UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA TEHNOLOGIA INFORMAȚIEI

Lucrare de licență

DEVOPS ÎN MICROSERVICII

Absolvent
Bahrim Dragoș

Coordonator științific
Conferențiar Doctor Kevorchian Cristian

București, iulie 2023

Rezumat

Microserviciile sunt o arhitectură de sisteme distribuite ce a luat amploare în urma structurării organizațiilor către piață. Aceasta a fost facilitată de schimbări în modul în care lansăm produsele prin dezvoltarea platformelor cloud și a ce a dus la îmbunătățiri la timpul de livrare dar și a consistenței. Obiectivul este prezentarea acestor concepte și implementărea acestora prezentând modul de gândire ce influențează ciclul produsului. În acest scop, mă documentez legat de domeniu și încerc să îmi fac o idee de ansamblu asupra aspectelor ce trebuie luate în construirea unui sistem iar apoi o să aplic aceste informații. Aceasta lucrare ar trebui să servească ca un prim contact cu microserviciile dar și a practicilor DevOps.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce vitae eros sit amet sem ornare varius. Duis eget felis eget risus posuere luctus. Integer odio metus, eleifend at nunc vitae, rutrum fermentum leo. Quisque rutrum vitae risus nec porta. Nunc eu orci euismod, ornare risus at, accumsan augue. Ut tincidunt pharetra convallis. Maecenas ut pretium ex. Morbi tellus dui, viverra quis augue at, tincidunt hendrerit orci. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam quis sollicitudin nunc. Sed sollicitudin purus dapibus mi fringilla, nec tincidunt nunc eleifend. Nam ut molestie erat. Integer eros dolor, viverra quis massa at, auctor.

Cuprins

Capitolul 1

Introducere

Domeniul informaticii, calculatoarelor și al tehnologiei informației este unul care va fi mereu în continuă evoluție. Dorința companiilor de a-și servi clienții cât mai rapid a dus la un avans tehnologic în care mereu apare ceva nou ce are ca scop îmbunătățirea proceselor actuale. Această arie și-a început dezvoltarea aproape acum o sută de ani iar avansul poate fi apreciat numai gândidu-ne la modul în care scriam cod și livram produse acum, acum zece ani și acum două zeci de ani. Se observă o diferență și la accesibilitatea unităților de calcul, actual majoritatea persoanelor au acces la un dispozitiv ce se poate conecta la Internet, ceea ce le permite să devină consumatori la diferite servicii.

Piața pentru unități de calcul a evoluat. În 1943, Thomas Watson, director la IBM menționează că „I think there is a world market for maybe five computers" (Cred că există o piață globală pentru poate cinci computere). În 1999, Michael Barr afirmă în „Programming Embedded Systems in C and C++" „One of the more surprising developments of the last few decades has been the ascendance of computers to a position of prevalence in human affairs. Today there are more computers in our homes and offices than there are people who live and work in them. Yet many of these computers are not recognized as such by their users." (Una dintre cele mai surprinzătoare dezvoltări a ultimelor decenii a fost ascensiunea computerelor într-o poziție predominantă în afacerile oamenilor. Astăzi sunt mai multe computere în casele noastre și în birouri decât persoane care trăiesc și lucrează în ele. Însă multe dintre acestea nu sunt recunoscute de către utilizatorii lor). Cele două fraze evidențiază modul cum s-a schimbat prezența computerelor în viața noastră. La început acestea erau foarte costisitoare, greu de administrat și de operat însă acestea au devenit din ce în ce mai mici iar recent aproape orice are nevoie de semiconductori datorită integrării ce le fac „smart".

Aceste afirmații se aplică și pentru produsele software, ce au ca menire să ofere asistență altor aplicații sau să fie consumate direct. Indiferent de locul în care mă duc, probabil persoana cu care aș interactiona folosește un computer, fie că îmi cumpăr ceva de la un magazin făcând o plată cu cardul sau că mi se livrează un colet iar curierul marchează pe AWB că a fost livrat. Christopher Little afirmă că „Every company is a technology com-

pany, regardless of what business they think they are in. A bank is just an IT company with a banking license." (Orice companie este o companie ce se axează pe tehnologie, indiferent de mediul de afaceri în care se află. O bancă este doar o companie IT cu licență de a funcționa ca o bancă.)

Desigur nu toate companiile sunt la fel și nu toate concurează cu toate companiile, însă chiar dacă nu ești o companie ce activează în tehnologie, probabil folosești componente tehnologice pentru a-ți îmbunătății randamentul sau să oferi clienților un avantaj față de competiție, care probabil gândește în același mod.

Însă pentru fiecare companie obiectivul este identic, să livreze clienților produse de calitate și cât mai rapid. Pentru aceasta trebuie să ne asigurăm că avem inginerii necesari pentru implementarea cererilor noi, însă momentul în care aceasta este terminată este doar începutul procesului de integrare și ulterior lansare, și uneori acesta este cel care reprezintă un blocaj din mai multe puncte de vedere ce limiteaza numarul de câte ori putem aduce o îmbunătățire produsului nostru.

În încercarea de a livra mai rapid sau pentru a ușura modul în care manevrăm un influx sau o lipsă de trafic, s-a încercat folosirea unei arhitecturi paralele asupra aplicațiilor creând arhitecturi orientate pe servicii, iar în momentul în care aceste servicii sunt concentrare pe un număr redus de funcționalități și pot fi lansate independent și să își păstreze funcționalitate putem vorba de o arhitectură bazata pe microservicii.

Însă schimbând arhitectura sau practiciile din modul de desfășurare al livrarii produsului nu este suficient pentru a asigura performanța, întrucât ambele dintre ele aduc dezavantaje ce trebuie tratate iar uneori acestea sunt mai mari decât avantajele pe care schimbarea le-ar aduce, însă implementarea corectă în locurile ce ar beneficia de o astfel de abordare poate să aducă performanțe ce nu ar fi posibile cu procesele vechi.

Afinitatea mea pentru microservicii provine de la lipsa de cunoștiințelor pentru a putea alege dintre mai multe limbaje de programare, framework-uri sau platforme. De exemplu, ce avantaje mi-ar face să aleg ca pentru server-ul meu să folosesc o aplicație în Node.JS sau una in Go? Chiar dacă putem să cunoaștem niște lucruri informative la început, cei mai buni indicatori sunt monitorizările proprii în producție. De asemenea pot fi cazuri în care îmbunătățirile aduse unei platforme pe parcurs o fac mai bună față de ce era inițial (dezvoltarea TypeScript poate să influențeze alegerea inițială a unei platforme SpringBoot ce este type-safe). Astfel, pe o arhitectură bazată pe microservicii putem să folosim limbajul potrivit pentru serviciul căruia îi aduce cele mai multe beneficii. Acest lucru se extinde și pentru baze de date. Poate în timpul evoluției serviciului apar tehnologii noi, însă având un singur lucru central ar face migrarea mult mai grea, de asemenea uneori ar trebui să facem compromisuri pe care alegerea unui alt tip de baze de date ar face să o dispară.

Dezvoltarea tematiciilor DevOps pornește de la ușurința cu care ne putem crea o bază de date folosind containere, întrucât instalarea unei baze de date pe calculatorul

personal în timp ce lucram la proiectele de facultate a făcut să am experiențe în care la finalul fiecărui an îmi reinstalam sistemul de operare doar pentru că o dată ce instalam o bază de date crea suficiente servicii, foldere care dupa o dezinstalare încă rămâneau. Folosind Docker, pot crea o bază de date cu o singură linie de cod, care în același timp să fie preconfigurată și să o pot împărtășii cu colegii de proiect ca să nu pierdem timp în configurarea mediului de lucru.

Lucrarea mea va urma o structură în care prezint noțiuniile teoretice la început iar la final încerc să aplic aspectele prezentate în crearea unei aplicații. Inițial pornesc cu prezentarea microserviciilor continuând cu diferitele moduri în care acestea pot fi lansate, imediat după cu dezvoltarea noțiuniilor de DevOps ce au ca scop accelerarea dezvoltării, iar la final o prezentare generală a modului în care îmi construiesc aplicația.

Capitolul 2

Preliminarii

Cel puțin până la momentul la care am decis ce temă să aleg pentru lucrarea mea de licență niciun curs de bază nu a abordat în detaliu tema pe care vreau să o dezvolt, din acest motiv lucrarea mea de licență nu este doar aprofundarea unui concept și crearea unei aplicații, ci încercarea familiarizării cu obiectele de lucru în același timp. Desigur, unele aspecte s-ar putea desprinde și în urma practicii în industrie dar nu a fost suficient ca să pot să consider că ma cunoștințe de bază în acest domeniu și doar încerc să găsesc lucruri cât mai specifice.

De asemenea, microserviciile sunt o arhitectură destul de nouă ce a luat amploare datorită limitărilor din punctul de vedere al dezvoltării și scalării aplicațiilor monolitice, astfel în opinia mea, este destul greu să vizualizezi anumite lucruri fără experiență la prima mână cu limitările acestea. Întrucât serviciul pe care îl dezvolt nu ar necesita neapărat o arhitectură bazată pe microservicii, ar funcția complet normal și aș reduce foarte mult din complexitate dacă ar fi dezvoltată în mod normal.

La fel și cu practiciile DevOps, acestea se axează mai mult companiilor care au nevoie de timp de penetrare a pieții foarte rapid, însă eu doar încerc să mă familiarizez cu setarea și folosirea serviciilor pentru uzurile mele proprii. Un lucru important ar fi ca ce folosesc să nu mă limiteze, de exemplu infrastructura de care am nevoie va provenii din Azure doar pentru că primesc credite gratuite ca și student, însă în absența lor probabil aș încerca lucruri cu cost minimal sau chiar nici sa nu le fac, și probabil aș avea același randamanet.

Capitolul 3

Microservicii

3.1 Concepte generale

Microserviciile sunt o arhitectură de sisteme distribuite ce a devenit populară în ultimele decenii datorită necesității scalării. Ceea ce o separă de o arhitectură orientate pe servicii este faptul că microserviciile sunt mult mai specifice, acoperind o singură parte din afacere și faptul că pot fi lansate în execuție independent, astfel microserviciile ar trebui să depindă cât mai puțin de altele. Comunicarea se face direct către interfețele expuse de fiecare, iar informațiile reținute de fiecare (de exemplu baza de date de date sau diferite fișiere) pot să fie modificate doar prin metodele de comunicare oferite ci nu direct.

Un concept important în microservicii este ascunderea de informații, astfel din exterior fiecare ar trebui să fie tratat ca o cutie neagră ce face anumite lucruri și expune anumite date, nu contează detaliile de implementare și nici tehnologiile folosite în implementare, ci doar capabilitățile fiecăruia. Acest lucru permite să modificăm microserviciul în funcție de cerințe, chiar și refactorizarea metodelor inițiale cât timp se respectă capabilitățile pe care microserviciul ar trebui să le îndeplinească.

Cel mai important aspect al microserviciilor este capacitatea de a lucra independent de celelalte. Acestea trebuie să funcționeze independent de celelalte iar dacă aducem schimbări într-unul atunci nu suntem obligați să facem schimbări în alt microserviciu, în caz contrat ar apărea dificultăți în lansare. Obținerea independenței poate să aducă contribuții majore în timpul de deployment însă implementarea este mult mai complicată datorită necesității comunicării cu alte servicii.

Un alt aspect de bază este acoperirea unui singur segment din afacere prin intermediul unui microserviciu. Alături de caracteristica precedentă, dacă afacerea și-ar dezvolta necesitățile ar fi mult mai dificil să modificăm mai multe microservicii și ulterior să coordonăm lansarea lor.

Independența microserviciilor este importantă, în acest scop, ele ar trebui să fie singu-

rele care ar putea să își modifice starea. Întrucât starea se referă la elementele componente, de exemplu o bază de dată, acestea nu ar trebui să fie împărtășite și nici accesate de alte microservicii. De asemenea ar trebui evitate modificarea interfețelor declarate pentru a evita necesitatea modificării altor microservicii. „ Dimensiunea microserviciile ar trebui să fie redusă prin oferirea unui număr restrâns de funcționalități declarate pentru a a fi ușor de înțeles și întreținut. Însă important este cât de multe microservicii pot fi administrate întrucât un număr mai mare de microservicii, deși mici ca funcționalități, aduc dificultăți în comunicare, scalare și depanare.

Microserviciile sunt agnostice din punct de vedere al tehnologiilor care ar putea fi folosite. Din acest motiv, acestea oferă flexibilitate însă prețul plătit este crearea a mai multor puncte vulnerabile.

Orice prezentare al acestui tip de arhitectură nu ar fi completă fără prezentarea conceptului de „monolith" (monolit), întrucât este considerată o arhitectură veche reprezentând modul de funcționare din trecut. În cel mai simplist mod de funcționare, sistemul monolitic este reprezentat de un singur proces „gigantic" ce acoperă toată funcționalitatea sistemului, în general conectat la o singură bază de date ce și ea este reprezentată de un singur proces. Putem intui problemele cu această organizare, dacă conexiunea către procesul monolitic cade sau baza de date are probleme, întreg sistemul este afectat. Microserviciile incearcă să acopere aceste probleme prin independență, astfel dacă un microserviciu este căzut, celelalte nu sunt afectate. Procesul poate fi însă separat pe module dezvoltate separat însă care la final se cuplează formând un singur proces. Există conceptul de sistem monolitic distribuit în care acesta este la fel separat în module independente însă care nu ar funcționa dacă nu sunt toate active, acest tip de sistem oferă toate dezavantajele sistemelor distribuite dar și a monolitului ceea ce îl face destul de rar întâlnit.

Chiar dacă o arhitectură monolitică prezintă dezavantaje evidente ce sunt ușor de văzut, acestea nu sunt la fel de mari pentru companiile mici. Numărul redus de aplicații ce trebuie lansate și monitorizate, posibilitatea de reutilizare a codului mult mai ușoară, obținerea infrastructurii pentru proces și uneori chiar și scalarea, deși este limitată poate să facă o arhitectură monolitică mult mai atractivă. Nu ar trebui să asociem arhitectura monolitică ca un lucru antic, ci să o considerăm ca o opțiune. Însă pentru o organizație mică, acesta ar trebui să fie doar un punct de plecare, eventual după ce serviciul oferit devine mai popular s-ar putea să ajungem să cunoaștem limitările arhitecturii și să ne orientăm tot către microservicii, însă dacă nu avem succes, eliminăm multe complexități apărute în urma introducerii microserviciilor.

Principalele avantaje ale microserviciilor provin de la baza acestora, o arhitectură distribuită, însă cuplat cu conceptul de ascundere a informației și domain-driven design (design orientat domeniu) aduc multe alte avantaje asupra altor arhitecturi distribuite. În cadrul unui sistem format din microservicii putem folosi orice tip de tehnologie pentru

program și orice tip de bază de date, întrucât ascundem implementarea de exterior. Astfel putem alege tehnologii ce aduc avantaje în dezvoltarea microserviciului. Acest avantaj duce la capabilitatea de a folosi tehnologii noi fară a afecta tot sistemul, însă limitează capacitatea de a împărtășii cod (de exemplu prin librării interne, întrucât acestea ar trebui să fie rescrice pentru fiecare limbaj). Un sistem distribuit rezistă mult mai ușor la căderi, întrucât acestea pot fi tratate ca să se prevină un lanț, însă cu un număr mare de puncte slabe devine greu de aproximat cât de mult este afectat un sistem. Scalarea unui sistem monolitic este ușoară, doar replicăm procesul, însă la microservicii scalarea poate fi mult mai concentrată pe punctele care chiar au nevoie să fie scalate. Lansarea unei schimbări într-un microserviciu este mult mai ușoară întrucât nu necesită crearea de timp mort în aplicație, întrucât nu tot sistemul este înlocuit ci doar o mică parte din acesta, ce poate fi chiar și mai controlată prin diferite aplicații de orchestrare.

Adaptarea microserviciilor vine cu un cost, pe lângă dificultățile de implementare a design-ului apar și alte probleme, de exemplu dezvoltarea locală. Atunci când dezvoltăm un microserviciu acesta, posibil, este nevoit să comunice cu alte microservicii care la rândul lor comunică cu alte microservicii și putem continua, toate acestea microservicii trebuie să funcționeze în același timp pe laptop-ul dezvoltatorului, întrucât nu putem folosi microserviciile din producție, însă nu e posibil să rulăm foarte multe microservicii pe un singur computer depinzând de configurație. Adoptarea microserviciilor, în general nu poate avea succes decât dacă este combinată cu elemente de DevOps, ceea ce înseamnă ca dezvoltatorii și operatorii să învețe tehnologii noi, ceea ce este un împediment în momentul în care vrem să lansăm un produs rapid în piață și nu avem experiența necesară. Microserviciile necesită mult mai multă infrastructură pentru a fi rulată, mai multă tehnologie ce trebuie să fie folosită pentru ca dezvoltarea și livrarea să decurgă eficient, însă pe termen lung acesta poate să fie prevăzută dacă arhitectura este folosită cum trebuie prin livrare mai rapidă și mai eficientă deci profitul ar putea să crească. Monitorizarea sistemelui devine mai complicată, într-un proces monolitic toate fișierele de jurnalizare (logging) sunt în același loc. Într-un sistem distribuit, acestea trebuie să fie colectate, și ulterior ansamblate pentru a avea sens, acest lucru devine mai dificil când intră în discuție fenomenul de replicare al unei instanțe. Securizarea unui sistem distribuit este mai grea, traficul de date se face mult mai mult prin rețele sau prin diferite căi alternative ceea ce necesită un grad ridicat de atenție. Testarea unui sistem format din microservicii este mult mai dificilă, mai ales când trebuie testate mai multe microservicii în același timp, acesta duce la creșterea timpului necesar pentru a primi răspuns. Sistemele distribuite cresc timpul de răspuns întrucât acestea nu se mai face în cadrul unui singur proces, deci datele trebuie codificate și decodificate atunci când trec dintr-un mediu în altul. În cadrul unui sistem distribuit o altă problemă care apare este consistența datelor, dată de faptul că în timpul unei cereri, datele dintr-un microserviciu se pot schimba, astfel tranzacțiile devin dificile mai ales când înainte ne bazam pe atomicitatea și consistența oferită de

baza de date.

Astfel, microserviciile funcționează cel mai bine pentru organizațiile mari ce vor ca dezvoltatorii lor să livreze facilități noi foarte repede care să nu fie întârziate de compatibilitatea cu caracteristici deja existente. Prin natura variată a acestei arhitecturi se pot folosi tehnologii noi dar și folosirea platformelor cloud, astfel se oferă multă flexibilitate cu prețul creșterii complexității. Pentru companiile mici, al căror rată de succes este greu de calculat, overhead-ul produs de adaptarea unei arhitecturi în acest stil nu ar aduce suficient de multe avantaje și ar trebui să opteze pentru o arhitectură tradițională ce este stabilă și oferă colaborare mult mai ușoară în cadrul aceluiași proiect la scale mici.

3.2 Design

3.3 Tehnici de implementare

3.4 Build

3.5 Testare

3.6 Deployment

3.7 Monitorizare

3.8 Securizare

3.9 Evolutie

Capitolul 4

Orchestrare

4.1 Istoric deployment

4.2 Tipuri de deployment

4.2.1 Baremetal

4.2.2 Mașină virtuală

4.2.3 Container

4.2.4 Function as a Service

4.2.5 Platform as a Service

4.2.6 Container as a Service

4.3 Docker

4.4 Docker Swarm

4.5 Kubernetes

4.6 Red Hat OpenShift

4.7 HashiCorp Nomad

4.8 Infrastructure as a Service

Capitolul 5

Îmbunătățirea metodelor de dezvoltare software

5.1 Schimbari in livrarea produselor

5.2 DevOps și aspecte ale comunitații

5.3 Ce înseamnă să aplici DevOps

5.4 Unelte

Capitolul 6

Aplicație

Capitolul 7

Concluzii