Capitolul 8. Propunerea arhitecturii sistemului informatic

CRITERII PENTRU ARGUMENTAREA ARHITECTURII

Introducere

Acest document prezinta succint criteriile care stau la baza evaluării si argumentării arhitecturilor software existente si propuse pentru implementare. Aceste criterii pot fi văzute ca si calități necesare ale arhitecturilor, pentru a facilita soluții tehnice optime.

Obiective

Obiectivul principal al arhitecturii sistemelor de informatizare, parte a „Strategiei de Informatizare a Județului Cluj", este sa faciliteze implementarea si dezvoltarea incrementala a diverselor servicii si produse software ce compun viziunea de informatizare, oferind o fundație solida pentru diversificarea si îmbunătățirea serviciilor oferite, prin intermediul software.

Aceeași importanta o au însă si celelalte obiective:

- Facilitarea calității funcționale si non-funcționale (securitate, performanta)
- Facilitarea implementării folosind „best-practices" la nivel de industrie
- Facilitarea dezvoltării colaborative, incrementale si evolutive
- Optimizarea costurilor de dezvoltare, extindere si întreținere

Calități esențiale ale arhitecturii

Portabilitate

- Arhitectura evaluata / propusa **TREBUIE** sa permită migrarea pe alte platforme (ex. de pe Windows pe Linux) in funcție de infrastructura disponibila
- Arhitectura evaluata / propusa **TREBUIE** sa permită migrarea pe platforme Cloud, atât private cat si publice sau hibride
- Arhitectura evaluata / propusa **TREBUIE** sa limiteze pe cat posibil dependența de sisteme terțe comerciale
- Arhitectura evaluata / propusa **TREBUIE** sa ofere posibilitatea de a inter-schimba anumite dependențe / subsisteme prin configurare

Performanta

- Arhitectura evaluata / propusa **TREBUIE** sa asigure latenta redusa in orice procesare conform cerintelor Non-Functionale (Non Functional Requirements)
- Arhitectura evaluata / propusa **TREBUIE** sa asigure un timp de răspuns conform cerințelor Non-Funcționale (Non Functional Requirements)
- Arhitectura evaluata / propusa **TREBUIE** sa permită execuția frecventa a testelor de performanta

Fiabilitate

- Arhitectura evaluata / propusa **TREBUIE** sa asigure rezistenta la erori induse de utilizator
- Arhitectura evaluata / propusa **TREBUIE** sa contina capabilități de detecție/ prevenție si revenire după erorile sistem

Scalabilitate

- Arhitectura evaluata / propusa **TREBUIE** sa asigure ușurința de a adaugă noi noduri de procesare
- Arhitectura evaluata / propusa **TREBUIE** sa asigure ușurința de a distribui nivelul de încărcare către noile noduri (load balancing)
- Arhitectura evaluata / propusa **TREBUIE** sa limiteze efectele componentelor comerciale asupra scalabilității.
- Arhitectura evaluata / propusa **TREBUIE** sa permită scalarea independenta a subsistemelor in funcție de nevoie

Securitate

- Arhitectura evaluata / propusa **TREBUIE** sa permită cu ușurința limitarea zonelor de potențial risc (subsisteme, componente, etc)
- Arhitectura evaluata / propusa **TREBUIE** sa asigure o securitate implicita a sistemului
- Arhitectura evaluata / propusa **TREBUIE** sa maximizeze utilizarea practicilor de securitate
- Arhitectura evaluata / propusa **TREBUIE** sa permită existenta si execuția frecventa a testelor de securitate

- Arhitectura evaluata / propusa **TREBUIE** sa conțină practici de audit si prevenție

Testabilitate

- Arhitectura evaluata / propusa **TREBUIE** sa permită posibilitatea de testare unitara automata
- Arhitectura evaluata / propusa **TREBUIE** sa permită posibilitatea de testare automata la nivel de modul
- Arhitectura evaluata / propusa **TREBUIE** sa permită posibilitatea de testare automata integrata (end to end)
- Arhitectura evaluata / propusa **TREBUIE** sa permită posibilitatea de a utiliza servicii/ componente mock in toate 3 nivelele de testare
- Arhitectura evaluata / propusa **TREBUIE** sa permită posibilitatea de a automatiza testarea interfeței cu utilizatorul

Interoperabilitate

- Arhitectura evaluata / propusa **TREBUIE** sa folosească la maxim standarde deschise de date/ comunicare/ protocoale (ex. HTTP, REST, AMQP, JSON, etc) si sa reducă formatele proprietare
- Arhitectura evaluata / propusa **TREBUIE** sa asigure interfețe de programare/ extensie (API's) care sa respecte standardele si bunele practici din industrie
- Arhitectura evaluata / propusa **TREBUIE** sa permită dezvoltarea cu ușurința a punctelor de integrare / extensie (ex. web hooks, events, etc)

Standardizare

- Arhitectura evaluata / propusa **TREBUIE** sa urmeze pe cat posibil modele/ șabloane (patterns) arhitecturale si de design devenite standard in industrie (Microservices, Event Drivern Arhitectures, CQRS, etc)
- Arhitectura evaluata / propusa **TREBUIE** sa asigure un format unitar ar interfețelor de programare/ extensie (API's) folosind bunele practici agreate in industrie

- Arhitectura evaluata / propusa **TREBUIE** sa limiteze pe cat posibil numărul de sisteme terțe/ tehnologii diferite utilizate in diversele aspecte funcționale (ex. baze de date, tipuri de servere, modalitati de deployment, etc.) si sa folosească un set agreat / prestabilit.
- Cu toate acestea, arhitectura evaluata / propusa **POATE** sa utilizeze sisteme terțe diferite atunci când funcționalitatea implementata in anumite componente/ servicii justifica utilizarea unui sistem terț specializat (ex. este recomanda chiar ca arhitectura sa folosească capabilitățile sistemelor de baze de date specializate - Document Database, Graph Database, Key Value Store – atunci când capabilitățile oferite de aceste sisteme facilitează dezvoltarea eficienta a anumitor capabilități)

Transparenta

- Arhitectura evaluata / propusa **TREBUIE** sa asigure urmărirea facila a tuturor erorilor
- Arhitectura evaluata / propusa **TREBUIE** sa asigure detaliile necesare in cazul erorilor, prin capturarea centralizata a așa numitor „stack traces”
- Arhitectura evaluata / propusa **TREBUIE** sa permită comutarea nivelelor de logging si de debug in sistemele si serviciile utilizate
- Arhitectura evaluata / propusa **TREBUIE** sa asigure vizibilitate si transparenta in privința tranzacțiilor din sistem (atât in curs, cat si finalizate sau in stadii de eroare)

Eleganta (simplitate si eficacitate sporita)

- Arhitectura evaluata / propusa **TREBUIE** sa aibă consecventa in soluțiile aplicate
- Arhitectura evaluata / propusa **TREBUIE** sa fie extensibila fără sa fie nevoie de modificări in sistemele existente, prin respectarea principiilor *Open Closed* si *Single Responsibility*
- Arhitectura evaluata / propusa **TREBUIE** sa aibă o coeziune ridicata si cuplare redusa intre componentele / subsistemele proprii.

Operabilitate

- Arhitectura evaluata / propusa **TREBUIE** sa utilizeze concepte de tip „Infrastructure as Code” pentru a facilita deployment-uri automate
- Arhitectura evaluata / propusa **TREBUIE** sa faciliteze scrierea testelor de orice tip

- Arhitectura evaluata / propusa **TREBUIE** sa asigure timpi reduși de compilare/ testare (ex. orice serviciu trebuie sa fie compilat si testat in maxim 5 minute)
- Arhitectura evaluata / propusa **TREBUIE** sa asigure ușurința in configurare, in special in contextul instanțelor multiple ale aceluiași serviciu
- Arhitectura evaluata / propusa **TREBUIE** sa permită utilizarea de hardware generic, fara cerințe specifice (brand/ model/ arhitectura CPU, etc). Este însă subînțeles faptul ca fiecare componenta a arhitecturii poate avea cerințe/ recomandări specifice ce tin de cantitatea de memorie, numarul de procesoare/ core-uri, capacitate si tip de stocare (ex. SSD)

Încheiere

Orice abatere / neconformitate fata de criteriile de mai sus **TREBUIE** analizata si justificata din perspectiva riscurilor aferente.

PROPUNERE ARHITECTURA PENTRU INFORMATIZARE INTEGRATA

Introducere

In urma analizei cerințelor diverselor proiecte componente ale viziunii de informatizare integrata a Consiliului Județean Cluj, acest document conține o serie de propuneri/ recomandări si observații pentru a facilita luarea de decizii, realizarea caietelor de sarcini si a planurilor de achiziție.

Obiective si perspective

Obiectivul principal al arhitecturii sistemelor de informatizare, parte a „Strategiei de Informatizare a Județului Cluj", este sa faciliteze implementarea si dezvoltarea incrementala a diverselor servicii si produse software ce compun viziunea de informatizare, oferind o fundație solida pentru diversificarea si îmbunătățirea serviciilor oferite, prin intermediul software.

Aceeași importanta o au însă si celelalte obiective:

- Asigurarea calității funcționale si non-funcționale (securitate, performanta)
- Facilitarea implementării folosind bunele practici la nivel de industrie
- Facilitarea dezvoltării colaborative, incrementale si evolutive
- Optimizarea costurilor de dezvoltare, extindere si întreținere
- Facilitarea inovației in serviciile oferite prin intermediul software

- Integrarea facila a sistemelor existente precum si a altor soluții existente (comerciale sau open source) care ar satisface nevoile funcționale si non-funcționale ale Consiliului Județean Cluj, însă nu au la baza premisele arhitecturale detaliate in acest document

Conform criteriilor pentru argumentarea arhitecturii, definite separat, arhitectura propusa trebuie sa aibă o serie de calități, fiecare cu importanta deosebita in atingerea unuia sau mai multor obiective dintre cele enumerate mai sus.

Pe lângă aceste calități, trebuie sa ținem cont si de premisele implementării si evoluției acestei arhitecturi:

- Sistemele informatice nu vor fi furnizate/ dezvoltate de către același furnizor, prin urmare arhitectura trebuie sa permită integrarea cu ușurința a soluțiilor oferite de furnizori diferiți.

- Anumite sisteme informatice nu pot fi tranzitionate către un nou model arhitectural, fără costuri majore din partea furnizorului, costuri ce ar afecta direct sau indirect o potențială achiziție, prin urmare arhitectura trebuie sa permită si integrarea cu sisteme de arhitectura diferita, însoțite de un plan de tranziție către un stadiu recomandat/ ideal, într-un timp rezonabil, agreat.

- Dezvoltarea/ implementarea/ integrarea diverselor soluții se va desfășura pe parcursul unei perioade mai lungi, timp in care arhitectura trebuie sa permită integrarea/ extinderea funcționalității curente fără a afecta semnificativ (ideal ar fi deloc) funcționalitatea existenta.

- Din aceleași considerente de durata, data fiind evoluția continua si rapida a tehnologiei, arhitectura trebuie sa permită adoptarea tehnologiilor noi in dezvoltarea noilor componente, pentru a putea utiliza beneficiile acestora, fără modificarea componentelor sau a funcționalității existente.

- Sintagma ***„lucrul potrivit la postul potrivit”*** (***„the right tool for the right job”***) ar permite aplicarea unor soluții eficiente si eficace pentru fiecare problema funcțională adresata. De aceea, arhitectura trebuie sa aibă capabilitatea de a absorbi o multitudine de „lucruri potrivite”, bineînțeles filtrate de implicațiile operaționale, costurile de întreținere si gradul de maturitate al fiecărei soluții alese. Ne putem referi aici atât la limbajele de programare, cat si la sistemele de tip server (ex. Baza de Date, Broker de Mesaje, Servere Web, etc), sau la alte aspecte tehnice alte componentelor software.

Date fiind obiectivele, precum si premisele/ perspectivele mai sus menționate, recomandările arhitecturale descrise in cele ce urmează, tind sa ofere o soluție inclusiva, standardizata, cu un grad ridicat de libertate, manifestat (in mod ideal) in contextul acestei standardizări.

Tipuri de aplicații

Considerând multitudinea de soluții necesare pentru implementarea strategiei de digitalizare a Consiliului Județean Cluj, am identificat următoarele stiluri / tipuri de aplicații:

Aplicații / module web de tip Website Clasic - orice aplicație sau secțiune a unei aplicații destinata consumului public de informație. In această tipologie putem include aplicațiile de tip „Content Management Systems".

Caracteristicile principale ale acestor aplicații sunt dictate de conținutul furnizat:

- Conținutul are un grad ridicat de dinamism
- Conținutul este destinat unui spectru mai larg de utilizatori, cu interese si așteptări multiple / variate, in funcție de aria/ secțiunea accesata
- Conținutul trebuie sa fie optimizat pentru a fi indexat de către motoarele de căutare (Google, etc)
- Aplicațiile trebuie sa furnizeze capabilități de navigare/ căutare si urmărire a conținutului
- Aplicațiile pot furniza capabilități de interacționare cu conținutul
- Aplicațiile pot furniza capabilități de personalizare a conținutului

Aplicații / module web de tip Website Static - orice aplicație sau secțiune a unei aplicații destinata consumului public de informație, cu un ciclu de viată limitat/ restricționat. Ne putem referi aici la pagini speciale de tip „Landing Page", create in special pentru anumite campanii de informare / evenimente sau concursuri. Aceste aplicații sunt de obicei limitate in cantitatea de conținut furnizat si de interacțiunile cu utilizatorul pe care le oferă.

De asemenea, aceste aplicații sunt de cele mai mult ori orientate înspre o convergenta a comportamentului utilizatorului către o confirmare sau o acțiune foarte bine definita (ex. abonarea la o lista de știri, crearea unui cont, obținerea unor informații promoționale, trimiterea anumitor documente/ informații către beneficiar, etc).

Nu in ultimul rând aceste aplicații (denumite si micro-site-uri) au o experiența cu utilizatorul limitata/ simplificata, pun un mare accent pe urmărirea comportamentului (in scopul optimizării eficientei) si pe performanta in afișare si interacțiuni.

De multe ori, funcționalitatea descrisa mai sus poate fi furnizata prin intermediul sistemelor de tip Content Management Systems, prin urmare cele doua concepte nu se exclud reciproc.

Aplicații / module web de tip "Account Area" / Zona Protejata - orice aplicație sau secțiune a unei aplicații destinata interacțiunii unui subset/categorie a publicului larg (categorie ce deține un mecanism de acces restricționat: ex. nume utilizator + parola, cont utilizator, etc) cu funcționalități avansate de introducere, extragere si consum al datelor (ex. lucrul cu formulare, încărcare de documente, parcurgerea unor fluxuri funcționale, operații diverse asupra datelor personale, interacțiune cu diverse sisteme de comunicare)

Caracteristicile principale ale acestor aplicații sunt dictate de funcționalitatea si nivelul de interacțiune intre utilizatori si sistemul informatic:

- Interfețele sunt optimizate pentru interacțiune facila (usability/ user experience)
- Aplicațiile vor minimiza numărul de interacțiuni care vor solicita o reîncărcare completa a paginii, si vor înlocui dinamic conținutul afișat in funcție de stadiul interacțiunii, prin utilizare modelelor „Single Page Application".
- Aplicațiile utilizează de obicei servicii de tip „back-end" implementate de obicei prin API-uri REST
- Aplicațiile pot oferi capabilități avansate de personalizare a experienței de utilizare
- Aplicațiile oferă preponderent funcționalități pentru utilizatori care revin ocazional si interacționează cu sistemul/ aplicația in mod frecvent, cel puțin pentru o durata predefinita (ex. durata aprobării anumitor documente)

Aplicații / module web de tip "Back Office" / Zona Administrativa si Operațională - orice aplicație destinata personalului administrativ / autorizat, prin intermediul căreia se procesează informațiile si solicitările utilizatorilor sau se executa diverse procese interne specifice activității desfășurate.

Caracteristicile principale ale acestor aplicații sunt dictate de procesele interne si de interacțiunile / integrările existente. De asemenea, funcționalitatea diferă de la utilizator la utilizator in funcție de rol si/sau nivel de permisiuni acordate fiecărui utilizator in parte:

- Aplicațiile vor minimiza numărul de interacțiuni care vor solicita o reîncărcare completa a paginii, si vor înlocui dinamic conținutul afișat in funcție de stadiul interacțiunii, prin utilizare modelelor „Single Page Application".
- Aplicațiile vor oferi capabilități proactive – funcții care permit utilizatorului sa creeze înregistrări / conținut in funcție de responsabilități si rolul specific in cadrul aplicației.
- Aplicațiile vor oferi capabilități reactive – funcții care permit utilizatorului sa reacționeze la diverse evenimente din cadrul sistemului (conținut realizat de alți utilizatori, procese interne care au atins anumite etape, notificări si alerte ale sistemului, etc.)

- Aplicațiile vor oferi capabilități analitice – funcții care permit utilizatorului sa analizeze/ vizualizeze diverse informații furnizate/ prezentate de către sistemul informatic, lăsând la latitudinea utilizatorului sa decidă pașii următori ai interacțiunii.
- Aplicațiile vor oferi capabilități executive – funcții care permit utilizatorului sa efectueze diverse activități in cadrul sistemului, in funcție de responsabilități si drepturi, sa parcurgă etape ale anumitor procese, etc).
- In majoritatea cazurilor, aplicațiile „Back Office" vor combina o interfața cu utilizatorul (Front End) de tip „Single Page Application" cu o aplicație Back-End care sa furnizeze interfața de programare/ interacțiune de tip REST API (Application Programming Interface folosind protocoale REST/ HTTP)

Aplicații / module web de tip „Back-End" - orice aplicație destinata utilizării prin integrarea in alte tipuri de aplicații de tip Front-End prin intermediul interfețelor de programare (ex. REST API).

Caracteristicile principale ale acestor aplicații sunt dictate de cerințele specifice ale interfețelor cu utilizatorul (pot exista multiple interfețe – web/mobile/etc – asociate aceleiași aplicații de tip Back-End) si de funcționalitatea oferită:

- Nivelul de complexitate cel mai ridicat este concentrat in majoritatea situațiilor in aceste aplicații/ module
- Funcționalitatea implementata variază de la simple operații de validare/ persistare si furnizare a informațiilor la funcții avansate de raportare, analiza inteligenta a datelor, import, export, interacțiune si integrare cu medii / sisteme externe, etc.
- Funcționalitatea oferita este in majoritatea situațiilor furnizata de către module/ servicii distincte, fiecare cu responsabilitate clara, bine definita si izolată, agregate prin intermediul unui serviciu centralizat de interfațare cu mediul exterior, denumit Gateway sau API Gateway in cazul utilizării serviciilor ce expun interfețe de tip REST API

Aplicații / module de tip „Puncte de Extensie/ Interfatare/ Integrare" - orice aplicație destinata integrării unei aplicații existente/ terțe cu o alta aplicație prin mecanisme care permit interoperabilitatea fără schimbări in aplicațiile integrate. Un exemplu îl constituie orice aplicație care preia informații dintr-un anumit sistem denumit sursa si transformarea/ conversia conținutului plus transferul informațiilor într-un sistem denumit destinație.

Aceste aplicații/ module pot furniza interfețe de integrare a sistemelor externe atunci când acestea necesita interfețe/protocoale diferite fata de cele adoptate ca si standard (de exemplu, pentru a evita utilizarea de formate proprietare in arhitectura integrata, se pot implementa puncte de extensie pentru integrări cu

sisteme terțe, care transforma informația standardizata, expusa prin interfețe de tip REST / JSON in servicii SOAP/XML care încapsulează componente/bucăți de informație in formate proprietare – de exemplu un format binar, comprimat, convertit in Base64 – este evident ca acest format (deși prezentat la extrem) nu trebuie sa facă parte din serviciile si interfețele expuse de către arhitectura integrata, însă pentru a putea interfața cu modulul extern, se poate implementa un modul special pentru integrare)

Aplicații specializate pentru terminale mobile - orice aplicație destinata utilizării de către utilizatorii interni (din cadrul Consiliului Județean Cluj) sau externi (cetățeni ai Județului Cluj – si nu numai) pentru a facilita o interacțiune simplificata si optimizata pentru terminalele mobile.

Fiind un înlocuitor al aplicațiilor de tip Web / Front-End, aplicațiile mobile vor utiliza capabilitățile oferite de aplicațiile de tip Back-End / API. In anumite situații însă, funcționalitatea oferita pe terminale mobile va necesita anumite capabilități suplimentare specifice, ce vor fi implementate tot in aplicațiile de tip Back-end. Acest lucru este necesar pentru îmbunătățirea si optimizarea experienței utilizatorilor, a performantei si a eficientei comunicării.

Principii fundamentale ale arhitecturii integrate

Pentru a identifica stilurile arhitecturale potrivite pentru informatizarea Consiliului Județean Cluj, pornim de la câteva principii din categoria „SOLID PRINCIPLES”, principii ce guvernează recomandările din acest document:

- **Open Closed**: conform acestui principiu, orice componenta a sistemului este „Deschisa la extindere” si „Închisă la modificare”. Cu alte cuvinte, orice subsistem trebuie sa permită extinderea sistemului dar sa nu necesite schimbări ale sistemului in sine pentru a facilita acest lucru.

- **Single Responsibility**: conform acestui principiu, orice componenta a sistemului are o responsabilitate unica, foarte bine definita, izolată si încapsulată.

Stiluri / modele de arhitecturi pentru aplicațiile de tip „back-end”

Pentru implementarea aplicaților / modulelor / funcționalităților de tip back-end, exista o multitudine de arhitecturi disponibile. Cu toate acestea, industria software a evoluat in ultimii ani, odata cu evoluția soluțiilor de tip Cloud (dar nu numai) înspre modele de arhitecturi care permit o dezvoltare accelerata, evolutiva si cu costuri mai reduse la nivelul întregului ciclu de viată al produselor software fata de soluțiile/ arhitecturile tradiționale.

In acest sens, se diferențiază o serie de modele de arhitecturi care pot conviețui in cadrul unor sisteme/ platforme integrate.

Microservices & Service Oriented Architecture (SOA)

Atat arhitecturile bazate pe Microservicii cat si arhitecturile de tip SOA trateaza cateva probleme majore a produselor software si a ciclului de viată ale acestora:

- Atat functional cat si non-functional, orice produs/solutie software necesita modificari pe intreaga durata de exploatare. Fie ca aceste modificari sunt solicitate de oportunitati de a genera profit („Return Of Investment”) sau alte tipuri de beneficii, sau nevoi de a adresa potentiale riscuri aparute sau schimbari externe (cadrul legislativ, etc), costurile de intretinere a produselor/solutiilor software cresc exponential odata cu complexitatea lor.

  Arhitecturile bazate pe Microservicii sau cele de tip SOA adreseaza aceasta problema prin impartirea complexitatii produselor/solutiilor software in servicii independente, fiecare cu un scop bine definit. In acest fel, orice schimbare este usor de localizat in serviciul responsabil iar

modificarile/ remedierile sunt mult mai usor de realizat. Un beneficiu colateral este si reducerea riscurilor la nivelul intregului sistem (indiferent ca vorbim de riscuri asociate calitatii, performantei, etc. prin crearea unui context in care diverse componente ale sistemului sunt semnificativ mai usor de testat/validat iar impactul schimbarilor este mai usor de evaluat.

- Solutiile software tranditionale, de tip Monolit, atrag după sine de cele mai multe ori așa numitul efect de „tehnology / vendor lock-in", sau client captiv al unui furnizor sau al unei tehnologii. Acest lucru este datorat in primul rând riscurilor aferente unei schimbari de o asemenea anvergura, fie ca este vorba de inlocuirea unui sistem sau de actualizarea / înlocuirea unei tehnologii. Si nu in ultimul rând, acest efect nedorit se propaga asupra tuturor tehnologiilor dependente.

  Arhitecturile bazate pe Microservicii sau cele de tip SOA adreseaza aceasta problema prin limitarea impactului fiecarui serviciu si a fiecarei tehnologii dependente. Fie ca este vorba de o baza de date utilizata sau de o librarie utilizata in cadrul unui serviciu, acestea pot fi eliminate / inlocuite / actualizate mult mai usor, impactul acestor operatii fiind minimal. In felul acesta, orice operatie poate fi planificata si efectuata gradual, cu costuri si riscuri minime.

- Microserviciile aduc beneficii multiple si la nivelul productivitatii si calitatii: mediul de dezvoltare pentru un astfel de serviciu este mult mai usor de replicat fata de mediile complexe impuse de solutiile de tip Monolit. Astfel, un programator / dezvoltator poate deveni productiv intr-un timp mult mai scurt.

Asynchronous: Message Driven Architectures & Event Driven Services

Chiar si in arhitecturile bazate pe Microservicii, oricât de independente sunt aceste servicii intre ele, exista de cele mai multe ori nevoia de comunicare. In aceste situații, trebuie sa ținem cont de faptul ca aceste servicii sunt distribuite, iar stadiul in care se afla ele nu este cunoscut fiecărui serviciu in parte. Prin urmare, comunicarea intre aceste servicii trebuie sa fie una asincrona. Acest lucru înseamnă ca aceasta comunicare, pe cat posibil, nu trebuie sa implice așteptarea unui răspuns.

Arhitectural, acest tip de comunicare implica următoarele modele:

- Servicii asincrone active - declanșate de mesaje / comenzi . Aceste mesaje sunt tratate tranzacțional in sensul in care mesajul va declanșa o singura execuție.

- Servicii asincrone reactive - declanșate de evenimente. Aceste evenimente pot fi tratate de către un număr nedefinit de servici.

Ambele modele aduc o multitudine de avantaje atunci cand sunt implementate corespunzator. Un exemplu il reprezinta extinderea sistemului cu noi capabilitati prin introducerea unui serviciu care trateaza un anumit eveniment si reactioneaza intr-o mod diferit fata de celelalte servicii. In acest fel, noul serviciu adauga functionalitate la sistem fara sa modifice starea anterioara a sistemului.

Event Sourcing

Spre deosebire de arhitectura bazata pe evenimente, Event Sourcing nu este un tip arhitectural atașat comunicării sau modului de execuție ci modului in care entitățile sunt privite (si eventual stocate). In acest sens, fiecare entitate este definita ca o succesiune de evenimente care au dus de la starea incipienta (creare) la starea actuala. In felul acesta, evenimentele pot fi reinterpretate in orice moment pentru a putea genera noi perspective asupra entităților de care acestea aparțin.

CQRS – Command Query Responsibility Segregation

Conform principiilor care guvernează recomandările arhitecturale cuprinse in acest document, este foarte importanta separarea responsabilităților. In special in cazul modelelor complexe, este deseori util sa utilizam modele diferite de prezentare sau reprezentare a entităților fata de modelul in care aceste entități sunt stocate. Aceasta separare, deși aduce o oarecare complexitate conceptuala, permite flexibilitate si extensibilitate dincolo de capabilitățile unui singur model.

O explicație este data de faptul ca niciun model nu poate corespunde tuturor cerințelor, actuale si viitoare. Prin urmare, un sistem/serviciu care stochează o categorie de entitati intr-un anumit fel va permite interogări ale acelor entități intr-un mod care poate fi optimizat pentru moment, însă pe măsură ce nevoile/ cerințele se schimba, modelul devine incompatibil cu ansamblul tuturor cerintelor, ceea ce duce la probleme de performanta, ineficienta sau timpi ridicați de ajustare a sistemelor.

Modelul arhitectural CQRS permite nu doar segregarea reprezentarilor entitatilor in model de stocare si model de interogare, ci si definirea unui număr nelimitat de reprezentări/ modele de interogare pentru aceleași categorii de entități. Iar prin utilizarea unei soluții optime locale pentru fiecare model de interogare, prin prisma soluțiilor de stocare, se obține o soluție optima globala in termeni de eficienta si performanta.

Prin combinarea modelelor CQRS cu Event Sourcing, se obține un model arhitectural flexibil si rezistent atât la schimbări cat si la erori: oricând un model de interogare nu corespunde cerințelor, entitățile reprezentate se pot remodela prin parcurgerea tuturor evenimentelor si regenerarea modelului de interogare.

Stiluri / modele de arhitecturi pentru aplicațiile de tip „front-end”

Aplicatii web statice

Data fiind simplitatea lor, aplicatiile web statice sunt rareori denumite aplicatii, ele fiind mai des idenficate ca si pagini statice. Cu toate acestea, acest tip de aplicatii sunt extrem de practice si eficiente atunci cand informatia servita nu necesita schimbari semnificative pe parcursul existentei. Aceste pagini / aplicatii statice sunt in majoritatea situatiulor formate din cateva pagini HTML care au associate fisiere de stil (CSS) si scripturi JavaScript, eventual librarii terte utilizate (ex. JQuery, etc) pentru implementarea anumitor functii specific interfetei cu utilizatorul. Aplicabilitatea lor este de obicei asociata cu pagini cu o durata de viata redusa (ex. o campanie, un eveniment, etc).

Cu toate acestea, pentru standardizare si usurinta in operare, este de dorit ca si aceste aplicatii, ce pot fi deservite de servere web simple, sunt impachetate in containere, folosind tehnologiile agreate.

Aplicatii web dinamice

Marea majoritate a informatiei servita publicului larg prin intermediul unui site va utiliza aplicatii web dinamice. Acest lucru inseamna ca acesta informatie este stocata intr-o baza de date si apoi afisata utilizatorilor prin transformarea acestui continut in cod HTML (insotit, asemenea aplicatiilor web statice de stiluri CSS si scripturi JavaScript). Aceasta transformare este realizata de aplicatii server (tehnologii de scripting sau server-side), iar procesul in sine poarta numele de “Server Side Rendering”. Cu alte cuvinte, este responsabilitatea aplicatiei server sa furnizeze continul in modul in care un “Browser Web” o poate interpreta si afisa.

Aplicatii web de tip „Single Page Application”

Diferenta majora intre aplicatiile SPA si aplicatiile web dinamice este ca in cazul SPA responsabilitea transformarii continutului intr-o forma pe care un browser web sa o poata afisa revine chiar browserului, de fapt a unei aplicatii ce ruleaza in browser (o serie de scripturi JavaScript). Aceasta aplicatie ce ruleaza in browser va utiliza continut preluat de la server prin intermediul unui API, de obicei in format JSON. Formatul JSON nu este un format considerat prietenos cu utilizatorul obisnuit, prin urmare aplicatiile SPA vor transforma aceste continut in HTML pe care browserul il va prezenta utilizatorului.

Vedem astfel ca si aplicatiile SPA au in spate o componenta dinamica, o componenta pe care de cele mai multe ori o numim API si care este deservita de aplicatii back-end.

O alta caracteristica a aplicatiilor de tip SPA este faptul ca odata incarcata aplicatia in browser, acesta nu va face o reincarcare completa decat atunci cand utilizatorul paraseste zona functionala delimitata de aceasta aplicatie. In schimb, pe masura ce utilizatorul navigheaza, aplicatia doar afiseaza in locurile potrivite elemente vizuale specifice functionalitatii si sistemului de navigare. Acest mecanism oferta utilizatorului o experienta imbunatatita fata de alte tipuri de aplicatii si de asemenea un tip de raspuns redus.

Aplicatii web mixte si componente web

In situatia integrarii mai multor aplicatii web, de tipuri diferite (statice, dinamice sau de tip Single Page Application), pentru realizarea unei experiente unitare este importanta asigurarea unei consecvente in ceea ce priveste anumite zone „comune” sau partajate de catre diverse aplicatii.

Un exemplu il reprezinta o zona a interfetei care permite utilizatorului sa efectueze operatii de control al profilului personal, sau o sectiune de navigare integrata intr-un portal cu multiple sectiuni.

Pentru uniformizarea experientei utilizatorului, se recomanda realizarea unor componente speciale, reutilizabile, independente de platforma utilizata sau tipul de arhitectura front-end utilizat, care sa incapsuleze diverse functionalitati comune.

Aceste componente poarta numele de Componente Web sau „Web Components”, o tehnologie care ofera un grad de independenta ridicat.

Stiluri / modele de arhitecturi pentru integrarea aplicațiilor „legacy” si servicii terțe

In cazul integrarii aplicatiilor si serviciilor terte intr-o infrastructura si platforma comuna se doreste alinierea tuturor serviciilor pentru a putea oferi utilizatorilor o experienta unitara si consecventa.

Cu exceptia aplicatiilor care permit implementarea de puncte de extensie/ personalizari care pot inlocui anumite functionalitati (ex. autentificarea) cu servicii adaptate platformei integrate, integrarea se poate realiza prin 2 modele arhitecturale:

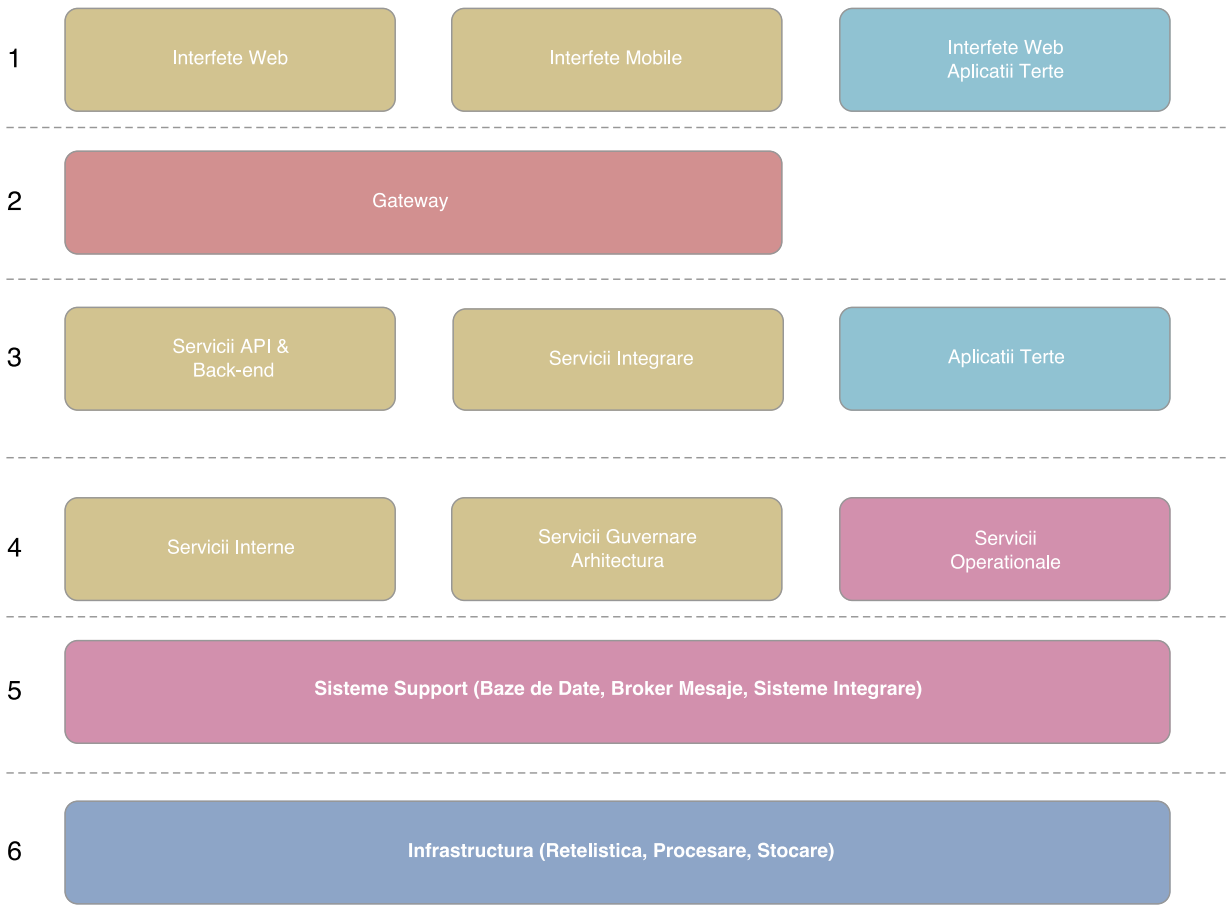
- Utilizarea unei platforme de integrare a serviciilor – Enterprise Service Bus (ESB), o solutie care oferta mecanisme de preluare, translatare si transmitere a informatiilor intre doua sau mai multe platforme diferite, ce utilizeaza tehnologii diferite (fie la nivel de DB, fie ca si arhitectura/ componente, limbaje, etc).

- Utilizarea unor tehnologii ce pot fi utilizate in cadrul serviciilor dezvoltate, pentru a expune capabilitati de integrare cu platforme terte (pentru a expune cu usurinta canale noi de preluare sau distribuire a informatiei). De exemplu, un serviciu care proceseaza solicitari primite individual prin interfata REST/ HTTP, ar putea fi adaptat sa preia volume mai mari de date din medii FTP, monitorizand locatii remote si incarcat / procesand fisierele noi care corespund anumitor criterii.

Aceasta abordare poate fi aplicata atat aplicatiilor legacy, depasite tehnologic cat si oricarei aplicatii terte a carei modificare, in scopul integrarii, ar putea atrage costuri semnificative relativ la beneficiile aduse de aceasta integrare.

Propunere arhitectura pentru informatizare integrata

Arhitectura pentru informatizarea integrata a Consiliului Judetean Cluj aduce sub aceasi umbrela o serie de aplicatii si servicii cu functionalitate/ utilitate diferita. Obtinem astfel o perspectiva la nivel de sistem asupra arhitecturii, prezentata in imaginea de mai jos:



Nivelul 1 reprezinta zona de interfata cu utilizatorul, compusa din aplicatii de tip „front-end” sau aplicatii mobile.

Cu exceptia solutiilor terte a caror modificare nu este posibila, pentru a uniformiza mecanismele de autorizare si filtrare a conexiunilor si solicitarilor de tip HTTP, accesul la aceste servicii/ servere web este intermediat de o componenta speciala, denumita **Gateway,** aflata pe **Nivelul 2** al arhitecturii sistemului.

Dupa cum a fost deja demonstrat, toate aplicatiile prezente in Nivelul 1 au in spate o serie de servicii (API) sau servere (web) care servesc atat continutul static cat si dinamic, aflate pe **Nivelul 3**. Pe acelasi nivel logic se afla si diverse servicii care sunt expuse pentru integrarea altor servicii/ aplicatii terte.

Urmatorul nivel, **Nivelul 4,** este dedicat serviciilor operationale sau interne, care sunt utilizate in alte servicii insa nu sunt la randul lor expuse ca si servicii/ API publice. In majoritatea cazurilor aceste servicii sunt asincrone, iar comunicarea este cu precadere una bazata pe mesaje si evenimente si nu pe interfete REST.

Nivelul 5 este dedicat sistemelor suport, Servere de Baze de Date, Servere Mesaje, etc.

Infrastructura fizica reprezinta **Nivelul 6**, nivel care incorporeaza toate sistemele hardware necesare sustinerii arhitecturii integrate.

Pentru a putea distribui si opera eficient multitudinea de servicii si dependente, este ideal ca toate acestea sa fie împachetate intr-un mod unitar. O abordare tehnologica care permite acest lucru este cunoscuta sub numele de containerizare. Aceasta permite abstractizarea aplicațiilor/ serviciilor de diverse atribute ale sistemelor de operare sau ale componentelor hardware. Acesta abordare este aplicabila cu precădere nivelelor 2-4 din perspectiva prezentata mai sus. In funcție de cerințele de performanta sau securitate, serviciile prezente pe nivelul 5 (bazele de date, etc) pot rula pe sisteme dedicate.

Odată cu creșterea numărului de servicii si diversificarea nevoilor de scalare (numărul de instanțe per serviciu, conexiunile si dependentele serviciilor), atingem un nivel ridicat de complexitate in operarea acestor servicii. De aceea este necesar sa apelam la tehnologii dedicate de orchestrare care permit distribuirea multiplelor servicii pe multiple noduri de procesare, asigura distribuirea incarcarii serviciilor (load balancing) catre toate instantele create si permit de asemenea ajustarea numarului de instante ale diverselor servicii in functie de nivelul de incarcare la nivel de sistem.

Tehnologii pentru punerea in practica a arhitecturii

Conform premiselor inițiale, arhitectura propusa pentru informatizarea Consiliului Judetean Cluj este una cu un grad ridicat de libertate, care permite adoptarea tehnologiilor potrivite pentru diversele capabilități dezvoltate.

Cu toate acestea, pentru diverse nivele ale arhitecturii, prezentam in cele ce urmează o serie de recomandări.

Limbaje de programare

In funcție de specificul fiecărui serviciu sau aplicații, exista o multitudine de limbaje care oferă suportul necesar. De aceea se recomanda alegerea limbajelor de programare in funcție de nevoile specifice si criteriile de performanta si calitate stabilite, si nu impunerea unui singur limbaj in întreaga implementare.

Tabelul de mai jos prezinta o serie de recomandări de limbaje de programare, asociate cu utilizările recomandate / uzuale. Orice alt limbaj nereprezentat in acest tabel este însă acceptat in condițiile in care acesta face parte din aceeași clasa de limbaje cu una dintre clasele prezentate sau, in condițiile in care prin utilizarea limbajul respectiv se aduc beneficii semnificative, consemnate si evidențiate in momentul implementării / livrării.

In momentul adoptării / implementării de servicii sau aplicații este important sa evaluam limbajele de programare (precum si alte tehnologii) din perspective multiple:

- maturitatea limbajului

- disponibilitatea si răspândirea limbajului in rândul comunității tehnice locale (datorita nevoilor ulterioare de dezvoltare / întreținere)

- nivelul de adecvare al limbajului la problema adresata

Limbaje	Utilizare recomandata
C#, Java, Haskel, Erlang, F#, Scala, Groovy	Servicii back-end si aplicatii de complexitate ridicata
NodeJS cu JavaScript sau TypeScript , Python	Site-uri web dinamice si servicii back-end de complexitate medie
PHP, Ruby	Site-uri web dinamice si servicii back-end de complexitate redusa
Python, R	Servicii back-end specializate in procesarea datelor (ex. Machine Learning)

Baze de date

Tipul bazelor de date utilizate este dictat de nevoile si de cerințele funcționale si non-funcționale ale fiecărui serviciu in parte. Ca si in cazul limbajelor de programare, alegerea unui tip de baza de date trebuie sa tina cont de câțiva factori:

- Nivelul de maturitate
- Costul asociat (de licențiere, exploatare, suport, etc.)
- Capabilități funcționale si non-funcționale
- Capabilități operaționale si administrative (back-up, migrare, export, etc.)

- Eficienta in atingerea cerintelor functionale si non-functionale
- Nivelul de scalabilitate (replicare, etc.)
- Nivelul de suport (atât comercial cat si la nivelul comunității locale)

Deși exista o multitudine de tipuri de baze de date, grupate in „Baze de date relaționale / SQL" si „Baze de date NoSQL", in tabelul de mai jos vom prezenta câteva recomandări.

Tip baze de date	Recomandari
Relaționale / SQL	MySQL, PostgreSQL, SQL Server, Oracle
Document Storage	MongoDB
Key/Value Storage	Redis
Column Storage	Cassandra
Graph	Neo4J
Search	Elastic Search

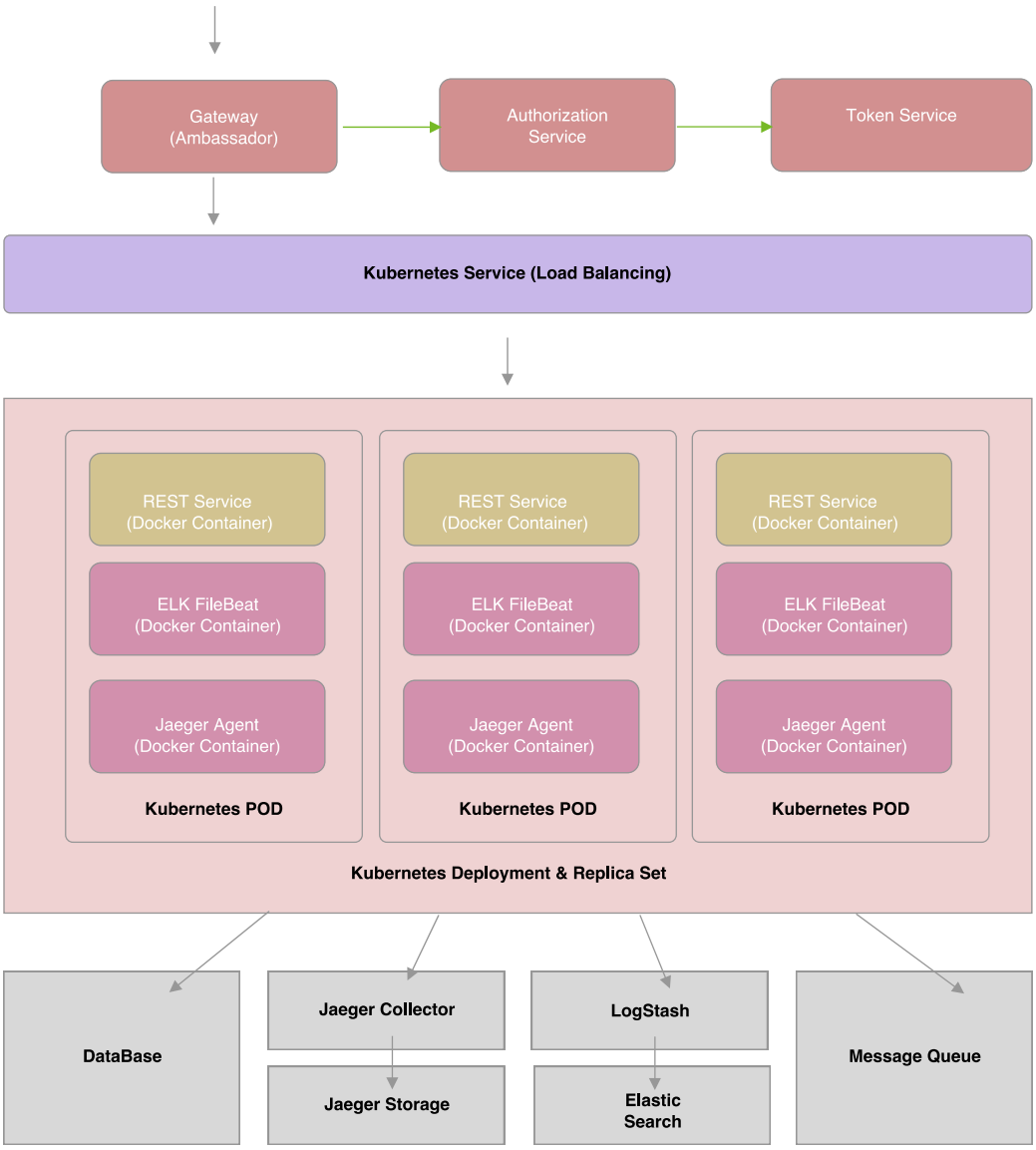
Alte tehnologii

Pornind de la multitudinea de tehnologii si soluții adiacente necesare, prezentam mai jos o lista de recomandări tehnologice pentru fiecare nevoie/ scenariu de utilizare.

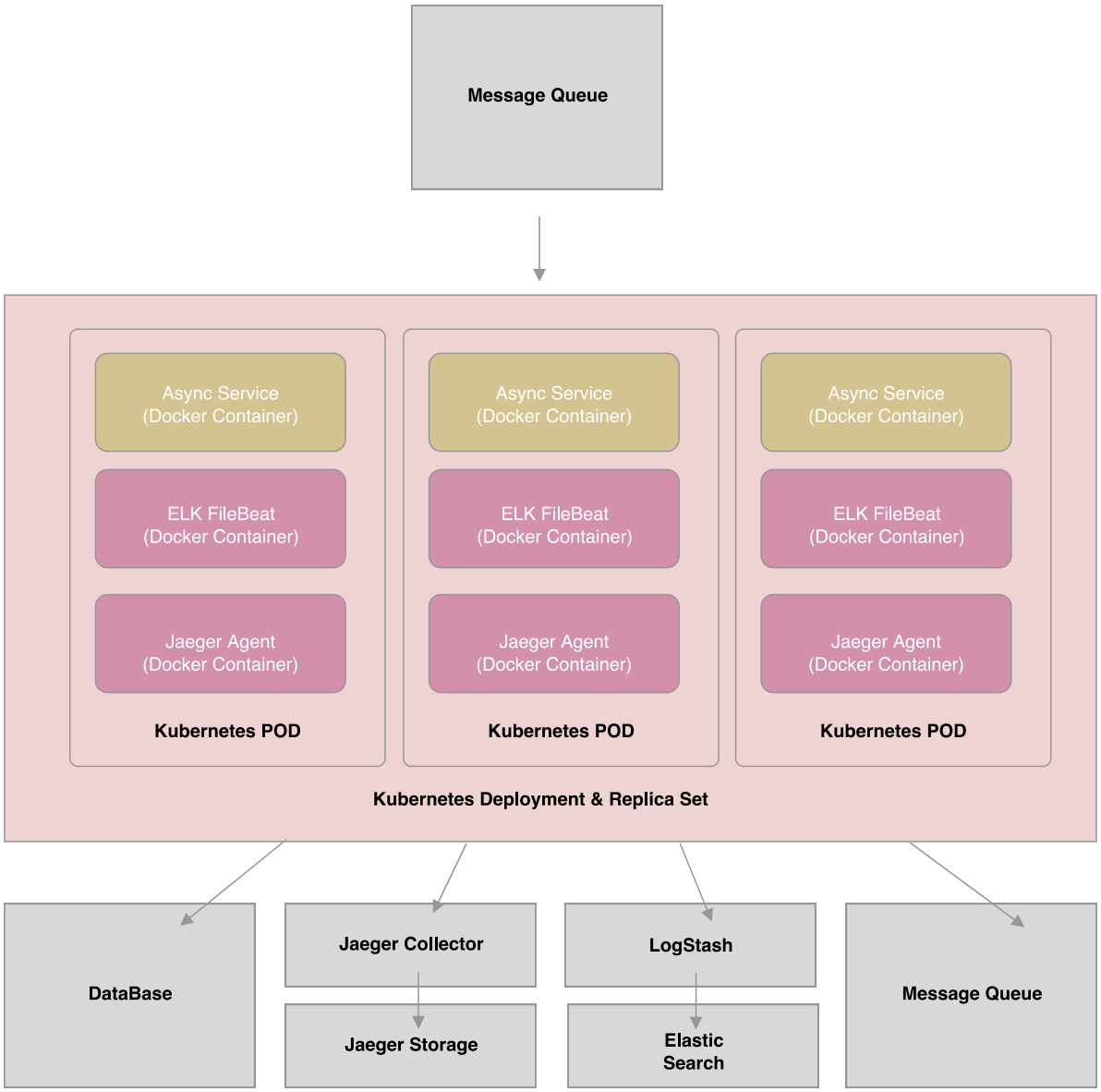
Utilizare	Recomandari
Servere Web	Apache HTTP, NGinx, Kestrel
Soluții/Servere Integrare	Mule ESB, Apache Camel, Spring Integration
Message Broker	RabbitMQ, ActiveMQ, Apache Kafka
Tracking & Correlation	Open Tracing folosind ZipKin sau Jaeger
Agregarea log-urilor aplicatiilor	Elastic Search + LogStash + Kibana
Impachetare servicii / Containerizare	Docker
Orchestrare servicii	Kubernetes
API Gateway	Ambassador

Aplicand tehnologiile recomandate pentru punerea in practica, obtinem o noua perspectiva asupra serviciilor oferite de platforma integrata, o perspectiva centrata pe cele 2 tipuri de servicii majore:

- Servicii publice, expuse prin REST



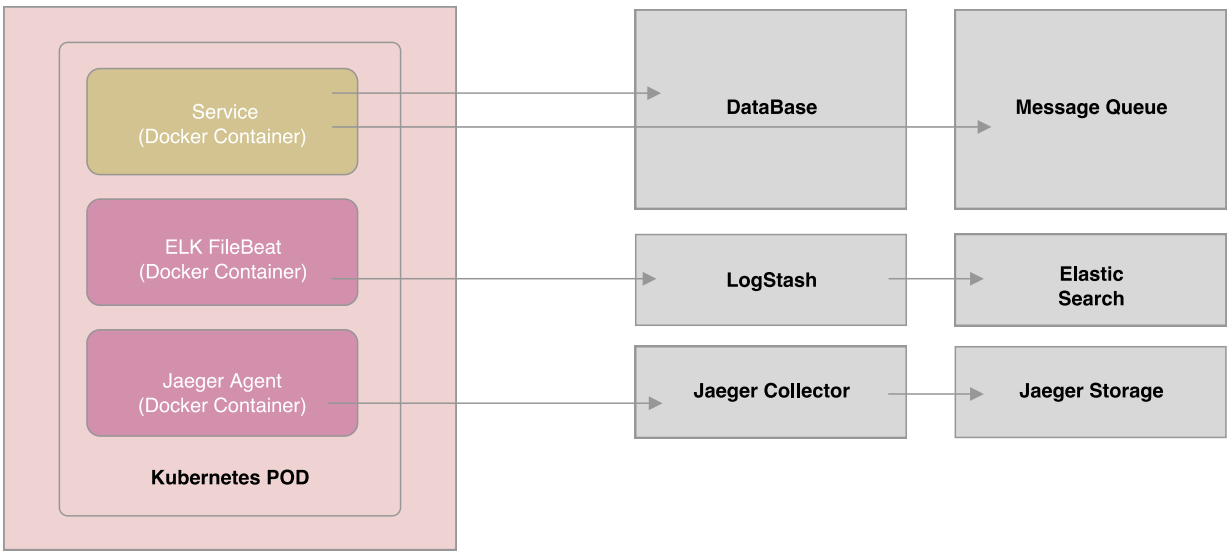
- Servicii interne/ asincrone care comunica prin intermediul cozilor de mesaje



Indiferent de tipul serviciului, observam cateva caracteristici comune:

- serviciile sunt incapsulate in containere Docker
- fiecare serviciu este „insotit” de o serie de servicii operationale, responsabile cu preluarea si centralizarea informatiilor de monitorizare / logging
- fiecare serviciu poate avea mai multe instante, asigurand astfel scalarea independenta de celelalte componente ale sistemului
- fiecare serviciu foloseste propriul mecanism de stocare, propria baza de date. Acest lucru asigura intependenta serviciilor si creste nivelul de scalabilitate
- fiecare serviciu publica o serie de mesaje si evenimente in cozile de mesaje, in functie de specificul serviciului

Putem avea si o perspectiva „marita” (zoom-in) asupra serviciilor, focusandu-ne pe interactiunea fiecarui serviciu cu sistemele dependente:



Esentiala in aceasta perspectiva este independenta serviciilor (ele depinzand in general de componente suport, si nu de alte servicii), calitate/capabilitate cu rol vital in atingerea obiectivelor arhitecturale propuse