Microservicii: șabloane, performanță, evoluție

Sergiu Breban[1]

[1] *Universitatea Babeș-Bolyai, Cluj-Napoca, Cluj*

February 5, 2018

Abstract

Arhitectura bazată pe microservicii este inspirată de tehnicile de calcul orientate pe servicii care au câștigat popularitate în ultima perioadă și este folosită pentru a construi sisteme complexe compuse din procese mici, independente și decuplate. Vom prezenta principalele caracteristici ale acestei tehnici, cu avantaje și dezavantaje, inclusiv impactul acestei abordări asupra performanței rețelei de comunicare, dar și câteva etape din dezvoltarea acestei arhitecturi.

1 Introducere

Abordarea arhitecturală bazată pe microservicii este relativ recentă în cadrul modelelor folosite în industria software. Această abordare folosește un set de servicii independente, de mărime cât mai mică, cu roluri bine determinate în cadrul domeniului, fiecare rulând în cadrul propriului process. Aceste servicii comunică în general prin mecanisme contruite în jurul protocolului HTTP.(Uckelmann, Harrison, and Michahelles, 2011) Astfel de servicii pot fi instalate și lansate independent. În ultimii ani, odată cu apariția cloud computing-ului, microserviciile au câștigat popularitate, acest șablon arhitectural fiind cel mai potrivit pentru a profita de avantajele oferite de cloud, printre care disponibilitatea și scalabilitatea. Acest lucru poate fi realizat prin folosirea de virtualizarea bazată pe containere, care oferă o alternativă mai bună la hypervizori(Soltesz et al., 2007), prin implementări de tipul Docker, care folosește containere bazate pe sistemul de operare Linux. Modelul opus celui bazat pe microservicii este modelul monolitic, care conține întreaga logică a sistemului într-o singură unitate instalabilă.(Richardson, 2014) Această abordare nu mai este potrivită după ce dimensiunea sistemului crește. Un sistem monolitic este mai greu de întreținut și modificat, oferă o scalabilitate limitată și limitează alegerea tehnologiilor pentru dezvoltatori.(Dragoni et al., 2017)

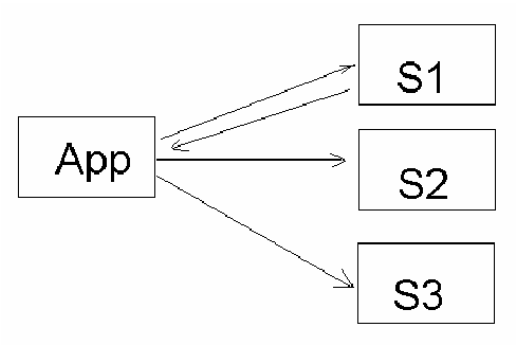
2 Șabloane

Dar arhitectura bazată pe micorservicii are și dezavantaje, cel mai mare fiind creat de complexitatea necesară pentru a crea un astfel de sistem distribuit și toate elementele necesare, de la comunicarea între servicii și procesarea transacțională. De asemenea, va crește și complexitatea instalării, fiind nevoie de gestiunea mai multor tipuri de servicii, dar și consumul de memorie în cadrul sistemelor folosite, datorită spațiului de adrese necesar fiecărui serviciu. Pentru a împărți un sistem monolitic în mai multe servicii independente este nevoie de diferite abordări, un tipar fiind delimitarea acestora pe baza cazurilor de utilizare. O altă abordare pentru partiționare este folosirea verbelor, a numelor sau a resurselor pentru delimitare, fiecare serviciu fiind responsabil pentru o anumită operație sau mai multe operații pe anumite entități.(Hassan, Zhao, and Yang, 2010) Se ține cont de pricipiul responsabilității unice (Single Responsible Principle (SRP)) pentru a garanta că fiecare serviciu are doar un set mic, bine definit, de responsabilități.

Microserviciile pot fi văzut ca niște componente software, unități software care unde înlocuibile independent, și au nevoie de un anumite tip ce proceduri chemate la distanță pentru comunicare. Pentru a garanta succesul unei astfel de arhitecturi, este nevoie de o interfață de comunicare atent construită și care poate fi ușor întreținută. Câteva primitive necesare pentru a asigura aceasta sunt: apeluri de tip cerere/răspuns cu date structura arbitrare, procesarea de eventimente asincrone în timp real în ambele direcții de comunicare, folosirea unui serviciu de serializare a mesajelor de tipul JSON, XML.

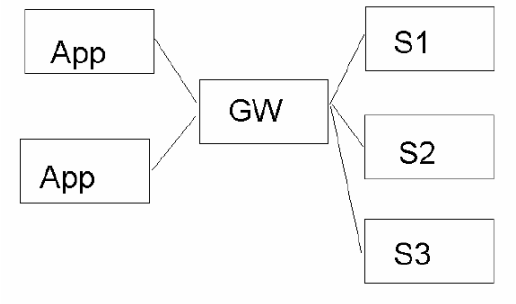
Componenta de rețea este foarte importantă în cadrul oricărei arhitecturi distribuite, deoarece oferă comunicare între diversele servicii. În cadrul arhitecuturii bazate pe microservicii, mai multe scenarii de comunicare sunt posibile. Un prim scenariu este cel în care aplicația client folosește fiecare serviciu în mod direct.

Figure 1: *Apeluri directe*



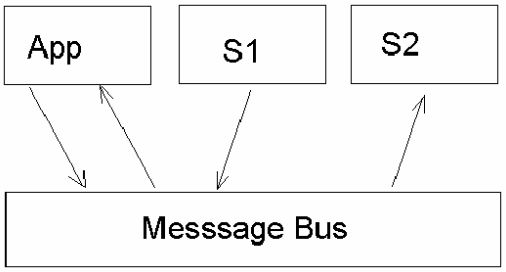
Acesta este cel mai flexibil mod. Pentru a scădea numărul de apeluri la distanță, se pot folosi diverse modele de caching sau middleware, ca și în figura de mai jos.

Figure 2: *Un gateway pentru microservicii*



Al treilea șablon folosit în gestionarea arhitectucrii cu microservicii este bazat pe un service bus, care permite aplicațiilor să trimită cereri și să citească răspunsurile mai târziu. Acest model facilitează adăugarea de noi componente fără modificarea celor existente.

Figure 3: *Service bus*

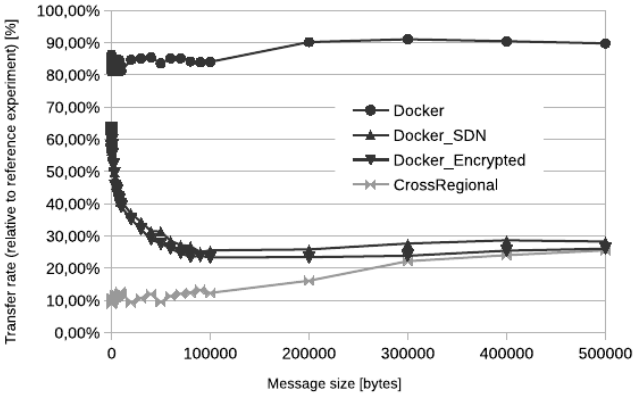


3 Perfomanțe

Virtualizarea bazată pe containere este văzută ca o soluție mai scalabilă și performantă decât hypervizorii în implementarea sistemelor bazate pe microservicii, dar aceasta din urmă este implementată cel mai des în sistemele de cloud computing. (Felter et al., 2015) oferă o analiză detaliată a consumului de memorie, CPU, stocare și resurse de rețea pentru a compara instalările tradiționale bazate pe mașini virtuale și utilizarea de containere Linux oferite de Docker, ajungând la concluzia că aceastea oferă aporximativ aceeași performanță în diferite procesări, fără a analiza însă impactul containerelor deasupra hypervizorilor.

Este nevoie de analiza performanței rețelei de comunicare a containerelor, a rețelelor virtuale definite în cadrul sistemelor software și a nivelelor de criptare asupra sistemelor distribuite bazate pe cloud care folosesc protocoale bazate pe HTTP și REST. Pentru a măsura acestea s-au făcut mai multe experimente, cel de referință fiind folosit pentru comparare. Sistemul de referință a fost un sistem bazat pe microservicii, instalat pe mai multe mașini virtuale și folosind REST și HTTP pentru comunicare. Performanțele acestui sistem au fost comparate pe rând cu adăugarea de containere, creare unei rețele virtuale definite software și aduăgarea unui nivel de criptare, pentru a măsura impactul fiecărei modificări. Se măsoară rata de transfer a mesajelor de mărime m (bytes), notate trans(m). Experimentul folosind Docker are rolul de a determina impactul containerelor asupra ratei de trasnfer în cadrul sistemelor bazate pe microservicii. S-a încercat de aseamenea și măsurarea performanței pentru sistemele a căror componente au fost instalate în regiuni diferite ale infrastructurii cloud oferite de către compania Amazon, AWS.

Folosind aceste sisteme de referință, s-au efectuat 12 ore de teste asupra lor, rezultând într-o transfer total de 316GB de date în mai mult de 6 milioane de cereri HTTP. Cea mai mare deviație a fost măsurată pentru mesaje de dimensiune foarte mică (10-20 de bytes). Deviația standard a crescut odată cu introducerea rețelei virtuale definită software. Astfel datele referitoare la deviație indică credibilitatea datelor experimentului. Viteza ratei de transfer este neglijabil afectată folosind containerele Docker, dar un nivel adițional de containere în cadrul unei mașini virtuale reduce performanța la 80% pentru mesaje mai mici de 100kB și la 90% pentru mesaje mai mari de 100kB. Un impact mai mare asupra vitezei de transfer o au rețelele virtuale definite în cadrul sistemului software, reducând performanța până la 60% pentru mesaje de dimensiune mică, sau chiar până la 25% pentru mai mari de 100kB. Adăugarea de criptare creează doar un impact minor asupra performanței.



Deci deși containerele sunt văzute ca având un impact mic asupra performanței, nu este întotdeauna cazul. Acestea oferă totuși o mai mare flexibilitate în cadrul sistemlor bazate pe microservicii, de aceea au devenit din ce în ce mai folosite în ultima perioadă.(Kratzke, 2017)

4 Evoluție

Arhitectura este cea care permite unui sistem să evolueze și oferă un nivel de serviciu de-a lungul ciclului de viață. În ingineria software, arhitectura oferă o punte între funcționalitățile sistemului și cerințele de calitate pe care sistemul trebuie să le atingă. De-a lungul ultimelor decenii, arhitectura software a fost studiată intens și multiple feluri de a compune un sistem software au fost create, până să fie dezvoltată arhitectura bazată pe microservicii.

Probleme asociate cu dezvoltarea de sisteme mari software a apărut pentru prima dată în jurul anilor 1960. Anii 1970 au însemnat un salt mare în interes pentru comunitatea de cercetători în domeniul proiectării acestor sisteme. În anii 1980, procesul de proiectare a fost integrat complet în cadrul procesului de dezvoltare a sistemelor software.

Prima referință a conceptului de arhitectură software a apărut pentru prima dat în anii 1980, dar prima definiție solidă a fost stabilită abia în anul 1992 de lucrarea (Perry and Wolf, 1992). Popularitatea crescândă a paradigmei de programare orientată pe obiect a adus contribuții la dezvoltarea domeniului arhitecturii software, în anii 1990 în mod special.

Atenția oferită separării intereselor în cadrul unui sistem software a dus recent la așa numita inginerie software bazată pe componente, care a oferit un control mai bun asupra proiectării, implementării și evoluției sistemelor software.

Sistemelor orientate pe servicii sunt o paradigmă folosită pentru dezvoltarea de sisteme distribuite care își are originile în programarea orientată pe obiect și pe componente. Aceasta a fost introdusă pentru a valorifica complexitatea sistemelor distribuite și pentru a integra diferite aplicații software. Beneficiile orientării către servicii sunt:

- dinamismul
- modularitatea și refolosirea
- dezvoltarea distribuită
- integrarea sistemelor heterogene și moștenite

Prima generație de arhitecturi de sisteme orientate pe servicii (SOA) a definit cerințe descurajante pentru servicii (descoperire și contracte între servicii), care au prevenit adoptarea acestui model. Microserviciile reprezintă a doua iterație a acestor concepte de arhitecturi orientate pe servicii. Scopul aceste iterații este de a îndepărta nivelele de complexitate inutile pentru a se pune accentul pe dezvoltarea de servicii simple care implementează eficient o funcționalitate.

Termenul de microservicii a fost introdus pentru prima dată în 2011 la un workshop de arhitectură software. Dar abordarea definită de acest termen era deja folosită, de exemplu în cadrul companiei Netflix, sub numele de *Fine Grained SOA*

Microserviciile sunt un trend nou în arhitectura software și oferă caracteristici distinctive facă de predecesorul SOA: mărimea unui serviciu este mică, contextul fiecăruia bine definit, iar funcționarea este independentă.

Caracteristicile cheie ale unui sistem bazat pe microservicii sunt:

- flexibilitatea
- modularitatea
- evoluția

Microserviciile au câștigat popularitate recent și încă există o lipsă de consens cu privire la ce anume reprezintă acestea de fapt. M. Fowler și J. Lewis oferă un punct de pornire prin definirea principalelor caracteristici, iar S. Newman oferă soluții și practici cu privire la această arhitecură.(Newman, 2015)

Dar acest sistem arhitectural este atât de nou încât putem spune că explorarea lui abia a început. Principala tărie a acestei abordări arhitecturale vine din distribuirea universală, chiar și a componentelor interne software, ca servicii autonome, care crează sisteme decuplate. În același timp, tot din acest aspect al distribuirii, rezultă și cea mai mare slăbiciune: programarea sistemelor distribuite este în esență mai grea decât programarea unor sisteme monolitice. Alte câteva exemple de posibile slăbiciuni ale acestei abordări vin din

modul în care se pot gestiona schimbările asupra unui serviciu fără a afecta celelalte servicii cu care acesta comunică sau cum pot fi prevenite atacurile care încearcă să exploateze comunicațiile prin rețea. De asemeanea, prevenirea erorilor de programare este mai grea în cadrul acestor sisteme.

5 Concluzii

Arhitectura bazată pe microservicii este un stil arhitectural care a câștigat popularitate în ultimii ani atât în mediul academic, cât și lumea industriei software. În particular, adoptarea microserviciilor este chestiune sensibilă pentru un număr de companii implicate în refactorizarea majoră a sistemelor backend. În ciuda faptului că este prezentat ca un concept revoluționar, este mai mult unul evoluționar. Fiind un concept recent, nu este dezvoltată încă o literatură cuprinzătoare în jurul lui. Rămâne de văzut dacă acest stil se va dovedi unul de succes, multe companii adoptând acest șablon pentru sistemele proprii, rezultatul acestei adopții putând fi observat de-a lungul anilor, prin performanța și valoarea pe care o aduc, și prin ușurința cu care pot fi modificare și evoluate.

Bibliography

Dragoni, Nicola et al. (2017). “Microservices: yesterday, today, and tomorrow”. In: *Present and Ulterior Software Engineering*. Springer, pp. 195–216.

Felter, Wes et al. (2015). “An updated performance comparison of virtual machines and linux containers”. In: *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. IEEE, pp. 171–172.

Hassan, Mahbub, Weiliang Zhao, and Jian Yang (2010). “Provisioning web services from resource constrained mobile devices”. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, pp. 490–497.

Kratzke, Nane (2017). “About microservices, containers and their underestimated impact on network performance”. In: *arXiv preprint arXiv:1710.04049*.

Newman, Sam (2015). *Building microservices: designing fine-grained systems*. " O’Reilly Media, Inc."

Perry, Dewayne E and Alexander L Wolf (1992). “Foundations for the study of software architecture”. In: *ACM SIGSOFT Software engineering notes* 17.4, pp. 40–52.

Richardson, Chris (2014). “Pattern: Microservices architecture”. In: *Microservices. io. http://microservices. io/patterns/microservices. html [last accessed on February 17, 2015]*.

Soltesz, Stephen et al. (2007). “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors”. In: *ACM SIGOPS Operating Systems Review*. Vol. 41. 3. ACM, pp. 275–287.

Uckelmann, Dieter, Mark Harrison, and Florian Michahelles (2011). “An architectural approach towards the future internet of things”. In: *Architecting the internet of things*. Springer, pp. 1–24.