# Глава 3. Изграждане и използване на облачна система за производствено предприятие Титан Цимент

Тази глава разглежда внедряването на базирана на облак и микроуслуги система за управление на поръчки за Titan Cement, водещ производител на пазара за цимент, бетон и други материали. Трета глава изследва технологиите и резултатите от внедряването на междинна система, интегрираща ERP, управление на автопарк и мониторинг. За да се потвърди приложимостта на предложената архитектура и модел на софтуерна система, разработени във втора глава, системата трябва да бъде внедрена в реална работна среда.

## 3.1. Обща характеристика на дейността на компанията

TITAN Cement е мултинационална компания със седалище в Атина, Гърция, и основен участник в глобалната индустрия за цимент и строителни материали. Корпорацията, която е основана през 1902 г., има операции в много страни, включително Европа, Близкия изток, Северна Америка и Африка. TITAN Cement Company S.A. произвежда, транспортира и разпространява широка гама строителни материали, като цимент, бетон, инертни материали и свързани стоки. Освен това те предоставят услуги на строителната индустрия.

Компания е специализирана в производството и доставката на бетонова смес, която се произвежда в централно съоръжение за дозиране. Терминът „готови смеси“ произлиза от факта, че тези смеси се произвеждат според спецификациите на клиента, което води до прецизен, висококачествен продукт, който може да се използва веднага след доставката.

За дозиране на бетон компанията използва големи централни съоръжения. Това дава възможност за прецизно и точно смесване, гарантирайки целостта на продукта. TITAN Cement произвежда смеси според изискванията за здравина, обработваемост и издръжливост. Често готовата смес се доставя в миксери и може да се използва веднага след пристигането. Това гарантира, че бетонът има най-високо качество, когато се излива, тъй като свойствата на бетона могат да се променят с времето.

Тъй като компаниите за готови смеси подготвят големи количества бетон на централно място, те могат да работят по-ефективно и в по-голям мащаб, отколкото ако бетонът се смесва на отделни строителни обекти.

TITAN Cement прилага строги мерки за контрол на качеството. Тъй като бетонът се комбинира в контролирана среда, TITAN Cement могат да гарантират, че съставът е прецизен и отговаря на всички приложими спецификации и стандарти. Централизираното смесване може да бъде по-благоприятно за околната среда от смесването на място, тъй като намалява отпадъците и позволява по-голям контрол върху използваните материали.

Продуктите на компанията се използват за изграждане на къщи, инфраструктура, търговски и промишлени съоръжения, като по този начин отговарят на нуждите на нарастващото световно население за жилища, мобилност и икономическо развитие. Основната дейност включва производство и дистрибуция на цимент, инертни материали, готови бетонови смеси и асфалт.

### 3.1.1. Основни бизнес процеси в компанията

Доставката на готов бетон (RMC) включва няколко основни бизнес процеса, от приемане на поръчки до доставка на клиента.

Приемане на поръчки: Това е първият етап, при който се получават поръчки от клиенти за конкретни видове и обеми бетон, често чрез специално бюро за поръчки или онлайн система. Този процес включва разбиране на нуждите на клиента по отношение на типа бетон, обема и времето за доставка.

График: След като поръчката бъде взета, тя трябва да бъде планирана за производство и доставка. Това е сложна задача, която отчита фактори като производствен капацитет, наличност на флота за доставка и периоди за доставка на клиента.

Дозиране: В този процес бетонът действително се произвежда или „дозира“. Това включва точно измерване и комбиниране на суровините (цимент, инертни материали, вода и всякакви добавки) в съответствие с конкретния изискван дизайн на бетонната смес.

Товарене: Доставеният бетон след това се зарежда в камион-миксер. Този процес често трябва да се извърши бързо, за да се сведе до минимум времето между дозирането и доставката, тъй като бетонът започва да се втвърдява веднага щом водата и циментът се смесят.

Доставка: Миксерът доставя бетона до обекта на клиента. Процесът на доставка трябва да вземе предвид най-ефективните маршрути и времена за доставка, за да се гарантира, че бетонът пристига, докато все още може да се работи.

Разтоварване и поставяне: На площадката бетонът се разтоварва и поставя според изискванията. Клиентът обикновено е отговорен за действителното поставяне на бетона, но екипът за доставка може да се наложи да съдейства или съветва.

Връщане и почистване: След доставката миксерът се връща в завода, където се почиства и подготвя за следващото натоварване. Този процес трябва да се управлява внимателно, за да се предотврати натрупването на втвърден бетон в миксера.

Таксуване и обслужване на клиента: След доставка клиентът се таксува за поръчката. Процесите на обслужване на клиенти също влизат в действие тук, включително справяне с всякакви запитвания или проблеми и събиране на обратна връзка, за да се помогне за подобряване на бъдещото обслужване.

Поддръжка: Този процес включва рутинна поддръжка и ремонт на производственото оборудване и флота за доставка, за да се осигури непрекъсната и ефективна работа.



Ефикасността и ефективността на тези процеси пряко влияят върху качеството на услугата, оперативната ефективност и рентабилността на операциите на RMC. Следователно, доброто управление на тези процеси е от решаващо значение за успеха на всеки RMC бизнес.

### 3.1.2. Стимулиране на продажбите чрез цифрови технологии

Пакета от приложения стимулира прозрачността на данните, стандартизация в ERP, по-бързо и рентабилено планиране в заводите, създават иновативно решение, ориентирано към потребителите. Услугитe с добавена стойност, правят клиентите по-логистично интегрирани и по-добре оборудвани за да посрещнат предизвикателствата, свързани с устойчивостта. Целят да намалят материалните и логистични разходи, въглеродния отпечатък, да подобрят производителността и клиентския опит.

Проследяване на камиона с готов бетон по пътя му към строителна площадка, съхраняване на билети и протоколи едно място, с мобилно и уеб приложение. Благодарение на контрола, циментът се произвежда по по-устойчив и ефективен начин, който намалява отпечатък върху околната среда и осигурява безопасност на работниците. Обслужването на клиенти се рационализира чрез постоянната връзка с бек-офиса и превозвача.

Технологичния пакет се основава на авангардни технологии с отворен код, с най-новите програми и езици. Контейнерни услуги, работещи изцяло в облак и разпределени в множество регионални клъстери.

Преход от хартиени документи към цифрови документи за доставки на бетон. Цифровите документи и цялата информация за доставка ще бъдат достъпни чрез приложението. Някои от плюсовете са:

▪ Без повече липсващи документи;

▪ Край на събирането и съхраняването на документи;

Те ще съдържат информация за поръчки, документи, фактури, резултати от тестове и др.

OnSite е нашето приложение, което ви помага да управлявате и проследявате напредъка на вашите конкретни доставки в движение в реално време.

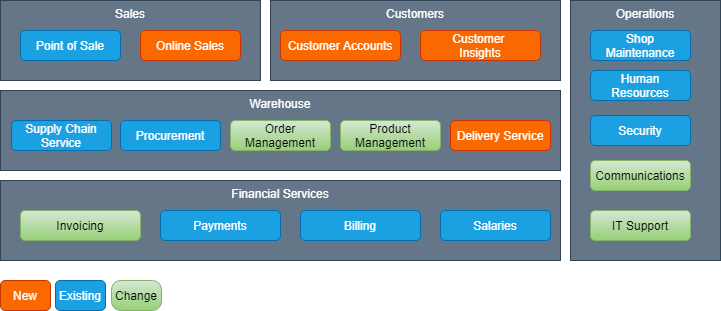
Hub е нашият онлайн портал за управление, проследяване и свързване на цялата информация, свързана с продукта. Вашите поръчки, документи, фактури и протоколи от тестове – всичко това е на едно място.

Приветстваме ви в нашето безхартиено пътуване през нашите дигитални платформи.

#### Описание

Основната бизнес цел е да се ускори и оптимизира обработката и изпълнението на поръчките, като се осигурят навременни и точни доставки, като по този начин се увеличи удовлетвореността на клиентите и евентуално продажбите. Системата за администриране на поръчки трябва да бъде лесна за използване, ефективна и адаптивна към множество устройства и платформи, като мобилни телефони, таблети и настолни компютри, като също така да създаде интегрирана връзка с външен системи за управление на бизнеса.

#### Бизнес архитектура: Карта на възможностите



### 3.1.3. SWOT анализ

Фигура 3.x. изобразява SWOT анализ на готовността на Titan Cement за приемане на облачното решение, включващ силни страни, уязвимости, възможности и заплахи, като се цели отстраняване на слабостите и смекчаването на заплахите.



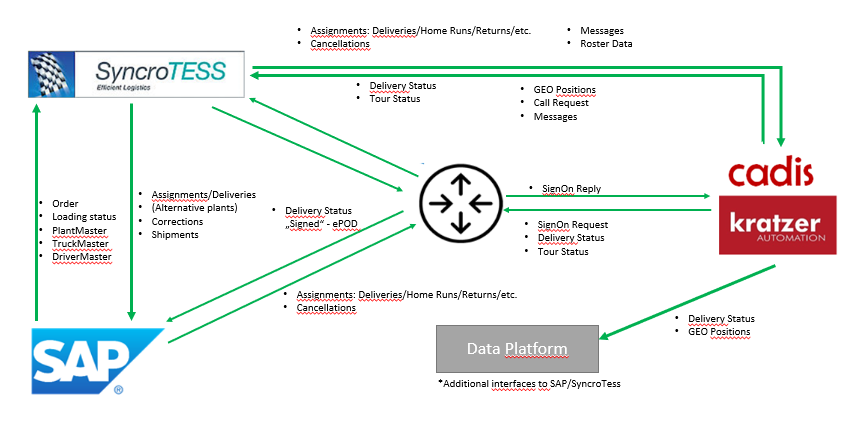
***Фиг 3.****: SWOT анализ*

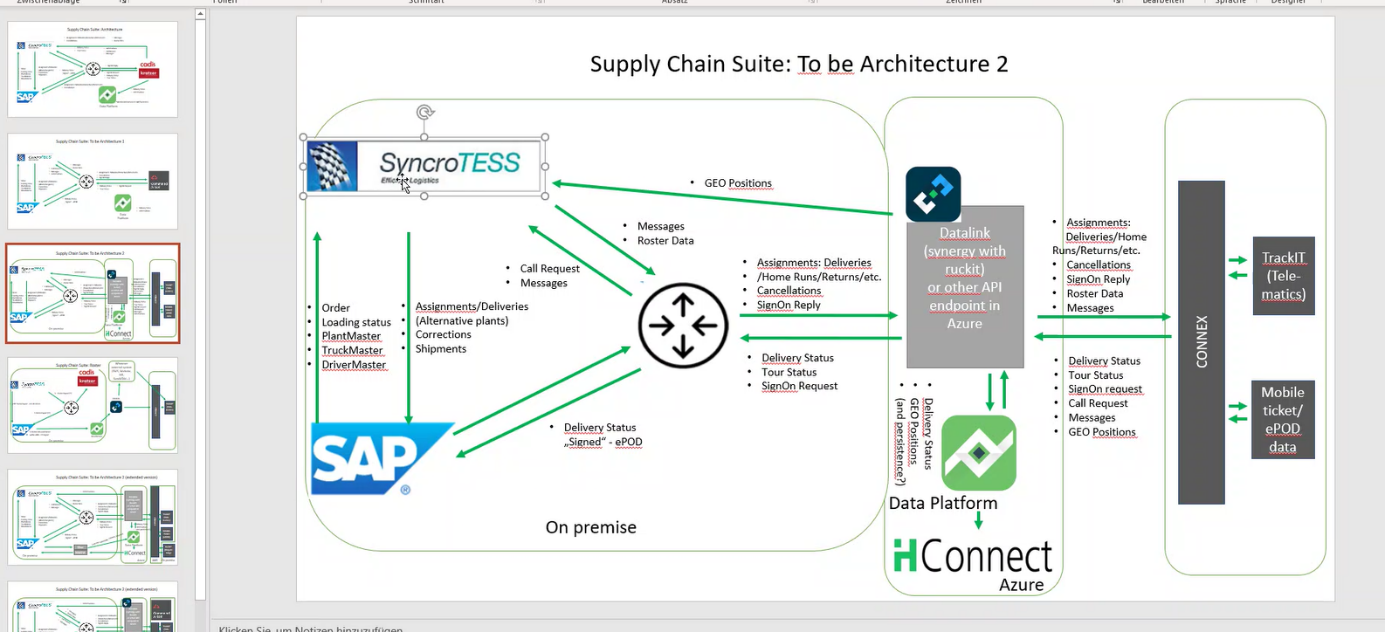
Двигателите на бизнеса са интелигентни функции, устойчивост и системна интеграция. Бизнес резултатите се състоят от оперативна ефективност и интелигентни функции. Поради липсата на интеграция и автоматизация между системите, оперативната ефективност е отрицателно повлияна от забавената верига на доставки. Броят на интегрираните системи е KPI. Поради недостига на интелигентни функции в превозните средства за доставка, зашеметяващият ръст на приходите е свързан с интелигентните функции. Ключовият показател за ефективност е внедряването на мобилното IoT приложение.

## 3.2. Приложение на облачната система в дейността на Titan Cement

Въвеждането на базирани в облака системи за управление на поръчки трансформира динамиката на веригата за доставки чрез централизиране на изолирани преди това процеси, като по този начин повишава прозрачността и ефективността на глобалния логистичен сектор. Чрез агрегиране на данни от различни глобални системи за ERP, управление на автопаркове и мониторинг, тези системи революционизират управлението на доставките, като предоставят на крайните потребители безпрецедентно ниво на контрол и достъпност. Техният дизайн включва както разпределено изчисление, така и усъвършенствани техники за извличане на данни, което им позволява безпроблемно да интегрират различни източници на данни, да ги анализират в реално време и да представят полезна информация. Това събиране на данни служи за двойна цел: не само дава възможност на клиентите да проследяват поръчките си в реално време и с детайлност, като по този начин подобрява потребителското изживяване и доверие, но също така предоставя на бизнеса критична информация за ефективността на веригата им за доставки, проправяйки начин за повтарящи се подобрения и иновации. Чрез намаляване на информационните бариери, базираните в облак системи за управление на поръчките ускоряват комуникацията и процесите на вземане на решения в сложни вериги за доставки, сигнализирайки за основен крайъгълен камък в похода на логистичната индустрия към по-голяма ефективност, прозрачност и ориентираност към клиента.

### 3.2.1. модел на процесите на управление на веригата за доставки





### 3.2.2. Ниво на зрялост на архитектурата

Todo

### 3.2.3. модел на разходите

A Cost Model Breakdown is a detailed analysis of all the expenses associated with a project, product, or service. For a cloud-based order management system that plays a central role in the supply management system, integrating with various systems such as ERP, fleet management, and tracking systems, the cost model would need to factor in a wide array of components. Here's a potential breakdown:

1. \*\*Infrastructure Costs:\*\*

- \*\*Cloud Hosting:\*\* Pricing for the storage and computing power needed.

- \*\*Backup and Recovery:\*\* Costs associated with ensuring data integrity and availability.

- \*\*Networking:\*\* Charges for data transfer, especially if dealing with multiple countries and vast amounts of data.

2. \*\*Software & Licensing Costs:\*\*

- \*\*Cloud Platform Licenses:\*\* If you're using services like AWS, Azure, or Google Cloud, they have specific pricing.

3. \*\*Development Costs:\*\*

- \*\*Initial Development:\*\* Costs related to the initial creation of the system.

- \*\*Integration Development:\*\* Building integrations to ERP, fleet management, and tracking systems.

- \*\*Continuous Development:\*\* Ongoing expenses for updating and expanding the system's capabilities.

4. \*\*Operational Costs:\*\*

- \*\*Support & Maintenance:\*\* Ongoing costs for troubleshooting, bug fixes, updates, etc.

- \*\*Monitoring:\*\* Tools and services to monitor system performance, health, and security.

- \*\*Training:\*\* Costs to train internal teams or customers to use the system.

5. \*\*Security Costs:\*\*

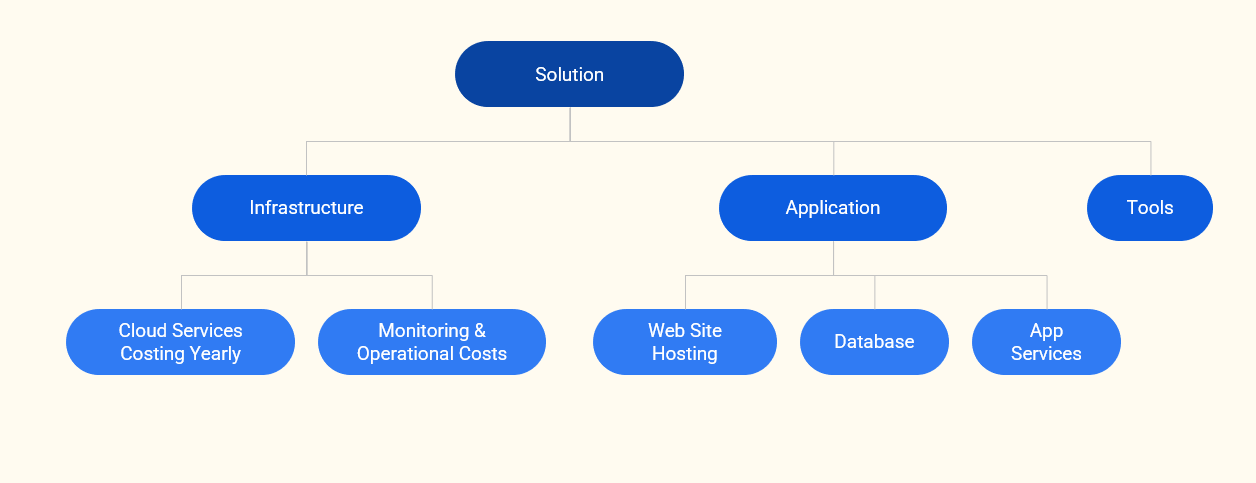
- \*\*Security Software & Tools:\*\* Investments in firewalls, encryption, intrusion detection systems, etc.

- \*\*Compliance Audits:\*\* If dealing with sensitive data, there might be costs related to ensuring regulatory compliance.

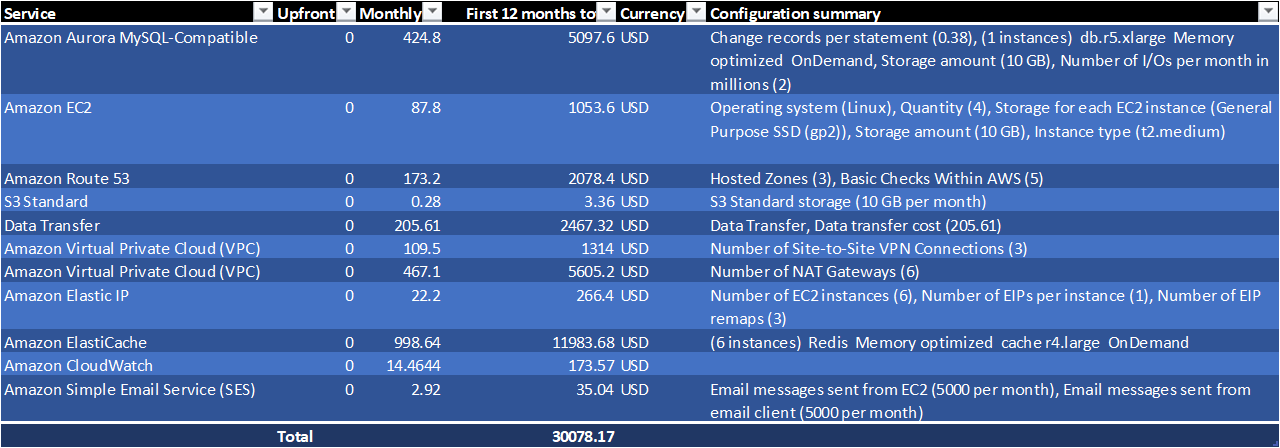
6. \*\*Integration Costs:\*\*

- \*\*Middleware:\*\* Costs related to software that acts as a bridge between different systems.

- \*\*API Calls:\*\* Some systems might charge based on the number of API calls or the volume of data transmitted.



expected Cost - Per year



## 3.3. Физическа реализация на системата

### 3.3.1. Избор на технологични средства за реализация на системата

Това е решаващ избор, главно защото е почти необратим и разработчиците имат привързаност към предпочитаните от тях инструменти. В допълнение към изпълнението на целта трябва да се вземат предвид следните елементи:

• Проучване в Google Trends за данни относно популярността на определено технологично средство;

• Настоящи умения на разработчиците и тяхното по-нататъшно развитие;

• Oценка на разходите за използване на съществуващи и нови инструменти;

Таблици 3.3 представят анализ на сървърните технологии, подходящи за изпълнение на заданията.

***Таблица 3.3****: Сравнение на сървърни технологии за разработка.*



**.NET Core**

Статични или динамични: .NET Core, като рамка, поддържа както статични, така и динамични типове системи.

Инфраструктура: Най-подходящ за инфраструктура на Microsoft. Безпроблемна интеграция с Azure и други услуги на Microsoft. Поддържа разработка на различни платформи.

Общност: Има стабилна общност, особено в корпоративния свят. Поддръжката на Microsoft осигурява редовни актуализации и подобрения.

Производителност: Добро представяне. Известен с приложения на корпоративно ниво.

Крива на обучение: Умерено до високо, особено за разработчици, които не са запознати с C# и .NET екосистемата.

Набиране на персонал: умерено. Разработчиците с .NET умения са търсени, но те може да изискват по-високи заплати от някои други езици.

Облачен хостинг: Отличен, особено на Azure, където има безпроблемна интеграция.

**Node.js**

Статични или динамични: Node.js поддържа както статични, така и динамични системи.

Инфраструктура: Node.js е лек и може да се използва с почти всяка инфраструктура, включително Linux, Windows и macOS.

Общност: Много активна общност. Има множество пакети с отворен код, налични за Node.js.

Производителност: Известен с висока производителност, особено в IO-свързани приложения.

Крива на обучение: умерено. Разработчиците, запознати с JavaScript, ще го намерят по-лесно.

Набиране на персонал: лесно. Като се има предвид популярността на JavaScript, има голям набор от потенциални разработчици на Node.js.

Облачен хостинг: Отличен. Node.js е много мащабируем и се представя добре в облачна среда.

**PHP**

Статичен или динамичен: PHP се използва предимно за динамични уебсайтове, но може да се използва и за статични сайтове.

Инфраструктура: PHP работи на почти всички видове сървъри и е част от LAMP стека (Linux, Apache, MySQL, PHP).

Общност: Има масивна общност с обширни библиотеки и рамки, като Laravel.

Производителност: Подходяща за много приложения, но може да не е подходяща за приложения с много висока производителност.

Крива на учене: Като цяло лесен за научаване, особено за уеб разработка.

Набиране на персонал: лесно. PHP се използва широко от много години и има голям набор от PHP разработчици.

Облачен хостинг: Добре. Много хостинг доставчици предлагат PHP поддръжка. Въпреки това може да изисква повече конфигурация от някои други технологии.

**Java**

Статични или динамични: Java поддържа както статични, така и динамични системи. Въпреки това, той се използва по-често за динамични системи.

Инфраструктура: Java е независима от платформата, което означава, че може да работи на всяка машина, която има среда за изпълнение на Java.

Общност: Java има много голяма и активна общност. Освен това има обширни библиотеки и рамки като Spring.

Производителност: Висока. Java се използва в много критични за производителността приложения.

Крива на обучение: Умерено до високо. Самата Java е сравнително проста, но екосистемата може да бъде сложна.

Набиране на персонал: умерено. Java се използва широко от много години, но по-новите технологии може да са по-привлекателни за някои разработчици.

Облачен хостинг: Отличен. Java има добра поддръжка за внедрявания, базирани на облак, особено в приложения на корпоративно ниво.

**Python**

Статични или динамични: Python поддържа както статични, така и динамични системи. Въпреки това, той се използва по-често за динамични системи.

Инфраструктура: Python е гъвкав и може да се използва на практика във всяка инфраструктура. Често се използва в Linux среда.

Общност: Python има активна общност, особено в областта на науката за данни и машинното обучение.

Производителност: По-ниска от някои други езици, но често достатъчна за много приложения.

Крива на учене: Лесно. Python често се препоръчва като първи език за програмиране.

Набиране на персонал: лесно. Python е популярен и широко преподаван, така че има голям набор от разработчици на Python.

Облачен хостинг: Добре. Python се използва широко в облачни среди, но може да не работи толкова добре, колкото някои други езици за приложения с много високо натоварване.

Всички гореспоменати технологии имат своите силни и слаби страни и най-подходящата зависи от конкретния случай на употреба. Като се има предвид това, .NET Core е стабилна работна рамка с силна поддръжка, особено в среди на Microsoft и Azure облачен хостинг. Има голяма общност и осигурява висока производителност. .NET Core се вписва като най-подходящ вариант, въпреки по-стръмна крива на обучение спрямо други технологии.

Следваща стъпка е анализиране на Azure, Google Cloud и AWS по следните фактори:

* Съвместимост и интеграция с различни технологии;
* Поддръжка от общността и от трети страни;
* Мащабируемост;
* Цена;
* Крива на обучение;
* Набиране на персонал (Лесно намиране на квалифицирани специалисти);

**Microsoft Azure**

Съвместимост и интеграция: Отлична интеграция със софтуера на Microsoft и .NET Core. Той също така поддържа широк набор от други технологии.

Поддръжка от общността и от трети страни: Значителна поддръжка от трети страни и нарастваща общност, особено сред фирми, инвестирали сериозно в продукти на Microsoft.

Мащабируемост: надеждни възможности за автоматично мащабиране.

Цена: Ценообразуването е конкурентно, с редица различни модели на ценообразуване и налични опции. Предприятията със съществуващи договори на Microsoft могат да получат по-добри сделки.

Крива на обучение: Умерено, особено за тези, които вече са запознати с екосистемата на Microsoft.

Набиране на персонал: Умерено, тъй като пазарният дял на Azure е по-малък от този на AWS.

**Google Cloud Platform (GCP)**

Съвместимост и интеграция: Добра поддръжка за различни технологии. Интеграцията с услугите на Google е отлична.

Поддръжка от общността и от трети страни: Разрастваща се поддръжка от общността и от трети страни, но по-малка от AWS и Azure.

Мащабируемост: Известен с отлична мащабируемост, особено за големи данни, анализи и натоварвания с машинно обучение.

Цена: Често се счита за най-рентабилната опция за изчислителни екземпляри, но зависи от конкретния случай на употреба.

Крива на обучение: умерено. Някои уникални концепции, като проекти и инструмента за команден ред gcloud.

Набиране на персонал: По-трудно, тъй като GCP има по-малък дял от облачния пазар в сравнение с AWS и Azure.

**Amazon Web Services (AWS)**

Съвместимост и интеграция: Поддържа широка гама от технологии. AWS разполага с широк набор от услуги, които се интегрират добре помежду си.

Поддръжка от общността и от трети страни: Има най-голямата общност и най-много инструменти от трети страни поради статута си на лидер на пазара.

Мащабируемост: Силно мащабируеми, с множество услуги, специално проектирани за мащабиране.

Цена: Ценообразуването може да бъде сложно, но AWS предлага широка гама от опции и е конкурентен в повечето области.

Крива на обучение: Умерена до висока, в зависимост от използваните услуги. AWS има много услуги и функции.

Набиране на персонал: Най-лесният сред трите поради доминирането на пазара на AWS.

Azure предлага отлична интеграция с .NET Core и може да използва всяка съществуваща връзка, която организация има с Microsoft като активна директория, мейл сървър и други. AWS и GCP също поддържат .NET Core, като всички три платформи имат силни и слаби страни. Най-подходящият избор, зависещ от конкретен случай на употреба и инфраструктура е Azure.

Разработката на мобилни приложения е виждала предимно три типа платформени технологии: собствени, хибридни и кросплатформени. Всеки от тях представлява уникална комбинация от езици за разработка, интегрирани среди за разработка (IDE), ниво на достъп до функциите на смартфона и потребителско изживяване. Родните технологии, като Swift за iOS и Java или Kotlin за Android, предлагат превъзходна производителност и високо ниво на достъп до възможностите на устройството, тъй като използват специални IDE като Xcode и Android Studio. Те обаче налагат паралелно разработване и поддръжка за всяка платформа. От друга страна, хибридни технологии като Apache Cordova (HTML, CSS и JavaScript) предлагат модел „пиши веднъж, изпълнявай навсякъде“ с умерено ниво на достъп на устройството. Все пак те могат да направят компромис с производителността на приложението и да се почувстват по-малко „родни“. И накрая, междуплатформени технологии като React Native (JavaScript и JSX) и Flutter (Dart) се опитват да преодолеят тази празнина. Използвайки свои собствени IDE като Visual Studio Code или IntelliJ IDEA, тези технологии улесняват единична кодова база, която се компилира към естествен код, осигурявайки по-подобно на естественото потребителско изживяване, като същевременно предоставя значителен достъп до функциите на устройството. Въпреки това, междуплатформените технологии може все още да изостават малко от местните технологии по отношение на достъпа до най-новите специфични за платформата функции или когато се работи с изключително сложни графични интерфейси. Важно е внимателно да разгледате предимствата и недостатъците на всяка платформа в контекста на специфичните изисквания на проекта, за да направите информиран избор.

***Таблица 3.4****: Сравнение на мобилни технологии за разработка.*



Решения за балансиране на натоварването с отворен код:

HAProxy:

Силни страни: Възможност за персонализиране, предоставя разширени опции за маршрутизиране на трафика, стабилно регистриране и страхотна производителност.

Случаи на употреба: Големите предприятия използват HAProxy не само за балансиране на уеб трафика, но и за TCP приложения.

Общност: Силна общност с отворен код. Винаги се актуализира с нови функции.

NGINX:

Силни страни: Освен балансиране на натоварването, той може да се използва като уеб сървър, пощенски прокси и HTTP кеш.

Случаи на употреба: Не е само за големи уебсайтове. Поради лекия си характер дори малки и средни фирми използват NGINX за обслужване на своите уеб страници.

Разширения: Комерсиалната версия, NGINX Plus, предлага разширени функции, включително постоянство на сесията, проверки на състоянието и др.

Облачно базирани решения за балансиране на натоварването:

AWS - Еластично балансиране на натоварването (ELB):

Интеграция: Тясно интегрирана с AWS услуги като EC2, ECS, EKS и др.

Сигурност: С AWS Identity and Access Management (IAM) можете да зададете разрешения и правила за балансиращите натоварването.

Производителност: AWS Global Accelerator може да се използва заедно с ELB за оптимизирано и последователно потребителско изживяване за вашите глобални потребители.

GCP – Балансиране на натоварването в облак:

Автоматично мащабиране: балансьорите на натоварването на Google работят ръка за ръка с групи екземпляри, за да позволят на вашите приложения да мащабират без ръчна намеса.

Балансиране на натоварването на базата на близост: Мрежовата инфраструктура на Google гарантира, че потребителските заявки се насочват към най-близкото глобално местоположение, осигурявайки ниска латентност.

Microsoft Azure Load Balancer:

Зони на наличност: Azure Load Balancer се интегрира със зони на наличност, за да осигури висока наличност.

Диагностика: Azure Monitor и Azure Security Center осигуряват задълбочена представа за операциите и сигурността на вашето приложение.

GSLB решения (Глобално балансиране на натоварването на сървъра):

Amazon Route 53:

Проверки на здравето: Освен DNS услугите, Route 53 може да наблюдава изправността на вашето приложение и да насочва трафика само към здрави крайни точки.

Геолокационно маршрутизиране: Route 53 ви позволява да маршрутизирате трафика въз основа на географското местоположение на вашите потребители.

Google Cloud Platform Load Balancer & Cloud DNS:

Anycast IP адреси: Намалява забавянето чрез маршрутизиране на потребителските заявки до най-близкото глобално местоположение.

Интегриран с GCP: Предлага тясна интеграция с други услуги на Google Cloud, като Cloud Armor, Cloud CDN и др.

Azure Traffic Manager:

Методи за маршрутизиране: Предоставя множество методи за маршрутизиране, включително производителност, претеглени, приоритетни и географски.

Мониторинг на крайни точки: Постоянно проверява изправността на вашите крайни точки и насочва трафика само към здравите.

### 3.3.2. Разгръщане на микро-услугите на системата

Съвременните стратегии за внедряване се поддържат от технологии за непрекъсната интеграция (CI), непрекъсната доставка (CD) и технологии за контейнеризация, като Docker и Kubernetes, които позволяват рационализиран и ефективен процес на внедряване. При непрекъсната интеграция разработчиците се сливат обратно към главния клон толкова често, колкото е възможно, обикновено няколко пъти на ден. Автоматизираните процедури за изграждане и тестване гарантират, че всяка интеграция е валидирана, като по този начин намалява вероятността от проблеми с интеграцията и ускорява цикъла на разработка. Тази практика за разработка се интегрира добре с Azure DevOps, изчерпателен набор от инструменти за разработка, услуги и функции, които улесняват ефективното сътрудничество и повишената продуктивност.

Интеграцията е допълнително подобрена от CD, който автоматично подготвя модификации за производствено пускане. Този непрекъснат поток от производствени актуализации намалява времето за изпълнение и ускорява цикъла на обратна връзка.

Azure предоставя услуги, които могат да помогнат за постигане на много неща, варирайки от обикновени, като създаване на ново приложение с база от данни – до по-развити  като създаване на работни потоци за непрекъсната интеграция (CI) и внедряване (CD). Това са само няколко примера за някои често срещани работни похвати. Много от тях трябва да бъдат създадени индивидуално, но облачната инфраструктура предлага  всичко това като услуги. Силата на облака е, че ресурсите са невероятно устойчиви, малко вероятно е аварийно да спрат работа, тъй като центровете за данни са разположени по целия свят, състоящи се от десетки хиляди сървъри. Ако един сървър се повреди, друг поема управлението. Един от най-убедителните аргументи в полза на облака е, че може да разширява мащаба на услуги и ресурси почти безкрайно, в определени моменти, като например "Черен Петък" или голяма маркетингова кампания с промоции и намаления на артикули. Също така, когато натоварването намалее, мащабът може да се намали до обикновените си параметри. Уважавани и опитни облачни доставчици като Microsoft разпознават моделите на използване на нормалните потребители и тези на злонамерените. Инфраструктурата е предпазена от най-често срещаните атаки. Интелигентни инструменти за наблюдение, алгоритми за обучение и изкуственият интелект предоставят възможност да откриват атаки в реално време.

Azure Pipelines, компонент на Azure DevOps, илюстриран на фиг 3.х, може ефективно да управлява това автоматизирано внедряване, като гарантира, че всяка компилация е готова за производство, след като бъде валидирана.



***Фиг. 3.х. Представя отделните компоненти и стъпки при процеса на интеграция и внедряване***

За изграждане, доставка и изпълнение на системи, изградени както като монолитни приложения, така и като ориентирани към услуги, се препоръчва използването на контейнеризирани технологии. **Контейнеризацията** е подход, в сферата на разработката на софтуер, при който кодът на приложение, всички негови зависимости и конфигурации са пакетирани в двоичен файл, наречен **изображение**. Изображенията са „шаблони“ само за четене и се съхраняват в **регистър**, който работи като хранилище или библиотека за изображения. Изображението се трансформира в работещ екземпляр на **контейнер**, който може да се стартира, спира, премества и изтрива. Създават се контейнери за различните части от приложението: уеб услуга, база данни, кеширане и др. Точно както транспортните контейнери позволяват транспортирането на стоки, независимо от товарите вътре, софтуерните контейнери се възприемат като стандартна единица за внедряване на софтуер, която може да съдържа различен код и зависимости. Контейнеризирането на софтуера дава възможност на разработчиците и ИТ специалистите автоматично да подновяват новите промени в различни среди. Контейнерите също така изолират приложенията едно от друго в споделена операционна система. Приложения се изпълняват върху хостът на контейнерите. От гледна точка на приложението, инстанцирането на изображение означава създаването на контейнер. Друго предимство на контейнеризацията е мащабируемостта. Разширяването става бързо: създават се нови контейнери за краткосрочни задачи. Контейнерите предлагат предимствата на изолация, преносимост, гъвкавост и контрол в целия жизнения цикъл на приложението.

Най-използваната и наложила се като стандарт технология е **Docker**. Това е проект с отворен код за автоматизиране на внедряването на приложения като преносими, самодостатъчни контейнери, които могат да работят локално или в облака. Също така е компания, която популяризира и развива тази технология. Docker контейнерите могат да работят върху Linux или Windows. Предимства за разработчиците са: ускорено въвеждане на нови програмисти в проекта, премахнете конфликтите в приложенията, актуализиране и мигриране на софтуера.

На фиг. 7 е представено сравнение между виртуална машина и Docker контейнер.

Graphical user interface

Description automatically generated

***Фиг. 7. Виртуални машини и Docker контейнерите***

Виртуалните машини включват приложението, необходимите библиотеки и пълна операционна система. Изисква пълна виртуализация повече ресурси, повече време за стартиране в сравнение.

Докер контейнерите включват приложението и всички негови зависимости. Те обаче споделят ядрото на ОС с други контейнери, изпълняващи се като изолирани процеси в потребителското пространство на хост операционната система. (с изключение на Hyper-V контейнери, където всеки контейнер работи вътре в специална виртуална машина).

Виртуалните машини имат три основни слоя: инфраструктура, хост, операционна система, Hypervisor и всички необходими библиотеки. Слоевете в Docker са инфраструктурата, ОС и двигател за контейнери, който поддържа изолация, но споделя основните услуги на ОС. Тъй като контейнерите изискват много по-малко ресурси (например не се нуждаят от пълна ОС), те са лесни за изпълнение, внедряване и започват бързо. Основната цел на изображението е да направи зависимостите еднакви в различните среди. Това гарантирана еднакво поведение на всички среди: локална среда, среда за разработка или продуктивна.

Docker осигурява значително предимство в контекста на внедряване на базирани на .NET Core приложения. Docker контейнерите капсулират приложението и неговите зависимости в самостоятелен изпълним пакет, гарантирайки, че приложението работи еднакво в различни изчислителни среди. В допълнение, Docker е съвместим с Azure и .NET Core, осигурявайки еднаквост между средата за разработка и производството.

Въпреки факта, че Docker опростява процеса на опаковане на приложения, решение като Kubernetes е необходимо за управление на тези контейнери, особено в мащаб. Kubernetes автоматизира разпределението и планирането на контейнери за приложения в клъстер, предоставя възможности за самовъзстановяване (като автоматично рестартиране на контейнери, повторно планиране и репликация) и улеснява хоризонталната мащабируемост.

Kubernetes предоставя операции от високо ниво, които да бъдат извършени чрез кода на самите микро-услуги. Pаботи с инструкции, които са прехвърлени върху облачните машини, така нареченият „клъстар“: набор от виртуални машини на Linux или Windows (възлови точки), върху които се разполагат самите приложения (но не директно). Kubernetes се грижи за маршрутизирането и логистика на микросервизните (най-често използван в тази архитектура). Azure Kubernetes Service (AKS) улеснява внедряването, администрирането и мащабирането на контейнеризирани .NET Core приложения, като използва силата на Kubernetes, като същевременно се възползва от удобството и функциите на Azure.

Следователно, чрез интегриране на CI/CD практики, използване на Docker за контейнеризация и използване на Kubernetes за оркестрация, внедряването на базирана на .NET Core система за управление на поръчки в Azure може да бъде много ефективна, надеждна и мащабируема.

Docker и Kubernetes са платформи, които се използват за улесняване на стратегии за внедряване, представени в таблица . deployment strategies: blue green deployment, rolling deployment, and canary deployment.

|  |  |
| --- | --- |
| Синьо-зелено внедряване  (blue-green deployment) | Тази стратегия включва две идентични производствени среди, „Синя“ и „Зелена“. По всяко време само една от тези среди е активна. Да кажем, че "Синята" среда е жива и обслужва трафик. Ако искате да внедрите нова версия на вашето приложение, вие го внедрявате в „зелена“ среда. След това можете да го тествате обстойно в тази отделна среда. След като сте доволни от стабилността и производителността, превключвате рутера към „зелена“ среда, която след това става активна. „Синята“ среда остава неактивна до следващото издание, като ви дава възможност за бързо връщане назад, ако е необходимо. |
| Постепенно внедряване  (rolling deployment) | При непрекъснато внедряване нова версия на приложението се внедрява постепенно в няколко екземпляра наведнъж, а не всички наведнъж, докато останалите екземпляри все още държат старата версия. Това позволява внимателно внедряване и също така поддържа наличността на услугата по време на внедряването. Ако възникнат проблеми, процесът на внедряване може да бъде спрян и ще бъдат засегнати само подгрупа от екземпляри. |
| Внедряване на Canary  (Canary release) | Наименувано след практиката на изпращане на канарче в мина за проверка за опасни газове, внедряването на Canary включва въвеждане на промяна в малка подгрупа от потребители, преди да я приложи към цялата инфраструктура. Целта е да се тества новото издание върху малка част от трафика, като се гарантира, че работи според очакванията, преди да се разпространи към по-широката потребителска база. Ако нещо се обърка, само екземплярите на Canary са засегнати и можете да върнете промените назад, без да засягате всички потребители. |

Всички тези стратегии предоставят различни начини за намаляване на риска и минимизиране на времето за престой по време на внедряване и могат да бъдат избрани въз основа на специфичните нужди и обстоятелства на вашия проект.

Когато става въпрос за стратегии за внедряване, Docker може да бъде полезен при синьо-зелени внедрявания, тъй като може да се настрои нов контейнер с новата версия на приложението и трафикът може да бъде насочен към него, когато е готов. При подвижни и канарични внедрявания Docker позволява лесно създаване и управление на необходимите отделни екземпляри. Kubernetes може да помогне, като управлява два различни комплекта подове (син и зелен). Сервизните обекти могат да се използват за превключване на трафика между двете среди. Kubernetes първоначално поддържа тази стратегия чрез своята стратегия за внедряване RollingUpdate. Позволява актуализиране на внедряване чрез постепенна замяна на стари модули с нови. Тази функция гарантира, че поне определен брой пакети са винаги налични по време на актуализацията и най-много определен брой пакети са създадени над желаното количество. Kubernetes може постепенно да прехвърлите трафика към новата версия на приложението и да наблюдавате ефективността. Ако новата версия работи добре, можете да продължите да пренасочвате трафика, докато новата версия не обработи всички заявки. Ако нещо се обърка, можете да насочите трафика обратно към по-старата, стабилна версия.

#### модели за производствено тестване

Моделите за производствено тестване са стратегии, използвани при разработването на софтуер, за да се гарантира, че софтуерът функционира според очакванията в производствена среда. Тези модели могат да помогнат за предотвратяване на софтуерни дефекти, подобряване на устойчивостта на системата и поддържане на качество и надеждност.

A/B тестването е един от тези модели, който в контекста на базирана в облака система за управление на поръчки дава възможност за вземане на решения, базирани на данни, като позволява едновременното внедряване на различни версии на системни подобрения или нови функции за подгрупи от потребители, като по този начин позволяващи сравнителни оценки на ефективността. Промените в потребителския интерфейс или агрегирането на данни от ERP, управление на автопарк и системи за наблюдение, например, могат да бъдат оценени, за да се определи техният ефект върху производителността на системата и потребителското изживяване. Тази процедура позволява оптимизирани модификации чрез разчитане на емпирични данни, а не на спекулации.



Фиг. A/B тестване

Chaos Engineering, когато се прилага към толкова важна система, подкрепя стремежа към устойчивост на системата. Тъй като тази система е център за безброй потоци от данни от различни страни и управлява чувствителна клиентска информация, трябва да се гарантира нейната надеждност. Chaos Engineering дава възможност за умишлено инжектиране на дефекти в системата по време на нейната работа, принуждавайки я в състояния на стрес и след това наблюдавайки реакциите на системата. Това може да включва симулиране на грешки при въвеждане на данни от системи за управление на флота в една страна или симулиране на латентност при предаване на данни от ERP системи и оценка на способността на системата да се справя елегантно с тези повреди. Чрез проактивно идентифициране на уязвимостите на системата в контролирана среда, екипите могат проактивно да измислят решения за подобряване на устойчивостта и устойчивостта на системата. Хармоничното интегриране на A/B тестване и Chaos Engineering позволява на базираната в облака система за управление на поръчките не само да оцелее в динамичния, взискателен пейзаж на глобалното управление на веригата за доставки, но и да процъфтява.



**Фиг.** Chaos Engineering,

Моделът за внедряване на Canary Release може да е особено подходящ за базирана в облак система за управление на поръчки. Тази стратегия включва прогресивно прилагане на промени към подгрупа от потребители, преди да ги приложите към цялата система. Чрез сегментиране на внедряването по този начин е възможно да се наблюдава въздействието на промените в системата в реално време, като по този начин се намалява рискът от повсеместно прекъсване. Той предлага цялостна тестова среда за нови функции или модификации на системата за управление на поръчки, планиране на ресурсите на предприятието (ERP), управление на автопарка и системи за наблюдение, което позволява на екипа да идентифицира потенциални проблеми, преди те да засегнат всички крайни потребители.

Прилагането на принципите на Chaos Engineering може допълнително да подобри устойчивостта на системата поради нейното глобално разпространение и управлението на данни с високи залози. Чрез умишлено въвеждане на грешки екипът може проактивно да идентифицира и адресира уязвимостите на системата, като по този начин повишава устойчивостта и надеждността на системата като цяло.

В заключение, въпреки факта, че всеки от гореспоменатите модели има предимства, комбинацията от Canary Release и Chaos Engineering може да осигури балансиран подход за поддържане на стабилност, като същевременно непрекъснато подобрява системата за управление на поръчки, базирана в облака.

### 3.3.4. Мониторинг и системен дневник

Ефективното водене на системен дневник и мониторинга са основни компоненти на всяка система, базирана в облачна среда. Разбирането на техните сложни задължения и огромния набор от налични инструменти е от важно значение за осигуряване на оптимална функционалност на системата. Мониторингът на инфраструктурата и мониторингът на приложенията са двете основни категории. Наблюдението на инфраструктурата включва оценка и контролиране на системни ресурси като процесор, памет, дисково пространство и мрежов трафик. Поради своите изчерпателни възможности за наблюдение на ресурси и капацитет за идентифициране на ограничения, инструменти като Nagios са много подходящи за тази цел. За разлика от това, наблюдението на приложения се фокусира върху функционалността и ефикасността на приложението в системата. Той разглежда аспекти като време за реакция, честота на грешки и проследяване на транзакции, които са от решаващо значение за безупречното потребителско изживяване на системата за управление на поръчки. Всеобхватни решения за наблюдение на производителността на приложенията се предоставят от продукти като New Relic и Azure Application Insights. Те предоставят прозрения в реално време, улесняват диагностиката на проблема и минимизират забавянето. Въпреки че често се пренебрегва, поддържането на системен дневник е мощно допълнение към мониторинга. Той помага на разработчиците при проследяване на грешки и разбиране на последователността от събития, довели до повреда на системата. ELK (Elasticsearch, Logstash, Kibana) Stack е система за регистриране с отворен код, която събира регистрационни файлове от различни източници, съхранява ги за бързо извличане (Elasticsearch), обработва ги и ги трансформира (Logstash) и след това ги визуализира в удобен за потребителя маниер (Кибана). Това използване на ресурсите полага основата за стабилна, надеждна и ефективна система за управление на поръчките. ELK позволява не само бързо идентифициране на системни грешки, но и проактивно решаване на проблеми, като по този начин улеснява предоставянето на качествен продукт на крайния потребител.

Основните разлики между регистрирането и мониторинга, два термина, които често се бъркат. Регистрирането включва проследяване на действията на потребителите и поведението на системата чрез документиране на дейността на системата и записване на действията на потребителите. Той е незаменим за системен анализ, откриване и проверка на грешки и одит. Тази практика помага при проследяване на поведението на потребителите, което е от съществено значение за регулаторни цели в индустрии като финанси или при възпроизвеждане на специфични потребителски действия, които водят до грешки. Регистрирането основно документира грешките, като отбелязва всеки съответен детайл, като клеймо за време, тип грешка, проследяване на стека, вътрешни изключения и т.н.

Мониторингът, от друга страна, изследва показатели, свързани с инфраструктурата (като CPU, RAM и използване на диска) и показатели, свързани с приложението (като заявки на минута и поръчки на ден). Тези показатели се представят на потребителите чрез изчерпателни дисплеи. Важен аспект на мониторинга е системата за предупреждение, която задейства предупреждения, когато специфични показатели се отклоняват от нормалния си диапазон, като например когато използването на процесора надвишава 90% или средното време за реакция надвишава пет секунди. След това системата уведомява определена група за разрешаване на проблема. Въпреки че регистрирането и наблюдението изпълняват различни функции, и двете са от съществено значение за осигуряване на надеждността и стабилността на системата за микроуслуги.

Мониторингът е незаменим за администрирането и управлението на приложения, особено в контекста на API. Този раздел обсъжда целта на мониторинга, неговото значение и параметрите, които трябва да се наблюдават за оптимална производителност на системата.

Цел на мониторинга:

Предвиждане и превантивни мерки: Идентифицирайте потенциалните проблеми преди тяхната ескалация.

Диагностицирайте проблемите веднага щом станат очевидни.

Оперативен надзор: Придобийте задълбочено разбиране на ефикасността и моделите на използване на API.

Значение на наблюдението:

Очаквания на клиентите: В днешната дигитална ера клиентите очакват вашият API да осигурява постоянна производителност и достъпност.

Защита на функционалността: Необходими са надеждни механизми за наблюдение, за да се гарантира, че API функционира оптимално и изпълнява своите цели за ниво на обслужване.

Важни показатели за проследяване в API екосистема:

Заявки за секунда: Този показател дава представа за текущия трафик и търсене на API.

Наблюдението на броя на повреди може да помогне за ранното идентифициране на повтарящи се проблеми или уязвимости.

Закъснение: Оценяването на времето за реакция на API дава представа за неговата ефективност и производителност.

Брой потребители: Мониторингът на броя на активните потребители може да даде представа за търсенето и популярността на системата.

Брой сесии: Това предоставя общ преглед на потребителското взаимодействие и ангажираност с API.

Разбирането на географското разпределение на потребителите може да помогне за оптимизиране на местоположението на сървъра и подобряване на потребителското изживяване.

Наблюдението на използването на процесора може да разкрие потенциални ограничения или области, изискващи оптимизация.

Използване на RAM: Редовното наблюдение на използването на паметта гарантира, че системата не е претоварена и работи ефективно.

## 3.4. Практики и модели за сигурност

В ерата на дигиталната трансформация сигурността се очертава като важен аспект при работа с данни за управление на поръчки. Трябва да се приложат подходящи мерки за сигурност. Всяка микроуслуга трябва да има свой механизъм за удостоверяване, а API трябва да бъдат защитени с помощта на токени или сертификати. Внедряването на сигурност на мрежово ниво, като защитни стени или системи за откриване на проникване (IDS), може да помогне за наблюдение и смекчаване на потенциални заплахи. Облачни платформи като Azure предоставят вградени механизми за сигурност като Azure Security Center и Azure Active Directory. Тези инструменти предлагат управление на сигурността и защита от заплахи за облачна инфраструктура и работни натоварвания. Грешни конфигурации, незащитени интерфейси са потенциални уязвимости, които могат да компрометират сигурността на цялата система. Редовните одити, спазването на принципа за най-малко привилегии и използването на собствените инструменти за сигурност на Azure могат да намалят тези рискове. Тъй като дигиталната среда продължава да се развива, важността на киберсигурността не може да бъде подценявана. Капанията трябва да я приоритизира, за да поддържат доверието и увереността на своите клиенти.

### 3.4.1. Същност и главни точки на защитата

1. Моделиране на заплахи:

Нашата глобална OMS ще бъде подложена на строго моделиране на заплахи, за да идентифицира потенциални заплахи в различни държави и точки на интеграция, като ERP и системи за управление на автопарк. Ключови въпроси като естеството на обработваните данни, потенциални уязвимости, стратегии за смекчаване и валидиране на мерките за сигурност ще ръководят тази фаза.

2. Сигурна архитектура:

Сигурност на мрежата и платформата:

Системата се хоства на облачна платформа със специални подмрежи за всяка микроуслуга. Групите за сигурност, правилата за защитна стена и IPS системите са конфигурирани, за да гарантират, че комуникацията между услугите е защитена и изолирана.

Удостоверяване:

Използва механизъм за удостоверяване на трета страна като Active Directory или Okta.

Хибридното потребителско хранилище интегрира различни потребителски бази данни, специфични за страната, като същевременно осигурява универсален достъп.

Съвременните протоколи за удостоверяване като OAuth2 улесняват сигурната комуникация между услугите и външната система.

Упълномощаване:

Внедрява се ролев контрол на достъпа (RBAC). Докато междуорганизационните роли се намират в машината за удостоверяване, специфичните роли за отделни микроуслуги се управляват в рамките на съответните услуги.

Всяка микроуслуга ще има собствен механизъм за оторизация, който гарантира, че получава само данни, които има право да обработва.

Сигурна комуникация:

Цялата комуникация между услугите се извършва през актуализирани версии на TLS, осигурявайки криптиране на данни при пренос.

VPN мрежите или специалните връзки се интегрират със системи на трети страни в различни страни по сигурен начин.

Код за сигурност:

Екипите за разработка се придържат към най-добрите практики от OWASP и редовно преминават обучение за безопасно кодиране.

Защитени данни:

Критични данни, като подробности за поръчката, информация за плащане и лични данни за клиента, са криптирани в покой.

В зависимост от чувствителността, стратегиите за криптиране ще използват или вградените механизми на доставчика на облак, или решения на трети страни.

Управлението на ключовете се обработва сигурно с помощта на решения като Azure KeyVault или AWS KMS.

Регистриране и наблюдение:

Решенията за регистриране улавят всички събития, свързани със сигурността, с настроено предупреждение в реално време за аномалии като високи нива на заявки или повтарящи се грешки при валидиране.

3. SDLC:

Практиките за защитен жизнен цикъл на разработка гарантират, че всяка микроуслуга, независимо дали е обработка на поръчки, ERP интеграция или управление на автопарк, преминава през оценки на сигурността по време на проектиране, разработка и внедряване.

4. Тестване:

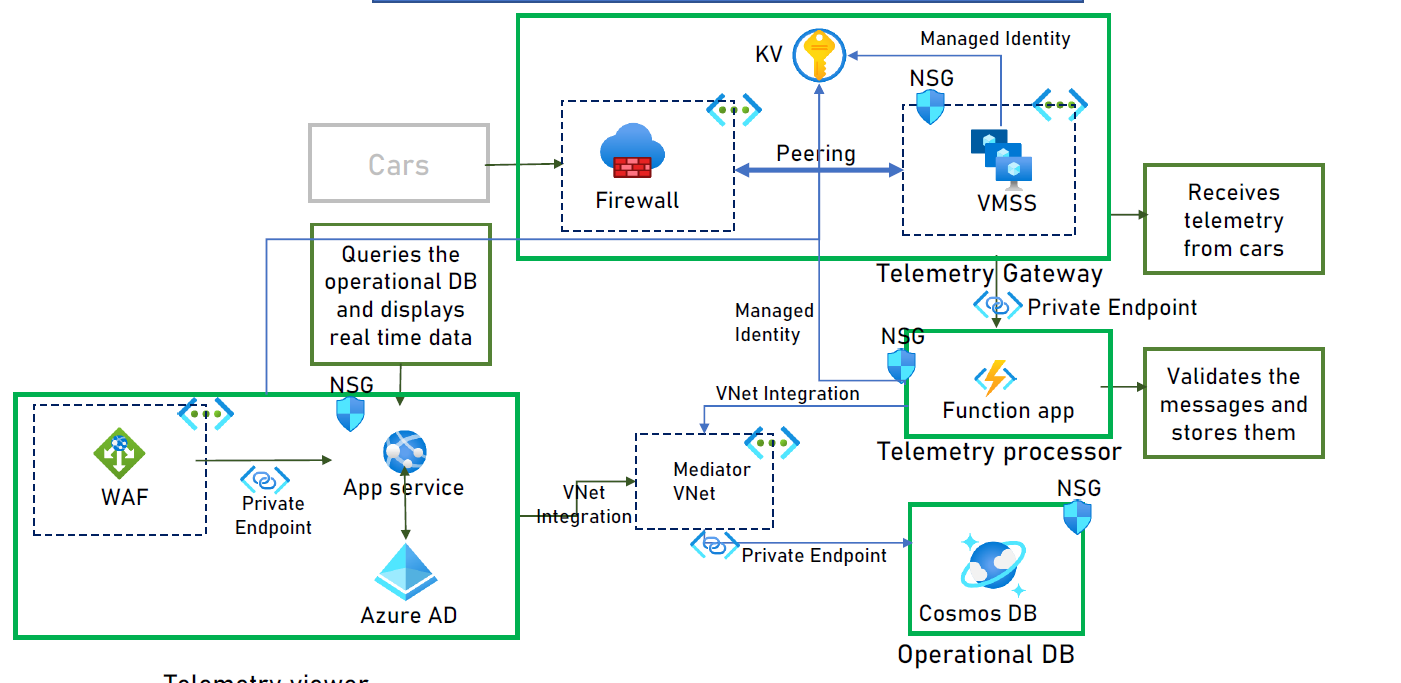
Провеждат се редовни тестове за проникване, както черна кутия, така и бяла кутия, за да се потвърди състоянието на сигурността на OMS и неговите интеграции. Тестовете за натоварване също гарантират, че системата остава сигурна при натоварване.

5. Производство:

След внедряването системата се подлага на периодични прегледи на сигурността. Всички констатации от тестове за проникване или други оценки водят до усъвършенстване на архитектурата.

В тази глобална OMS безпроблемната и сигурна интеграция на ERP, управление на автопаркове и системи за проследяване в множество страни е от първостепенно значение. Използването на микроуслуги гарантира мащабируемост и изолация, докато строгите практики за сигурност гарантират безопасността на данните и операциите.

### 3.4.1. Технологични инструменти за киберзащита



Фиг.

### 3.4.2. Разкриване на кибер атаки чрез центърът за сигурност

Full Blog: <https://azure.microsoft.com/en-us/blog/how-azure-security-center-unveils-suspicious-powershell-attack/?cdn=disable>

SQL Brute Force attack: <https://azure.microsoft.com/en-us/blog/how-azure-security-center-helps-reveal-a-cyberattack/>

Bitcoin mining attack: <https://azure.microsoft.com/en-us/blog/how-azure-security-center-detects-a-bitcoin-mining-attack/>

DDoS attack using cyber threat intelligence: <https://azure.microsoft.com/en-us/blog/how-azure-security-center-detects-ddos-attack-using-cyber-threat-intelligence/>

Good applications being used maliciously: <https://azure.microsoft.com/en-us/blog/how-azure-security-center-aids-in-detecting-good-applications-being-used-maliciously/>

### Издаване на удостоверение

Избирането на базирани в облак сървъри за идентичност за микроуслуги .NET Core, хоствани на Azure, има няколко опции, които може да искате да разгледате. Този анализ ще се съсредоточи върху Azure Active Directory (Azure AD), IdentityServer, Auth0 и Okta, тъй като те са често използваните сървъри за самоличност в индустрията.

Azure Active Directory (Azure AD): Това е естествен избор, ако хоствате своите .NET Core микроуслуги на Azure. Azure AD предлага стабилна поддръжка за различни протоколи за удостоверяване, включително OAuth 2.0 и OpenID Connect, които са от съществено значение за внедряването на базирани на микроуслуги архитектури. Освен това Azure AD се интегрира добре с други услуги на Azure и предоставя изчерпателен набор от функции за управление на идентичност и достъп. Можете да използвате Azure AD B2C за приложения, ориентирани към потребителите, или Azure AD B2B за приложения, насочени към бизнеса.

IdentityServer: IdentityServer е рамка с отворен код за ASP.NET Core, която позволява създаването на доставчик на OpenID Connect. IdentityServer е добър избор, ако имате нужда от голяма гъвкавост или имате много специфични изисквания, които не се поддържат от готови продукти като Azure AD, Auth0 или Okta. Той също така поддържа всички потоци в OAuth 2.0 и OpenID Connect веднага. Едно нещо, което трябва да имате предвид е, че вие ще отговаряте за хостването и управлението на вашето копие на IdentityServer, което може да е недостатък, ако предпочитате управлявано решение.

Auth0: Auth0 е базирано на облак решение, което предлага проста, мащабируема и много адаптивна платформа за управление на самоличността. Auth0 предлага обширни опции за персонализиране, с налични SDK за различни езици и платформи (включително .NET Core) и се интегрира с много различни доставчици на социална идентичност. Той също така поддържа необходимите протоколи OAuth 2.0 и OpenID Connect. Основният му недостатък е цената; въпреки че предлага безплатно ниво, използването над това бързо става скъпо.

Okta: Okta е друго базирано на облак решение, известно със силния си акцент върху корпоративната сигурност. Той предоставя широк набор от функции, включително единично влизане, многофакторно удостоверяване, управление на жизнения цикъл и функция за управление на достъпа до API, което го прави подходящ за управление на самоличности в архитектура на микроуслуги. Подобно на Auth0, Okta предлага удобна за разработчици платформа, но също така може да стане доста скъпа, когато употребата ви се увеличи.

TrustBuilder е компания, която предлага платформа за управление на самоличността и достъпа (IAM). Пакетът на TrustBuilder позволява на бизнеса да контролира кой има достъп до техните цифрови ресурси и да управлява ефективно тези самоличности. Неговите решения включват функции за удостоверяване на потребителя, оторизация, единично влизане (SSO) и федерация, между другото.

Ping Identity е подобен доставчик в това пространство, с платформа, предназначена да осигури защитен достъп до всяка услуга или приложение от всяко устройство. Решенията на Ping Identity включват многофакторно удостоверяване, единично влизане, сигурност на достъпа и предоставяне на потребители. Те също така предлагат решения специално за API и микроуслуги.

Всяка от тези опции има своите силни и слаби страни и най-добрият избор ще зависи от вашите специфични нужди, включително фактори като естеството на вашето приложение (насочено към потребителите или B2B), вашия бюджет, вашите изисквания за сигурност и степента на персонализиране, което трябва. Всички тези доставчици ще работят добре с .NET Core микроуслуги и Azure; изборът зависи основно от вашия конкретен случай на употреба и предпочитания.

Aнализът на тези опции, показва, че IdentityServer се очертава като най-добрият избор за конкретния случай на употреба, като се има предвид неговата съвместимост с .NET Core микроуслуги на Azure.

Има няколко причини, поради които IdentityServer е особено подходящ за вашите нужди:

Гъвкавост и контрол: Като рамка с отворен код, IdentityServer предоставя гъвкавостта за проектиране и внедряване на решение за управление на самоличността, което отговаря на вашите уникални бизнес изисквания. Съвместимостта му с .NET Core го прави добра опция, когато стекът за разработка е съсредоточен върху .NET.

Поддръжка на протокол: IdentityServer поддържа напълно всички потоци в OAuth 2.0 и OpenID Connect, което го прави стабилно решение за установяване на защитена комуникация между вашите микроуслуги и техните клиенти.

Ефективност на разходите: За разлика от услугите, базирани на облака, като Auth0 и Okta, които могат бързо да станат скъпи с увеличаване на използването, IdentityServer включва само разходите, свързани с хостинг и поддръжка на сървъра, осигурявайки по-предсказуема структура на разходите.

Интеграция: IdentityServer може да бъде ефективно интегриран с Azure, осигурявайки безпроблемно изживяване при разработка и внедряване на платформата. Освен това ви отваря възможности да се възползвате от други услуги на Azure, ако е необходимо.

Важно да се отбележи, че управлението и поддържането на екземпляр на IdentityServer наистина изисква известно усилие и опит, за организация с опит в .NET Core и изисквания, които се нуждаят от висока степен на персонализиране, това е силен, рентабилен избор. Способността да контролирате и управлявате собствената си услуга за идентичност е значително предимство, което ви позволява да се адаптирате към променящите се нужди и изисквания с течение на времето.

#### Съхраняване на потребители и идентификационни данни в локална база данни

OpenID Connect и OAuth не обработват механиката на удостоверяване на потребителя. Спецификацията определя дали потребителят е удостоверен от сървъра за оторизация (доставчик на самоличност или IDP), без да указва как трябва да се случи този процес. Задачата на IDP е да предостави на клиентското приложение доказателство за потребителска идентичност (чрез токен за самоличност), независимо от използвания метод за удостоверяване.

Методите за удостоверяване на потребителите варират от традиционни системи за потребителско име/парола до биометрични методи, доказателства за собственост (като сертификати за USB ключове или кодове на приложения за смартфони) и удостоверяване на транзакции въз основа на несъответствия в IP. Съвременните системи често изискват многофакторно удостоверяване за повишена сигурност.

Идентификационните данни, като доказателство за самоличност, могат да се съхраняват локално или отдалечено. Локално съхранените идентификационни данни могат да бъдат удостоверени чрез Active Directory, докато отдалеченото съхранение включва използване на идентификационни данни на трети страни (Microsoft, Google, Facebook). Това налага централен доставчик на идентичност, който да управлява промените в средствата за удостоверяване и подходите във всички клиентски приложения.

Схемата на потребителската база данни може да започне с таблица User и таблица UserClaims, свързани помежду си във връзка "един към много". Таблицата User съхранява данни, свързани с локалното удостоверяване, докато таблицата UserClaims съдържа други стойности, свързани с потребителя. И двете таблици използват поле ConcurrencyStamp за управление на потенциални проблеми с паралелността, като по този начин подобряват целостта на данните.

Накратко, OpenID Connect не се занимава с това как крайният потребител се удостоверява на ниво IDP, а се фокусира върху предоставянето на доказателство за потребителска идентичност на клиентското приложение, докато методите и съхранението на потребителското удостоверяване остават гъвкави, за да отговарят на различни нужди.

#### Управление на потребители

внедряването на екрани за самоуправление на самоличността, поставянето им в системата и прилагането на техники за солиране, хеширане и разтягане на ключове за подобряване на сигурността на паролите. Тази дискусия може да помогне при създаването на системи за регистрация на потребители, като се фокусира върху това къде да се интегрира тази функционалност, която може да варира в зависимост от системната архитектура и нужди. От гледна точка на сигурността всички опции като създаване на отделни уеб приложения или интегриране на функционалност на ниво доставчик на идентичност са жизнеспособни. Решението трябва да вземе предвид фактори като мащабируемост, възможност за внедряване и сложност на разработката.

Ние също се задълбочаваме в сигурността на паролите, като подчертаваме, че паролите с ясен текст представляват значителен риск. Ние предлагаме паролите да преминат през процес от три стъпки преди съхранение: осоляване, хеширане и разтягане на ключа. Осоляването включва прикачване на криптографски произволна част от данните към паролата преди хеширане, осигурявайки високо ниво на произволност и непредсказуемост, което я прави по-сигурна и предпазва от атаки в речника. Хеширането трансформира паролата в еднопосочен криптиран низ, известен като хеширана парола, осигурявайки защита срещу декриптиране и допълнително укрепване срещу атаки в справочна таблица и дъгова таблица.

Ключовото разтягане, последната стъпка, включва хеширане на солената парола многократно, обезсърчаване на груби атаки чрез значително увеличаване на изчислителното време за всяко отгатване на парола. Разтягането на ключовете също удължава незначително времето за влизане на потребителя, но това е малък компромис за подобрена сигурност. Алгоритми като PBKDF2 и Argon 2 комбинират тези стъпки в една функция, но тяхното използване трябва редовно да се оценява и коригира според напредъка в изчислителната мощност.

Описаните концепции са внедрени в демонстрация на ASP.NET Core, която опростява целия процес, демонстрирайки достъпността на стабилни решения за защита на пароли.

управление на потребители в дигитални системи, с акцент върху потвърждението на имейли, нулиране на сигурни пароли и стратегии за борба с грубата сила. Препоръчваме да съхранявате и проверявате имейлите на потребителите, за да установите надеждна комуникация и да улесните повторното задаване на пароли. Този процес включва изпращане на линк за активиране на предоставения адрес, който става валиден за ограничен период от време. Ако потребителите действат незабавно, ние потвърждаваме имейл адреса и активираме техните акаунти. От жизненоважно значение е тази информация да се съхранява сигурно в потребителската таблица заедно с код за сигурност, свързан с процеса на активиране и времето му на изтичане.

Ние признаваме, че управлението на потребителите варира в различните системи, така че предлагаме допълнителни екрани и функционалности за работа с външни доставчици и многофакторно удостоверяване. Наблягаме на проверката на новите имейл адреси при настъпване на промени и съветваме прилагането на функция за повторно изпращане на връзка за връзки за активиране или потвърждение. Нулирането на парола винаги трябва да включва потвърждаване на самоличността на потребителя, за предпочитане чрез връзка, изпратена до потвърдения имейл.

Автоматичното блокиране на потребители след неуспешни опити за влизане обикновено не се препоръчва поради уязвимостта му към злоупотреби, водещи до атаки за отказ на услуга. Вместо това предлагаме да използваме разтягане на ключове, за да удължим времето за удостоверяване и да обезсърчим атаките с груба сила. Алтернативно, включването на CAPTCHA може да добави още едно ниво на сигурност.

Политиките за пароли трябва да се развият от традиционни практики на сложни знаци и редовни промени към насърчаване на дълги пароли или фрази за достъп. Обезсърчаването на използването на често използвани пароли и насърчаването на мениджъри на пароли може да увеличи сигурността. Допълнителни фактори за удостоверяване също трябва да бъдат взети предвид.

Като цяло, внедряването на персонализиран код за екрани за управление на потребителите, които взаимодействат с услугата за взаимодействие на IdentityServer, като същевременно поддържа сигурността на паролата чрез солиране, хеширане и разтягане на ключове, остава от решаващо значение. Проверката на имейл е неразделна част от активирането на акаунта и сигурното нулиране на паролата, като същевременно избягването на автоматично блокиране и насърчаването на усъвършенствани политики за пароли подобрява сигурността.

#### Multi-factor Authentication

Многофакторното удостоверяване (MFA) съчетава два или повече фактора за удостоверяване като това, което знаете (пароли, пин кодове), какво притежавате (смартфони, цифрови пропуски) и какво сте (биометрични данни). Два по-рядко срещани фактора са какво правите (жестове) и къде се намирате (IP адрес). MFA е от решаващо значение за проверка на самоличността; двуфакторното удостоверяване (2FA) е подгрупа на MFA, използвайки точно два фактора. Използването на едни и същи типове фактори, като например две двойки потребителско име/парола, е неефективно. Типичен пример за MFA е тегленето на пари в брой от банкомат, което изисква нещо, което знаете (ПИН код) и нещо, което притежавате (банкова карта).

Еднократните пароли (OTP) са уникални пароли, валидни за една сесия за влизане, предлагащи по-силна защита срещу повторни атаки, отколкото обикновените пароли. Те могат да бъдат изпратени на потребителите по имейл след валидна комбинация от потребителско име/парола, но това не представлява истинска MFA, тъй като достъпът до имейл не доказва категорично притежание на устройство. Националният институт за стандарти и технологии (NIST) предлага по-сигурен метод за удостоверяване: меки еднократни пароли, генерирани от приложения за удостоверяване, подобряващи само удостоверяването с потребителско име/парола.

Приложенията за удостоверяване като Google Authenticator или Microsoft Authenticator генерират OTP на устройство. Два преобладаващи метода за генериране на OTP са HMAC-базиран OTP (HOTP) и Time-based OTP (TOTP). HOTP генерира OTP на базата на брояч на събития, докато TOTP базира движещия се фактор на времето, осигурявайки краткотрайни OTP за подобрена сигурност.

TOTP се генерира от тайна, споделена от клиента и сървъра. Тайната обикновено се предава на клиентското устройство ръчно или чрез сканиране на QR код, за да се избегнат рискове за сигурността. Успешното генериране на TOTP и удостоверяване може да доведе до цялостно MFA решение, когато е съчетано с удостоверяване на потребителско име/парола.

Обмислянето на това кога и къде да се изискват OTP е от съществено значение. Опциите включват изискване на OTP за всяко влизане, само за конкретни потребители или клиенти или само когато потребителите използват локални идентификационни данни. В идеалния случай доставчиците трети страни биха ни информирали за методите за удостоверяване на потребителя, което ни позволява да поискаме допълнителен фактор, когато е необходимо. Демото обаче ще го приложи за локални идентификационни данни за простота.

#### IdentityServer to Azure

IdentityServer, уеб приложение на ASP.NET Core, може да бъде внедрено в Azure чрез множество методи като Visual Studio или CI/CD конвейер. Въпреки това се появяват предизвикателства при управлението на конфигурацията и оперативните данни, особено в сценарии, включващи балансиране на натоварването и множество сървърни екземпляри. Понастоящем такива данни са твърдо кодирани или се съхраняват в паметта, което не е ефективно за скалируемост. Следователно трябва да използваме база данни SQL Azure за централно съхраняване на конфигурационни и оперативни данни.

Освен това трябва да обърнем внимание на защитата на данните, тъй като настройката по подразбиране е предназначена за внедряване на една машина и не би била достатъчна в увеличена среда с балансирано натоварване. Централизираното хранилище за ключове за защита на данните е от съществено значение.

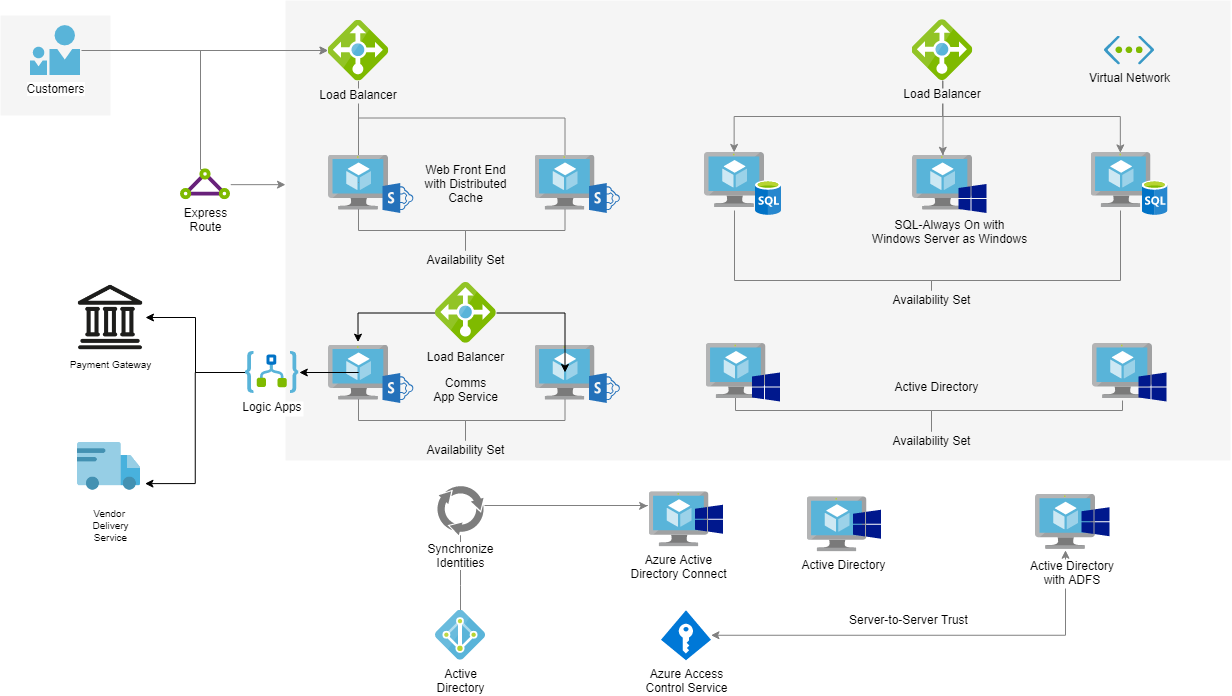
Ключовете за подписване на токени също трябва да се управляват централно, тъй като различните копия на сървъра може да притежават различни идентификационни данни за подписване, което води до проблеми с валидирането. За да разрешите това, Azure Key Vault може да се използва за съхранение на сертификати.

Трябва също да се обърне внимание на мидълуера на препратените заглавки поради потенциалната неяснота на информацията за заявката от прокси сървъри, балансиращи натоварването и други мрежови устройства.

И накрая, лиценз за IdentityServer, дори за неговото безплатно издание на общността, е необходим при внедряване в производствена среда.

За тези задачи се предполагат основни познания за Azure, включително познаване на общи ресурси и основно управление на достъпа. Целта е да се осигури мащабируемост и надеждност, като същевременно се смекчат опасенията, свързани с IdentityServer.

#### Azure Solution Architecture Diagram



#### Geo routing

