UNIVERSITATEA „BABEŞ-BOLYAI”
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE DIPLOMĂ

STUDIUL EFICIENȚEI FOLOSIRII UNEI ARHITECTURI BAZATE PE MICROSERVICII ÎNTR-UN SISTEM DE PLATĂ A UTILITĂȚILOR

Coordonatori științifici:
lect. dr. **Mircea Ioan Gabriel**

Absolvent:
Petruțiu Paul-Gabriel

Cluj-Napoca
2019

Cuprins

Capitolul 1

Introducere

Capitolul 2

Concepte de bază

2.1 Arhitectura unei aplicații soft

Deoarece tot mai multe companii din diferite domenii au nevoie de o aplicație software personală pentru a-și imbunatații calitatea produselor, pentru prezentarea produselor sau pentru ușurarea unor activități din cadrul companiei, cererea de a crea cod este tot mai mare. Pentru ca aplicațiile sa fie usor de dezvoltat, pentru ca o imbunătățire sa poată fi implementată fără prea mult efort, este important ca aplicația sa aibă la baza o structură bine definită. Acest lucru face ca o aplicație să fie ușor de intreținut în decursul timpului.

Având in vedere standardul IEEE Std 1471, arhitectura unei aplicații soft este definită ca „Organizarea fundamentală a unui sistem încorporat în componentele sale, relațiile dintre ele și mediul, precum și principiile care conduc proiectarea și evoluția sa.”((Hilliard, 2000))

Așadar, putem considera un sistem ca fiind o aplicație sau un set de aplicații care impreună rezolvă diferite probleme.

2.2 Șabloane de proiectare

Un șablon de proiectare este reprezentat ca fiind o solutie cunoscuta pentru o problema recurenta.

„Șabloanele de proiectare pot fi văzute ca un mijloc de a reuși reutilizarea pe scară largă prin captarea unei practici de design de dezvoltare a software-ului de succes într-un anumit context” ((Alencar, Cowan, & Lucena, 1996)). Deci, un șablon de proiectare reprezintă doar în mod abstract o soluție pentru o problemă. Din acest motiv, șabloanele de proiectare pot fi aplicate în oricare limbaj de programare, in funcție de contextul problemei.

Motivul pentru care șabloanele de proiectare sunt relevante indiferent de limbajul de programare folosit, este acela că șabloanele de proiectare sunt doar concepte despre cum ar trebui să fie impementat codul si nu cod propriu zis.

2.3 Serviciu Software

Un serviciu este o componentă a unei aplicații soft care furnizează una sau mai multe funcționalități altor componente din sistem. Aceste componente pot să fie aplicații web, mobile sau chiar alte servicii.

Spre exemplu, putem presupune ca avem un site în care utilizatorii pot să achite fucturile de curent, gaz, etc.. În momentul în care utilizatorul efectuează o plată, browser-ul va apela un serviciu care va procesa, în spate, aceasta tranzactie(verificarea daca suma introdusă este validă, dacă suma este disponibilă, etc.)

Sistemele ce folosesc mai multe servicii, similar cu exemplul expus mai sus, sunt considereate ca fiind sisteme care au la bază o arhitectură orientată pe servicii.

Capitolul 3

Aplicații Monolit

3.1 Ce este o aplicație monolit?

O aplicație monolit este o aplicație a cărei cod este scris in cadrul unei singure unitați structurale. Componentele din care este alcătuită aplicația sunt gândite in așa fel incât sa funcționeze împreună și să se folosească de acelasi spatiu de memorie si de aceleași resurse.

Aplicațiile monolit sunt printre cele mai raspândite din lume. Acest fapt se datorează felului în care oamenii abordeaza problemele. O soluție de tip monolit este prima solutie pe care un programator o va avea, deoarece este un mod natural de a gandi. În plus, pentru multe din aplicații, o solutie monolit va fi soluția perfectă, având în vedere că multe companii mici spre medii doresc sa aiba o aplicație care să automatizeze anumite procese, procese care nu au o complexitate extrem de mare, iar numărul utilizatorilor nu este enorm.

Spre exemplu, să spunem că o firmă are un sediu destul de mare care are 5 săli de conferință, având in vedere numărul mare de întâlniri din interiorul firmei, aceștia doresc o aplicație în care să poată rezerva o sala de conferință pentru o anumită perioadă intr-o anumită zi. O astfel de aplicatie se poate realiza usor si rapid, sub forma unei aplicații web de tip server client(3.1).

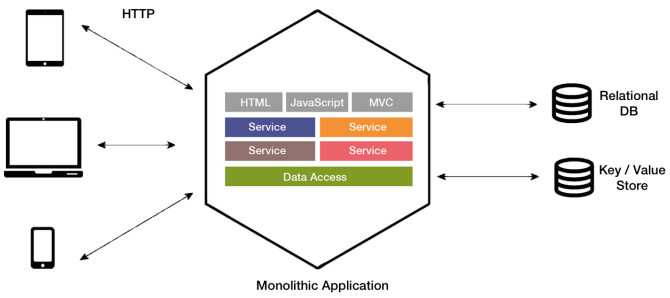

Figura 3.1: Arhitectură Aplicație Monolit(imagine : http://bits.citrusbyte.com)

3.2 Avantaje și dezavantaje

O aplicație monolit are mai multe avantaje in contextul unui business mic:

- Faza de dezvoltare durează mai puțin timp, IDE-urile moderne reușind să genereze mult cod automat, ceea ce implică scădearea costurilor de producție.
- Aplicația este usor de testat, automatizând procesul de testare a codului folosind o combinație de unit tests și integration test.
- Procesul de lansare în producție a unei aplicatii nu este complicat, deoarece este nevoie de rularea unei singure aplicații per server.

Pe de altă parte, in momentul în care aplicația va creste ca și volum al codului, un număr mai mare de programatori vor trebui implicați în procesul de dezvoltare al aplicației, acest lucru nu este un lucru rău, doar ca aceștia vor intampina anumite impedimente:

- Cuplajul dintre componentele aplicației creste, iar acest fapt va îngreuna procesul de dezvoltare a unor noi functionalități, acest lucru va afecta timpul si costul necesar dezvoltării noii funcționalități.
- Alegerea tehnologiilor folosite pentru dezvoltarea aplicației este permanentă.
- Integrarea/Intrarea unui programator in proiect va fi mai dificilă. Volumul aplicației fiind mare, va fi necesar un timp mai mare pentru înțelegerea tuturor funcționalităților si a deciziilor luate pe proiect in trecut.

Acestea sunt doar câteva dintre avantajele si dezavantajele unei aplicații monolit, dar sunt suficiente incât putem sublinia o idee generală. Aplicațiile bazate pe o arhitectura monolit sunt mai ușor de implementat în primă fază cu un cost mai redus, dar în momentul în care aplicația ajunge la un nivel mai avansat, este nevoie de o reconstruire parțiala sau totală a aplicației sau cel puțin este nevoie de o refactorizare care sa diminueze dezavantajele create de stilul arhitectural folosit. Majoritatea aplicațiilor încep ca și aplicații tip monolit, iar mai târziu se pot transaforma in aplicații cu arhitecturi bazate pe microservicii de exemplu.((Thönes, 2015))

Capitolul 4

Arhitectura bazată pe servicii

4.1 Fundamentele arhitecturii bazate pe servicii

4.2 Sabloane de proiectare pentru arhitecturi bazate pe servicii

Capitolul 5

Sabloane de proiectare bazate pe microservicii

5.1 Sabloane de implementare a microservicilor

5.1.1 Agregator

5.1.2 API Gateway

5.2 Sabloane arhitecturale de proiectare

5.2.1 Service bus

5.2.2 Procesare asincrona

5.2.3 Agregarea proceselor

5.2.4 Alte Sabloane

Capitolul 6

Studiu de Caz

6.1 Introducere

Netflix este o companie care a început prin a vinde sau închiria filme pe DVD-uri. Ulterior furnizând acces online la filme si seriale. Netflix fiind un gigant in industria televiziunii online. Aplicația netflix este o aplicație la scară mondială, care în momentul in care firma a horărat schimbarea arhitecturii avea un trafic de 8 milioane de utilizatori, ajungând la finalul anul 2018 la 139 de milioane de utilizatori.

Dupa cum am discutat până acum, o arhitectură bazată pe microservicii nu are chiar o definiție propriu zisă, dar dupa cum susține Martin Fowler, microserviciile sunt implementarea corectă a arhitecturii bazate pe servicii.

În acest capitol vom cuprinde trecerea de la o arhitectură monolit la o arhitectura bazată pe servicii, motivele pentru care s-a facut aceasta tranziție, pașii prin care s-a facut aceasta trecere, avantajele cât si dezavantajele acestei treceri, cât si despre rezultatul final.

6.2 Tranziția către microservicii (Netflix)

În acest studiu de caz o să ne bazăm pe informațiile oferite de doua dintre personajele importante care au luat parte la tranziția catre microservicii:

- Ruslan Meshemberg
- Adrian Cockcroft
- Josh Evans

6.2.1 Arhitectura aplicației de azi

În următoarea diagrama, este reprezentată, arhitectura bazată pe microservicii a aplicației Netflix.

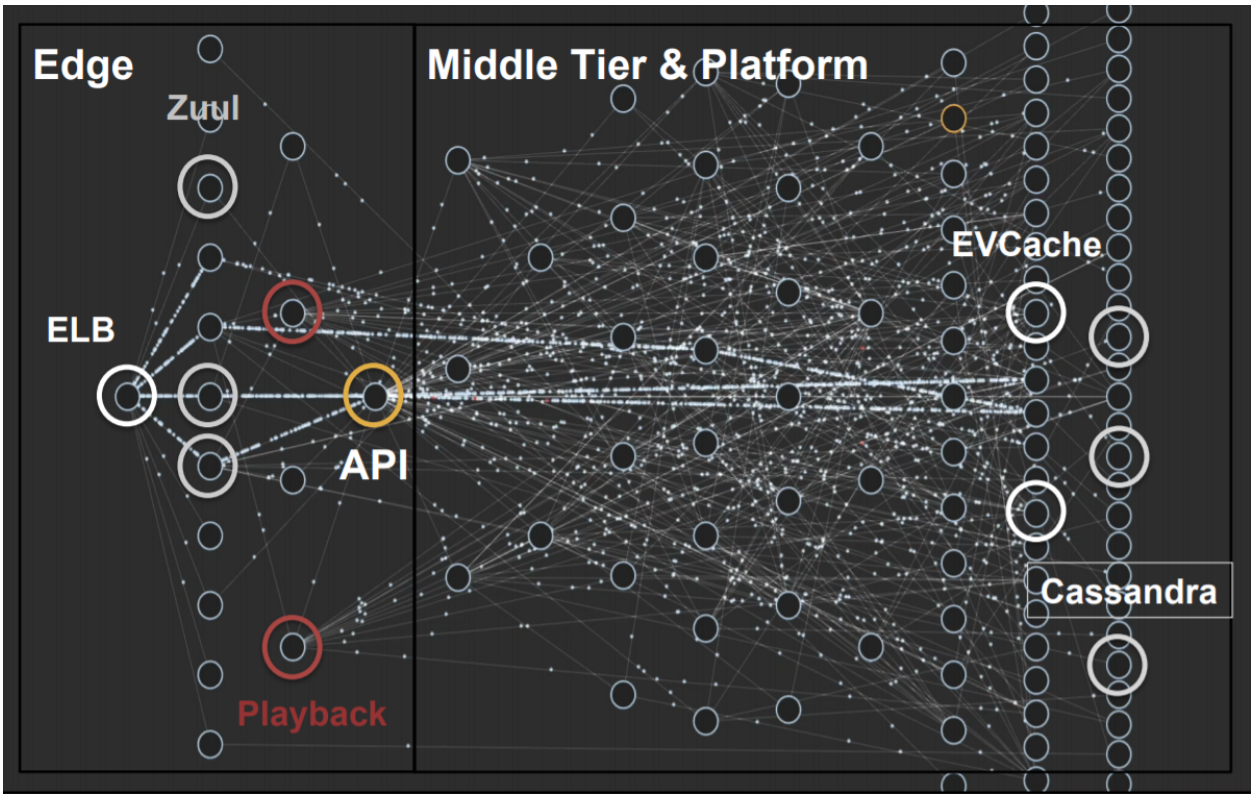

Figura 6.1: Arhitectura Aplicației Netflix()

Făcând o scurtă analiză asupra diagramei, putem să observam că prezentarea este una stratificată, nodul notat „ELB” rebrezintă un balansier de capacitate elastic (Elastic Load Balancer), care se o ocupă cu distrubuirea uniforma a cererilor catre microservicii. Stratul in care avem nodul notat „Zuul”, este un strat de tip proxy care ajută la oprirea atacurilor cibernetice. Iar zona dintre acest strat si „Middle Tier” este reprezentată de componente accesibile clientilor. În partea a 2-a a diagramei, este un strat destul de stratificat de microservicii care conțin mai multa logică si serversc componentelor din prima parte a diagramei. Ultimul strat, cel care contine si nodul notat cu „Cassandra” pe diagramă, reprezintă stratul de acces la bazele de date, datele fiind salvate separat in baza de date NoSQL. Iar intre ultimele 2 straturi explicate, avem un strat care se ocupa de caching.

O diagramă mai reprezentativă pentru arhitectura bazată pe microservicii a Netflix, care contine mai mult de 500 de microservicii este numita si diagrama de arhitectura „Death Star”(6.2).

Așadar, având în vedere scala la care știm deja ca funcționează aplicația Netflix, putem deja considera că arhitectura bazată pe microservicii isi servește bine scopul. În continuare vom discuta efectiv despre motivele, pașii, beneficiile și costurile acestei schimbari de arhitectură.

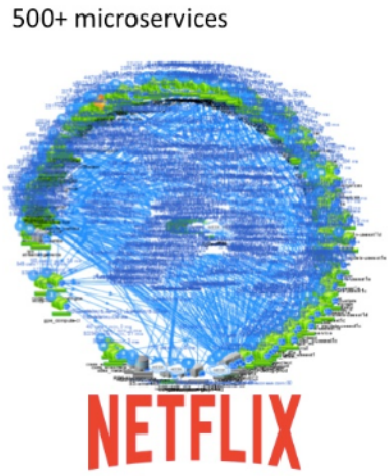

Figura 6.2: Arhitectura Aplicației Netflix()

6.2.2 Motive

Unul dintre primele motive care au înpins compania netflix isi mute aplicația catre cloud și odata cu asta, inceperea tranziției catre noua arhitectură, a fost o criza întampinata in 2008 când baza de date a fost coruptă, iar acest lucru a dus la intreruperea activitații timp de 3 zile.

6.2.3 Avantaje și dezavantaje, costuri și sacrificii

Capitolul 7

Aplicație Practica

7.1 Cerința

7.2 Specificații

7.3 Arhitectura aplicatiei

7.4 Implementare

7.5 Sablone bazate pe microservicii in practica

Capitolul 8

Concluzii

Bibliografie

Alencar, P. S., Cowan, D. D., & Lucena, C. J. P. d. (1996). A formal approach to architectural design patterns. In *International symposium of formal methods europe* (pp. 576–594).

Hilliard, R. (2000). Ieee-std-1471-2000 recommended practice for architectural description of software-intensive systems. *IEEE, http://standards. ieee. org*, *12*(16-20), 2000.

Thönes, J. (2015). Microservices. *IEEE software*, *32*(1), 116–116.

Cockcroft, Adrian. 2015. Talking microservices with the man who made Netflix's cloud famous. [interviu cu] Derrick Harris. 2015.

Cockroft, Adrian. 2014. Migrating to Microservices by Adrian Cockroft. YouTube. [Interactiv] 2014. https://www.youtube.com/watch?v=1wiMLkXz26M.

Erl, Thomas. 2009. SOA Design Patterns. Crawfordville Indiana : Prentice Hall, 2009.

Fowler, Martin. 2014. Microservices, 2014,. MartinFowler.com. [Interactiv] 2014. https://martinfowler.com/articles/microservices.html.

OASIS Foundation. 2006. Reference Model for Service Oriented Architecture. [Interactiv] 2006. https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html.