пресечните точки на качество и функция гарантира, че системата отговаря или надхвърля очакванията за производителност, като същевременно запазва основните си функционалности. Потвърждаването и планирането на компромиси между различните системни атрибути също може да осигури балансирана и ефективна архитектура. Чрез фокусиране върху тези значими случаи на употреба, особено такива от огромно значение за бизнеса и крайните клиенти, системата може да бъде проектирана така, че да приоритизира жизненоважни функции и да осигури оптимална производителност, дори при взискателни обстоятелства.

Use Case Diagrams

Създайте преглед на приложението:

Базираната в облака система за управление на поръчки е замислена като многостепенно уеб приложение, предназначено да оптимизира процесите на веригата за доставки чрез централно обединяване на данни от различни източници. Като се има предвид сложната интеграция, необходима с ERP системи, платформи за управление на автопаркове и системи за проследяване от множество нации, ограниченията за внедряване трябва да отчитат различни регулации за данни, съображения за латентност и стандарти за оперативна съвместимост в тези страни. Предложеният архитектурен модел за това приложение клони към подход на микроуслуги, осигуряващ мащабируемост, устойчивост и лесна интеграция с различни външни системи. За да се улесни агрегирането на данни в реално време и безпроблемното разпространение на информация до крайните потребители, се обмислят авангардни облачни технологии и платформи като AWS Lambda за изчисления без сървър, Amazon RDS за управление на релационни бази данни и Kubernetes за оркестрация на контейнери. Този основен преглед поставя началото на създаването на стабилно, ефективно и глобално способно решение за управление на поръчки.

Определете типа на приложението, ограниченията за внедряване, модела на архитектурата и технологията.

Идеален момент за създаване на диаграма на слоеве.

Layer Diagram

Идентифицирайте ключовите проблеми

За успешното функциониране на нашата базирана в облака система за управление на поръчки, няколко качествени характеристики възникват като незаменими. Характеристиките на времето за работа на системата, като производителност и наличност, са приоритетни, за да се гарантира, че събирането на данни от ERP, управлението на автопарка и системите за проследяване в различни географски райони е непрекъснато и ефективно. Модулността и скалируемостта остават решаващи от гледна точка на дизайна, позволявайки на системата да се адаптира към увеличаващите се потоци от данни и потребителски бази. Характеристики, ориентирани към потребителя, като използваемост и достъпност, осигуряват последователен, интуитивен интерфейс за крайните потребители в различни страни. Като се имат предвид проблемите в цялата система, удостоверяването и оторизацията представляват значително предизвикателство, особено предвид международния обхват на източниците на данни; трябва да гарантираме, че само упълномощен персонал има достъп до подходящите набори от данни. Необходими са важни механизми за кеширане, за да се ускори извличането на данни и да се подобри потребителското изживяване. Комуникацията без усилие между различните интегрирани системи, управлението на конфигурацията за справяне с различните настройки и предпочитания в различните държави и цялостните механизми за регистриране и управление на изключения са от съществено значение за бързото идентифициране и разрешаване на проблеми. Валидирането гарантира, че данните, интегрирани от различни източници, са точни и последователни, запазвайки надеждността на системата. От съществено значение за ефективното разгръщане и функциониране на нашето решение за управление на поръчките е директното справяне с тези решаващи проблеми.

Обърнете внимание на качествени атрибути като производителност, сигурност, мащабируемост и др.

Обърнете внимание на проблемите на цялата система като удостоверяване, кеширане, регистриране и др.

5. Създайте кандидат решение

Пътуването към внедряване на ефективна облачна система за управление на поръчки започва с основната архитектура. Тази първоначална структура служи като наша отправна точка, използвайки настоящите технологии и методологии и вземайки предвид интегралната функция на агрегиране на данни от ERP, управление на автопаркове и системи за проследяване по целия свят. Докато се движим напред, ние описваме нашата перспективна архитектура, усъвършенствана версия, която включва подобрения, усъвършенствания и потенциални алтернативи, които се стремят да се приведат по-добре в съответствие с нашите стратегически цели и оперативни изисквания. Тази фаза е особено важна за задоволяване на разнообразните изисквания на крайните потребители в множество страни, осигуряване на безпроблемен поток от данни и представяне на информация в разбираем формат. Несигурността обаче е неизбежна в областта на архитектурата. По този начин ние използваме архитектурни шипове, които са ограничени във времето и целенасочени инициативи, за да се изправим и разрешим конкретни неясноти или препятствия в дизайна. Чрез тези скокове можем да потвърдим жизнеспособността на нашите решения, като гарантираме, че не само крайната система е стабилна и мащабируема, но също така удовлетворява строгите изисквания на интегрирана екосистема за управление на доставките в много държави.

Проектиране на архитектура с помощта на UML диаграми.

Обърнете внимание на архитектурните скокове (изследователски проекти).

Опционално създаване на диаграми на класове въз основа на специфични нужди.

Насоки за всеки цикъл:

Намалете архитектурните рискове с всяка итерация.

Обърнете внимание на повече изисквания, сценарии и ключови проблеми във всеки цикъл.

Пътуването към внедряване на ефективна облачна система за управление на поръчки започва с основната архитектура. Тази първоначална структура служи като наша отправна точка, използвайки настоящите технологии и методологии и вземайки предвид интегралната функция на агрегиране на данни от ERP, управление на автопаркове и системи за проследяване по целия свят. Докато напредваме, ние определяме нашата кандидат-архитектура, усъвършенствана версия, включваща подобрения, усъвършенствания и потенциални алтернативи, които се стремят да се приведат по-добре в съответствие с нашите стратегически цели и оперативни изисквания. Тази фаза е особено важна за задоволяване на разнообразните изисквания на крайните потребители в множество страни, осигуряване на безпроблемен поток от данни и представяне на информация в разбираем формат. Но в областта на архитектурата несигурността е неизбежна. Ние използваме архитектурни скокове, които са ограничени във времето, целенасочени инициативи, за да се изправим и разрешим конкретни неясноти или препятствия в дизайна. Всеки цикъл на усъвършенстване е щателно проектиран, за да гарантира, че няма да бъдат въведени нови опасности. В действителност нашият фокус продължава да бъде върху смекчаването на по-голям брой рискове от базовото, отговаряне на допълнителни изисквания, улесняване на по-голям брой критични сценарии и адресиране на по-голям брой критични проблеми. Чрез тези итеративни подобрения ние сме уверени, че ще предоставим система, която може да издържи на сложността на интегрираната екосистема за управление на доставките с множество държави, като същевременно остава устойчива и стабилна.

система ще свързва потребителите с шофьорите и ще предоставя карта на пътуването и за двете страни.

Възникват няколко въпроса относно дизайна:

Трябва ли системата да изчислява ETA въз основа на местоположението на водача?

Могат ли шофьорите да бъдат съпоставени с множество пътници за споделени пътувания?

Как изчисляваме таксите за пътуване: въз основа на време, разстояние или комбинация?

От тези въпроси заключихме:

Регистрацията на потребителя и влизането са от съществено значение.

Водачите се подлагат на проверка на миналото и преглед на превозното средство преди ръчна регистрация.

Нашата основна цел е да минимизираме времето за изчакване на потребителите и разстоянието за пътуване на водача.

Ще разчитаме на външни услуги за ETA и обработка на плащанията.

Нефункционални изисквания:

Ние сме глобална услуга с милиони потребители и хиляди водачи.

Нашата система трябва да управлява пиковете на търсенето и да се мащабира според изискванията.

Местоположенията на драйверите се актуализират на всеки 5-10 секунди, което води до милиарди ежедневни съобщения.

Високата наличност е критична; ние се стремим към почти нулев престой.

Влизането на потребителите и съпоставянето на превозите трябва да са бързи, в идеалния случай в рамките на определени милисекунди.

Процес и последователност:

Системните участници включват драйвери и потребители.

Регистрираните драйвери получават токен за удостоверяване за по-нататъшна комуникация.

Шофьорите споделят местоположението си чрез GPS на устройството си.

Новите потребители предоставят съществени подробности за регистрация.

Потребителите могат да поискат превози, които се съпоставят въз основа на наличните шофьори наблизо.

Двупосочната комуникация е необходима както за водача, така и за потребителските приложения.

След пътуването се извършват изчисления на плащанията и потребителите и шофьорите получават подробности за пътуването по имейл.

Диаграмата на състоянието представя различните състояния на водача, от влизане до това да стане достъпен за возене, да бъде в пътуване и да излезе от системата.

Сега, с изискванията, диаграмите на последователността и състоянието и дефинициите на API, можем да продължим към дизайна на архитектурата, който ще бъде разгледан в следващия раздел.

Проектиране на услуга за споделено пътуване: архитектура и функционалности

Проектиране на системата:

Използвана архитектура на микроуслуги.

Услуга за потребители и водачи: Поддържа профили на потребители и водачи. Съхранява изображения в хранилище на обекти. Управлява процесите на влизане и регистрация.

3. Взаимодействие водач-ездач:

Услуга на драйвер: Обслужва връзките на драйвера, които са дълги и двупосочни. Използва това за актуализации на местоположението в реално време и за изпращане на подробности за пътуването.

Услуга за местоположение: Актуализира и управлява местоположението на драйвера. Взаимодейства с помощта на модел, управляван от събития. Съхранява местоположението в база данни NoSQL.

4. Механизъм за съвпадение на карането:

Rider Service: Обслужва връзките на водача, подобно на водачите.

Услуга за съпоставяне: Изчислява най-добрите съвпадения въз основа на данни в реално време. Използва външна ETA услуга за точно съвпадение. Също така съдържа усъвършенствана логика за ефективно споделяне на превози и справедливо разпределение на пътуванията между водачите.

Услуга за пътуване: Управлява данни за текущи пътувания. Записва подробности за началото, напредъка и края на пътуването.

5. Дейности след пътуването:

Ще обсъди дейности като фактуриране, плащания, карти на пътувания, разписки и имейли в следващата лекция.

Сега нашият фокус се измества към нефункционални изисквания, за да гарантираме мащабируемост, достъпност и оптимална производителност. Основните прозрения включват:

Потребителски трафик: Нашата платформа е свидетел на милиони ежедневни потребители и значителен трафик на водачи. Тези числа могат да скочат до десетократно по време на пиковите часове.

Мащабируемост на услугата: Директните уеб API услуги се поставят зад балансьорите на натоварването, работещи като идентични екземпляри. Въпреки че услуги като генератора на карта на пътуването и услугата за известяване не са изправени пред директен уеб трафик, те могат да се мащабират въз основа на обема на трафика.

Двупосочна свързаност: Услугите за водач и ездач, които са изключително важни за поддържането на двупосочни връзки, изискват повече от просто балансиране на натоварването. Представяне на услуга за управление на връзките, оборудвана с хранилище за ключ-стойност в паметта, помага за проследяване на връзките. Всеки път, когато потребител или драйвер се свърже, тази услуга се актуализира със съответната информация за хоста. Това улеснява услугата за местоположение при локализиране и комуникация с желания хост.

Мащабируемост на хранилище за данни: Използваме разделяне и шардинг за нашите хранилища за данни и въвеждаме API шлюз за отделяне на потребителски и драйверни приложения от вътрешността на системата.

Висока наличност: Въпреки че се добавя репликация за бази данни и брокери на съобщения, за да се гарантира висока наличност, ние ще насочим основното си внимание към оптимизиране на производителността.

Ефективност при влизане: Ние се стремим към бързи отговори при влизане на потребителя, особено при въвеждане на неправилно потребителско име. Вместо маршрутизиране през цялата система, ние предлагаме подход в паметта. Първоначално обмисляйки хеш-таблица, осъзнаваме, че е неефективна поради високите разходи за памет за мащабиране. Решението? Филтър за разцвет. Тази пространствено ефективна структура от данни проверява потенциални потребителски имена с минимален отпечатък на паметта. Използването на множество хеш функции намалява шансовете за сблъсък, което прави процеса по-надежден. Bloom Filters обаче могат само да добавят нови потребителски имена, но не и да ги премахват. Това ограничение не е загрижено, тъй като изтриването на потребител е рядко събитие и филтърът Bloom може да бъде преизчислен, когато е необходимо.

Бързо съпоставяне: Основно изискване е бързото сдвояване на потребители и водачи, когато е заявено пътуване. Ще се задълбочим в това предизвикателство в следващите дискусии.

Резюме: Укрепихме мащабируемостта и наличността на системата с помощта на балансиране на натоварването, шардинг и репликация. Въведохме услуга за управление на връзките, за да преодолеем празнината между потребители и хостове. И, най-важното, интегрира Bloom Filter, за да ускори влизането на потребителя, като същевременно минимизира консумацията на ресурси. Следващата стъпка ще се съсредоточи върху оптимизирането на процеса на съпоставяне между потребител и драйвер.

В последното си изследване ние имахме за цел да разгледаме мащабируемостта, наличността и оптимизирането на нашата услуга за споделяне на превози. Ключовите оптимизации включват намаляване на заявките към базата данни за неправилно въведени потребителски имена чрез филтри за разцвет. Основното предизвикателство остава: ефективното свързване на потребителите с драйверите.

Традиционният метод включваше създаване на виртуален радиус около запитващ потребител, намиране на драйвери в рамките на този радиус и след това определяне на най-близкия драйвер с помощта на външна ETA услуга въз основа на координати в реално време. Външната ETA услуга не можа да бъде оптимизирана, тъй като е извън нашия контрол. И все пак методът за определяне на близките драйвери беше в нашата компетентност и беше най-скъпата от изчислителна гледна точка част поради изчисленията с плаваща запетая върху огромен брой драйвери.

За да коригираме това, въведохме техника за местоположение, базирана на клетки. Земната повърхност беше разделена на уникални клетки, всяка идентифицирана с ID. Местоположението на водачите ще бъде картографирано към тези клетки. Когато потребителите поискат превози, само няколко клетки (обикновено 3-4) в близост се запитват за шофьори, което значително ускорява процеса. Освен това внедрихме Geohash, популярен алгоритъм, който разделя Земята на мрежа от 32 правоъгълника, допълнително усъвършенствайки нашия подход. Това позволява ефективно индексиране и заявки без сложни математически изчисления.

В заключение, с помощта на базирана на клетка система за местоположение и по-специално алгоритъма Geohash, ние разработихме мащабируема, ефективна и ефективна услуга за споделяне на превози, която успешно отговаря на всички наши нефункционални изисквания.

**Scalable Rideshare Service - Additional Resources**

[**Bloom Filter**](https://en.wikipedia.org/wiki/Bloom_filter)

* A space-efficient probabilistic data structure, where false positive matches are possible, but false negatives are not. Elements can be added to the set but not removed. This is a perfect technique for applications where the amount of source data would require an impractically large amount of memory if "conventional" error-free hashing techniques were applied.

[**Geographical distance calculation**](https://en.wikipedia.org/wiki/Geographical_distance)

[**Geohash**](https://en.wikipedia.org/wiki/Geohash)

* Encodes a geographic location into a short string of letters and digits. For a full description, see [this Wikipedia article](https://en.wikipedia.org/wiki/Geohash" \t "_blank). Note that GeoHash and GeoHashing are different things.
* [GeoHash Explorer](https://geohash.softeng.co/)- Online and interactive GeoHash visualization tool.

[**Google's S2 Geometry Library**](https://s2geometry.io/)

* Used by [Uber Technologies, Inc](https://www.uber.com/" \t "_blank), this library represents all data on a three-dimensional sphere (similar to a globe). This makes it possible to build a worldwide geographic database with no seams or singularities, using a single coordinate system, and with low distortion everywhere compared to the true shape of the Earth.