**UNIVERSITATEA DIN BUCUREȘTI**

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**LUCRARE DE DISERTAȚIE**

|  |  |
| --- | --- |
| Coordonator științific  **Conf. Dr. Cristian  KEVORCHIAN** | Absolvent  **Alexandru Mihai GEOROCEANU** |

**București   
2015UNIVERSITATEA DIN BUCUREȘTI**

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**Platformă WEB pentru Jocuri în Timp Real**

|  |  |
| --- | --- |
| Coordonator științific  **Conf. Dr. Cristian KEVORCHIAN** | Absolvent  **Alexandru Mihai GEOROCEANU** |

**București   
2015DECLARAŢIE PRIVIND ORIGINALITATEA LUCRĂRII**

Prin prezenta declar că lucrarea de disertație cu titlul “*Platformă WEB pentru Jocuri în Timp Real*” este elaborată de mine şi este rezultatul propriei mele activităţi de documentare şi cercetare. Declar de asemenea că ea nu a mai fost prezentată parțial sau integral la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate.

De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare:

* toate fragmentele de text reproduse exact, chiar dacă sunt traduceri proprii din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
* reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă la sursă;
* codul sursă, imagini etc. preluate din proiecte *open*-*source* sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;
* rezumarea ideilor altor autori precizează referinţa precisă la textul original.

București *2015*

Absolvent *Alexandru Mihai GEOROCEANU*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Cuprins**

Introducere 1

CAPITOLUL 1. Dezvoltarea pentru cloud 2

1.1 Introducere în cloud computing 2

1.2 Tipurile de servicii în Cloud Computing 3

1.3 Ce probleme rezolvă Docker 3

1.3.1 Problema mediului 3

1.3.2 Actualizarea software-ului 4

1.3.3 Trimiterea actualizărilor în medii de test / productive 5

CAPITOLUL 2. Arhitectura aplicației 7

2.1 Arhitectura infrastructurii 7

2.1.1 Imagine de ansamblu 7

2.1.2 Securitatea mediului 7

2.1.3 Scalarea resurselor per număr de utilizatori 7

2.2 Arhitectura software 7

CAPITOLUL 3. Prezentarea aplicației 8

3.1 Bla bla bla 8

3.1.1 Blu blu blu 8

3.1.2 Bleb le ble 8

CAPITOLUL 4. Concluzii 9

Bibliografie 10

Anexe 11

# Introducere

Prezenta lucrare demonstrează eficacitatea noilor tehnologii web – WebSockets, JavaScript pe server – și posibilitatea facilă de a utiliza o soluție ce poate scala orizontal fie pe mașini proprii, fie pe soluții *cloud* comerciale precum Amazon Web Services, IBM SoftLayer, Microsoft Azure și altele.

S-a ales utilizarea unui mediu gol pe mașina de dezvoltare, neinstalând niciun program din cele necesare rulării aplicației – server web, de structuri de date sau server de aplicație – și s-a folosit *Docker* cu imagini oficiale atât pentru mediul local cât și de producție. În acest fel se obține rapid o suită de programe instalate independent (fiecare într-un container separat), ușor de descărcat și de actualizat. Mai mult, instalările sunt verificate și actualizate periodic de către producătorii aferenți.

Global, tot mai multe persoane au acces la Internet. Un joc online, de exemplu, poate deveni oricând *viral* și, negândind de la bun început o arhitectură ce se poate scala orizontal, sistemul se poate supra-aglomera rapid și chiar închide din cauza traficului. Soluția prezentată în această lucrare este de tip *3-tier*: unul sau mai multe servere cu rol de *proxy* și *load-balancer* (nginx), unul sau mai multe servere care rulează aplicația (node.js) și unul sau mai multe servere de structuri de date (redis) prin care se face legatura între nodurile de aplicație.

Aplicația rulează în toate navigatoarele moderne de Internet (Chrome, Safari, Firefox) folosind cea mai eficientă (și nouă) tehnologie de comunicare client-server, cu posibilitatea serverului de a trimite mesaje clienților (server push) fără supraîncărcarea generată de metodele clasice – periodic pooling – și cu un plus major de performanță, nerenunțând nici la securitate. Conexiunea folosită se numește WebSocket și este un protocol nou *full-duplex* peste o singură conexiune TCP. Aceasta este o conexiune persistentă spre deosebire de conexiunile HTTP prin care se transportă, spre exemplu, resursele statice precum fișiere JavaScript, CSS, HTML.

Arhitectura aplicației se bazează pe evenimente – coechipierul s-a conectat, a făcut o mutare, a scris un mesaj pe chat ș.a.m.d. – care sunt captate și interpretate de server și clienți. Toate acțiunile sunt validate în continuare de către serverul de aplicație, așa încât nu sunt permise mutări ilegale sau alte acțiuni prin care un jucător să poată trișa.

# Dezvoltarea pentru cloud

## Introducere în cloud computing

Începuturile serviciilor informatice au fost dominate de computere tip *mainframe* create de către IBM și serveau în principal clienților de tip corporație sau băncilor, deoarece aceștia aveau nevoie să suporte un număr mare de clienți concurenți conectați și/sau tranzacții concurente.

Potrivit începuturilor serviciilor informatice, când tehnologia costa foarte mult, un asemenea *mainframe* era deseori doar închiriat pentru niște sume exorbitante de bani, cumpărarea unui asemenea hardware fiind o opțiune chiar cu mult mai costisitoare și oricum mai puțin rentabilă deoarece progresele rapide în tehnologie ar fi făcut ca scumpa mașină să fie depășită relativ rapid.

Legea lui Moore a rămas validă de-a lungul vremii, însă performanța nu a crescut liniar în viteză a procesorului – așa cum era inițial de așteptat. În schimb, producătorii au început să creeze procesoare cu mai multe nuclee și să păstreze o frecvență relativ mică a acestora. Această nouă abordare a dus la dezvoltarea accelerată a *parallel computing*, inginerii software punând mai nou accentul pe programele ce folosesc mai multe fire de execuție în paralel. Bineînțeles, creșterea pe orizontală a puterii de procesare a implicat și posibilitatea sporita de multi-tasking, utilizatorii fiind mult mai confortabili în ziua de astăzi să navigheaza de exemplu pe Internet în timp ce pe fundal se petrece o operație de durată ce solicită intensiv procesorul.

Cu același principiu s-a trecut în ziua de astăzi la un nivel mai mare: scalarea cu totul a mașinilor fizice pe orizontală. Dacă o aplicație sau un server suportă acest tip de scalare, atunci indiferent de numărul de utilizatori, se vor putea adăuga oricâte servere de același tip, nefiind nevoie de o așteptare în creștere a performanței produselor hardware.

Primul și cel mai important actor (chiar și astăzi) pe piața distribuitorilor de servicii cloud computing este Amazon. Acesta are o filozofie „pay per usage” așa încât utilizatorul nu are nevoie de abonamente sau contracte, ci pur și simplu plătește în funcție de volumul de resurse consumate. Resursele pot fi mașini virtuale deschise în funcție de bunul plac al utilizatorului – atât ca specificații tehince (memorie RAM, procesor, spațiu pe disk) cât și servicii specializate precum baze de date preconfigurate, servere DNS, servere de email și așa mai departe.

## Tipurile de servicii în Cloud Computing

SaaS (Software as a Service) este un model de serviciu cloud computing unde a un distribuitor licențiază o aplicație către utilizatorii săi pentru a fi utilizată ca și serviciu „on demand”. Un exemplu de SaaS este aplicația CRM (content relationship management) Salesforce.com.

IaaS (Infrastructure as a Service) este oferirea unei infrastructuri computerizate (de regulă o platformă pentru virtualizare de mediu) ca și serviciu. În loc de a cumpăra servere, software, spațiu în data center sau echipament de rețea, clienții cumpără în schimb aceste resurse ca și serviciu externalizat. Un asemenea distribuitor de servicii IaaS este Amazon web services.

PaaS (Platform as a Service) este oferirea unei platforme de calcul și stivă de soluții ca serviciu. Acesta facilitează lansarea aplicațiilor fără costul și complexitatea de a cumpăra și administra straturile hardware și software din spate. PaaS oferă resursele necesare întregului ciclu de viață a dezvoltării și lansării aplicațiilor și serviciilor web. Un exemplu de PaaS este GoogleApps.

Docker funcționează atât peste mașini virtuale cât și pe mașini „bare metal” deopotrivă. Momentan Amazon și Google oferă servicii pentru a folosi Docker în stadiu alfa, refolosind serviciile de mașini virtuale deja existente, cu aceleași prețuri / servicii folosite. Acest lucru practic anulează beneficiul costurilor scăzute care ar fi provenit din evitarea mașinilor virtuale și rularea doar a containerelor.

## Ce probleme rezolvă Docker

### Problema mediului

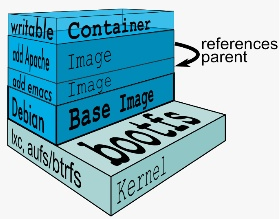
Fiecare mediu arată diferit – laptop dezvoltare 1, laptop dezvoltare 2, mediu de testare, producția, ș.a.m.d.

Fiecare dezvoltator are nevoie de lucruri diferite – de exemplu, dezvoltatorul ce lucrează la back-end are nevoie de alte unelte de dezvoltare față de cel care lucrează la partea de front-end a aplicației. Ideal, aceste medii ar trebui sa fie identice, așa încât să nu mai existe vechea scuză în cazul erorilor de aplicație „la mine funcționează”.

Problemele acestea se pot rezolva și cu ajutorul mașinilor virtuale, însă acestea din urmă prezintă alte costuri suplimentare în special în ceea ce privește performanța: au nevoie de foarte multe resurse și de timp pentru a porni sistemele de operare, au nevoie de regulă de foarte mult spațiu pe disc, partajarea de mașini virtuale nu este deloc facilă deoarece dimensiunile acestora tind să crească rapid către mai mulți gigaocteți.

Docker rezolvă problemele de performanță folosind o apariție relativ recentă în lumea Linux – containere LXC, suportul pentru acestea apărând în versiunea 3.8 de Kernel Linux. Acestea containere partajează kernel-ul existent cu mașina gazdă și rulează procese într-un mod izolat dar fără să deschidă sau să emuleze efectiv o mașină virtuală nouă cu absolut tot ceea ce presupune procesul de boot. Acest lucru scurtează pornirea „la rece” a unei aplicații complete de la câteva minute cu o mașină virtuală la câteva secunde cu containerele LXC.

Pentru a rezolva problema spațiului, Docker folosește AuFS (Advanced Multi-Layered Unification Filesystem), un sistem de fișiere format din „imagini” doar în citire, stivuite unul peste celălalt (ca în figura 1).



Figura

Spre deosebire de mașinile virtuale, o imagine se poate rula în oricâte instanțe – se va crea câte un container cu o imagine din vârf modificabilă și i se va aloca acesteia una sau mai multe adrese IP pentru accesul la rețea și comunicarea cu celelalte containere care rulează.

### Actualizarea software-ului

Pentru crearea unei imagini Docker, se poate folosi un fișier în format standard – Dockerfile. Cu comanda *docker build .* se poate crea o imagine nouă plecând de la instrucțiunile din fișierul Dockerfile din directorul curent. Aceasta se poate apoi partaja pe un repository public (hub.docker.com) sau unul privat.

Recrearea imaginii se face fie transmițând fișierul Dockerfile și rulând comanda de build, fie cu ajutorul repository-ului, de unde se pot descărca imaginile complete.

Actualizările vor conține întotdeauna doar modificările față de imaginea precedentă, astfel încât se vor face rapid și eficient. Mai mult decât atât, se poate vedea folosi comanda *docker diff* pentru a inspecta fișierele modificate în container.

Pentru cele mai populare resurse open-source (redis, ubuntu, MySQL, Postgres, CentOS, node.js și altele) există deja imagini oficiale urcate pe Docker Hub de către persoanele răspunzătoare cu mentenața proiectelor. Aceste imagini sunt actualizate periodic, create și optimizate de către dezvoltatori profesioniștii fiecărui proiect, fiind gata de a rula atât pe mediile de dezvoltare cât și în mediile productive deopotrivă.

Folosind comanda *docker pull nume-imagine* se vor căuta și descărca actualizările pentru imaginea dată ca parametru (exemple sunt redis:latest, node:0.11-slim, node:latest ș.a.). Imaginea curentă va rămâne pe disc, așa încât utilizatorul să poată rula aplicația cu noua versiune de software și, dacă întâmpină probleme sau testele automate nu validează, să rămână la imaginea cu care aplicația funcționează corect.

### Trimiterea actualizărilor în medii de test / productive

Cea mai facilă metodă pentru a distribui

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Coloana 1 | Coloana 2 | Coloana 3 | Coloana 4 | Coloana 5 |
|  |  |  |  |  |

**Tabelul 1.1.** Denumirea tabelului.

### Eliminarea surprusului virtualizării

Macro-benchmark-urile sunt cote de referință care se concentrează pe componente interconectate, cum ar fi infrastructura aplicației ca un întreg și nu doar executarea unui singur algoritm într-un anume limbaj de programare (micro-benchmark). Adevărata diferență dintre tehnologiile de virtualizare poate fi deseori găsită în felul în care mașinile virtuale comunică una cu cealaltă și cum încărcarea unei anumite mașini virtuale o poate influența pe alta.

Mașinile virtuale depind de infrastructura aplicației și îndeplinesc sarcina lor predefintiă. Această infrastructură poate fi construită peste un grup de mașini fizice, toate rulând sistemul de operare CoreOS sau XenServer.

Aceste implementări software „low level” sunt nucleul pe care un centru de calcul este construit. Apariția Docker a făcut folosirea containerelor LXC considerabil mai ușoară.

Pentru a compara Xen cu LXC au fost instalate XenServer 6.2 și CoreOS 324.3.0 cu Docker 0.11.1 pe două mașini identice echipate cu 4GB de RAM și un procesor Intel Xeon Quad core. Pe acestea au rulat sistemul de operare Ubuntu 12.04 în mașini virtuale sau containere.

Mașinile virtuale care au rulat XenServer folosesc un kernel paravirtualizat și au access la 2 nuclee ale procesorului. Prima mașină virtuală funcționează ca și un server de aplicație – are acces la 2GB de memorie RAM și rulează Apache 2.2, PHP 5.3 și WordPress 3.9. A doua mașină virtuală are acces la 1GB de RAM și rulează serverul de baze de date MySQL 5.5 având conținutul implicit pe care îl oferă WordPress.

Testul de performanță s-a axat pe performanța aplicației când este folosită de un număr în creștere de utilizatori. A fost folosit JMeter pentru a genera un număr mare de conexiuni simultane. Mașina gazdă a fost observată folosind top2, monitorizarea de la NewRelic și software XenServer.

Arhitectura aplicației

## Arhitectura infrastructurii

### Imagine de ansamblu

### Securitatea mediului

### Scalarea resurselor per număr de utilizatori

## Arhitectura software

# Prezentarea aplicației

## Bla bla bla

### Blu blu blu

### Bleb le ble

# Concluzii

# Bibliografie

1. Carl de Boor, *A practical guide to splines*, Springer-Verlag, New York Heidelberg Berlin, 1978.
2. R.A. DeVore and G.G. Lorentz. *Constructive Approximation*. Springer Verlag, Berlin Heildelberg New York, 1993.
3. T. Popoviciu. Introduction à la théorie des différences divisées. *Bull. Math. de la Soc. Roumaine des Sci.,* XLII: 65–78, 1940.
4. Titlu descriptiv, http://www.clasificare.com, (data accesării: zz.ll.aaaa).

# Anexe

Anexele vor conţine elemente precum:

* porţiuni de cod;
* tabele de date;
* tabele de rezultate de ieşire pe baza cărora s-au efectuat comparaţii sau s-au tras diverse concluzii în cadrul lucrării;
* alte elemente specifice la care s-a făcut referire în lucrare.