UNIVERSITATEA ALEXANDRU IOAN CUZA IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE DISERTAȚIE

A study on point in time recovery on relational databases

propusă de

Ștefan Stan

Sesiunea: *Iulie, 2017*

Coordonator ştiinţific

Lect. dr. Cristian Frăsinaru

UNIVERSITATEA ALEXANDRU IOAN CUZA IAȘI

FACULTATEA DE INFORMATICĂ

A study on point in time recovery on relational databases

Ștefan Stan

Sesiunea: *Iulie, 2017*

Coordonator ştiinţific

Lect. dr. Cristian Frăsinaru

Supervizor companie

DevOps Engineer Gabi Ichim, TSS-Yonder

DECLARAŢIE PRIVIND ORIGINALITATEA ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că lucrarea de desertație cu titlul "A study on point in time recovery on relational databases" este scrisă de mine şi nu a mai fost prezentată niciodată 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, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar şi în traducere proprie 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;
- 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.

Iași,

Absolvent *Ștefan Stan*

_____

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca lucrarea de disertație cu titlul "A study on point in time recovery on relational databases", codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.
De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de disertație.

Iași,

Absolvent *Ștefan Stan*

_____

Cuprins

Introducere

Avand în vedere contextul unui dezvoltator software, am observat în timpul experienței faptul ca pentru echipa efectivă de dezvoltare era destul de dificil de întreținut baza de date (incluzând deasemenea și operațiile de pornire/oprire, mai ales pentru juniori acest lucru putea pune probleme dacă baza de date se afla instalată pe o platforma UNIX). În cazul în care se dorea aplicarea unei strategii de point in time recovery pentru restabilirea bazei de date la o stare existentă la un moment dat, era nevoie de intervenția echipei de devOps, care se ocupa în mod expres de acest tip de operații, având cunoștințe avansate despre subiect.

Astfel că, am observat necesitatea existenței unei aplicații care să ușureze operațiile asupra bazei de date, începand cu pornirea/oprirea/verificarea starii, până la întreaga operație de backup și point in time recovery(întoarcere la o stare existentă la un anumit moment dat). Această aplicație trebuia să fie intuitivă, ușor de folosit chiar și pentru cineva care nu avea cunoștințe de devOps, nefiind necesar accesul efectiv pe mașina de UNIX unde se afla clusterul de baze de date. Este de la sine înteles că aplicația ar trebui să detecteze toate bazele de date din clusterul de pe mașina pe care este încarcată și ar trebui să permită operațiile prezentate mai sus pentru fiecare instanță în parte.

Impactele unei astfel de aplicații sunt atat de natură tehnică, le oferă dezvoltatorilor mai mult control asupra sistemului necesitând mai puține cunoștințe, precum și de business, nu mai trebuie angajată o echipă dedicată de devOps și prin urmare se reduc costurile.

Contribuții

În crearea aplicației am văzut o necesitate exitentă întâlnită la dezvoltatorii de software și am încercat să o rezolv. Procesul general de restabilire a unei baze de date este descris în *Capitolul 1*.

În *Capitolul 2* se prezintă într-o manieră detaliată o metodă de organizare a sistemului de fișiere UNIX, urmând ca în *Capitolul 3* să fie discutată arhitectura aleasă pentru dispunearea clusterului de baze de date pe sistemul de fișiere, precum și avantajele aduse de aceasta.

Soluția software este trecută în revistă în *Capitolul 4*. Astfel că, se prezintă arhitectura aplicației, urmată de fiecare componentă ce intră în alcătuirea ei. Tot în acest capitol este descrisă și setarea aplicației și a mediului de lucru în Amazon Web Services.

Capitolul 6 oferă un ghid detaliat al aplicației, urmând ca în *Capitolul 7* să fie discutate direcțiile de dezvoltare viitoare și idei de îmbunătățire ale platformei dezvoltate.

Point in time recovery

"Point in time recovery" în contextul informaticii se referă la funcționalitatea unui sistem folosită de un administrator pentru a restabili sau recupera un set de date existent la un moment dat.

PostgreSQL menține în fiecare moment un *write ahead log* WAL în subdirectorul pg_xlog al directorului de bază unde avem clusterul de baze de date. Acest log reține fiecare schimbare facută în fișierele bazei de date. Există, în principiu, pentru momentele în care sistemul cedează. Dacă acest lucru se întampla, datele pot fi recuperate prin o executare a secventei instrucțiunilor din log de la momentul ultimei salvări până înainte de momentul critic. Totuși, existența acestui log face posibilă o a treia strategie de restaurare a bazei de date: putem combina un backup la nivel de sistem de fișiere a bazei de date cu un backup al fișierelor WAL. Dacă avem nevoie să recuperăm datele vom restaura backup făcut la nivel de sistem de fișiere și apoi vom executa o secvență de fișiere WAL pentru a aduce baza de date la o versiune stabilă. Această abordare este complexă și dificil de setat, dar are multe beneficii:

- Nu avem nevoie de o stare consistentă a backup la nivel de sistem de fișiere. Orice inconsistentă va fi remediată prin executarea secvenței de fișiere WAL (nu este cine știe ce diferit de ceea ce se întamplă în momentul în care sistemul își revine după ce a cedat). Astfel că, avem nevoie de un utilitar capabil să creeze salvări pe disc, tar zip etc.

- De vreme ce putem folosi o secvență nedefinită de fișiere WAL, backup continuu se poate realiza pur și simplu arhivând fișierele WAL. Această tehnică este foarte folositoare în cazul bazelor de date mari, unde nu este avantajos să facem backup complet în mod frecvent.

- Nu este necesar să executăm toată secvență de fișiere WAL. Putem să ne oprim la orice moment din timp în care baza de date era consistentă. Astfel că, această tehnică suportă *point in time recovery*. Este posibil să restaurăm baza de date la orice moment existent dacă avem fișiere WAL.

- Daca oferim încontinuu fișiere WAL unei mașini noi ce a fost încărcată de la aceeași salvare, putem avea un sistem aflat în hibernare pe care îl putem aduce în orice moment în funcțiune având setul de date aflat în producție.

Pentru a recupera date cu succes folosind strategia de arhivare continuuă este nevoie de o secvența continuuă de fișiere WAL arhivate ce există cel putin de la momentul în care am creat acel basebackup (o stare a bazei de date aflata pe disc, un checkpoint). Pentru a începe e nevoie să setam și să testăm procedura de salvare a fișierelor WAL, înainte de a face primul basebackup. In cele ce urmează vom discuta mecanismul de arhivare a fișierelor WAL.

1.1 Setarea arhivarii fișierelor WAL

Într-un mod abstract, un sistem PostgreSQL produce o secvență nedefinită de înregistrări WAL. Sistemul împarte această secvență în fișiere WAL, care sunt în mod normal de 16MB fiecare (deși dimensiunea segmentului poate fi alterată cand facem build la PostgreSQL). Fișierele segment au numele într-un format numeric care reflectă poziția lor în secvența abstractă WAL. Când nu folosim arhivarea fișierelor WAL sistemul creează cateva fișiere pe care le reciclează redenumindu-le pe cele vechi și marcându-le pentru ștergere.

Când arhivăm datele WAL trebuie să prindem conținutul fiecarui segment de îndată ce devine complet și să salvăm datele undeva până ca segmentul devine reciclat și refolosit. În funcție de aplicație și harware folosit, pot exista mai mult moduri să "salvăm datele undeva": putem copia segmentul într-o locație pe un disc extern, le putem trimite în cloud pe un cont de Amazon S3, Google Drive... le putem trimite prin rețea oriunde etc. PostgreSQL încearca să nu facă supoziții despre modul de arhivare a datelor pentru a facilita munca administratorului. În schimb, PosgreSQL lasă administratorul să ofere o comandă shell ce se va executa pentru a copia orice segment complet oriunde trebuie să meargă. Această comandă poate fi la fel de simpla ca un **cp**, sau se poate invoca un script complex de linux.

Pentru a activa arhivarea WAL trebuie să se parcurga o serie de pași:

- să se seteze **wal_level** ca fiind **archive** sau mai mare. (wal_level determină câtă informație se scrie în fiecare WAL. Implicit e setat ca fiind **minimal**, ceea ce înseamnă că se scrie doar informația necesară pentru a se restaura dupa o problemă sau dupa o închidere imediată; **archive** adaugă informații necesare pentru arhivarea WAL; **hot_standby** adaugă informații necesare pentru a rula interogari în mod citire pe un server aflat in pauza; **logical** adaugă informații necesare pentru a suporta decodare logică)

- să se specifice o comandă shell în parametrul **archive_command**. În comanda de arhivare, parametrul %p este înlocuit cu calea către fișierul

WAL, în timp ce parametrul %f este înlocuit cu numele fișierului. Calea este relativă la fișierul curent (fișierul în care se află clusterul de baza de date). Cea mai simplă comandă este de genul:
archive_command = 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f'
Comanda va copia segmentul WAL în directorul /mnt/server/archvedir.

După ce s-au înlocuit parametrii %p si %f cmd. ce fa fi exec. va arata astfel:
test ! -f /mnt/server/archivedir/00000001000000A900000065 && cp pg_xlog/00000001000000A900000065 /mnt/server/archivedir/00000001000000A900000065
O comandă similară va fi executată pentru fiecare segment nou ce fa fi generat. Comanda de arhivare va fi executată sub același utitizator ca și serverul PostgreSQL. De vreme ce seria de fișiere WAL conține efectiv datele din baza de date, va fi nevoie ca datele arhivate să fie protejate de către accesul neautorizat. Putem să nu oferim drept de citire pentru grup și pentru ceilalți.

Comanda de arhivare este important să returneze statusul 0. PostgreSQL, cand primește 0 va crede că fișierul a fost arhivat cu succes și îl va șterge/ marca pentru reciclare. Însă, un status diferit de 0 va spune PostgreSQL ca arhivarea a eșuat și acesta va păstra fișierul reîncercând ulterior.

Comanda de arhivare ar trebui construită în așa fel încât să refuze să suprascrie fișiere pre-existente. Această funcționalitate este o metodă de siguranță prin care se asigură integritatea datelor în cazul unei erori a administratorului (trimite rezultatul de pe doua servere în același director).

Este de preferat să se testeze comanda propusă pentru arhivare, astfel că aceasta sa nu suprascrie fișiere și să returneze status diferit de 0 cand se încearcă acest lucru. Comanda oferită ca exemplu mai sus asigură acest lucru prin pasul de testare inițial. Unix deține pe anumite platforme opțiuni precum **-i** ce pot fi folosite pentru a realiza același lucru, dar nu trebuie să ne bazam pe ele ci să verificăm codul returnat de fiecare dată.
Aceste setări vor fi modificate în fișierul **postgresql.conf**

1.2 Crearea unui basebackup

Un basebackup este o salvare pe disc la un moment dat a bazei de date, de la care se va porni procesul de executare a fisierelor WAL.

Cel mai ușor mod de a crea un basebackup este folosirea utilitarei **pg_basebackup**. Se poate crea un basebackup fie ca și fișiere normale pe disc sau ca și arhivă(tar). Nu este neaparat nevoie să fim îngrijorați de timpul necesar crearii basebackup. Totuși, dacă serverul rulează cu opțiunea **full_page_writes** dezactivată s-ar putea observa o scadere în performantă, de vreme ce procedura de basebackup rulează automat cu opțiunea **full_page_writes** activată în mod forțat.

Pentru a ne folosi de basebackup trebuie să reținem toate fișierele WAL din timpul creării basebackup și după. Pentru a ne ajuta în acest proces, procedura de basebackup creează un fișier backup_history ce este mutat imediat în locația fișierelor WAL. Fișierul este denumit după primul fișier care este necesar pentru recovery folosind acel basebackup. Odată ce procesul de basebackup s-a terminat cu succes nu mai este nevoie de fișierele WAL de dinainte de crearea basebackup, de vreme ce avem o stare stabilă a bazei de date la momentul creării basebackup, de la care se va porni un proces ulterior de executare a fișierelor WAL pentru revenirea în caz de dezastru a bazei de date.

De vreme ce trebuie să salvăm fișierele WAL arhivate până la momentul ultimului basebackup înseamnă că intervalul între care se creează basebackups trebuie ales în funcție de cât spațiu de stocare vrem să folosim pentru a salva fișierele WAL. De asemenea, este de considerat și timpul pe care vrem sa îl alocam unui eventual proces de restabilire a bazei de date. Dacă este necesar acest lucru va trebui să executam fiecare fișier WAL și ar putea dura destul de mult în caz că ultimul basebackup a fost facut acum mult timp.

1.3 Restabilirea db. folosind Continuous Archive Backup

Să presupunem că ce e mai rau s-a intamplat și e nevoie să restabilim baza de date. Procedura este urmatoarea:

1. Se oprește serverul, dacă ruleaza.
2. Se copie conținutul fișierul **pg_xlog** ce poate conține fișiere log cu motivele pentru care sistemul a ajuns într-o stare inconsistentă, precum și segmentele WAL nearhivate (ce nu au atins pragul necesar (implicit 16MB) pentru a fi arhivate)
3. Se șterge orice fișier din locația clusterului de baze de date.
4. Se copie datele ultimului basebackup în locația clusterului de baze de date. Trebuie verificate drepturile de acces la fișierele/directoarele copiate.
5. Se șterge conținutul directorului pg_xlog copiat din basebackup, deoarece nu este de actualitate. Se copie fișierele din directorul pg_xlog salvat anterior.
6. Se creează un fișier **recovery.conf**. De asemenea se va modifica și fișierul **pg_hba.conf** pentru a nu da voie utilizatorilor obișnuiți să se conecteze până când procesul de recuperare nu se termină.
7. Se pornește server-ul. Acesta va intra automat în mod de recuperare și va începe să execute fișierele WAL, dacă există. Daca procesul eșuează din motive externe, se poate restarta serverul și va continua. În momentul în care procesul se termina se va redenumi fișierul **recovery.conf** în **recovery.done** pentru a preveni porniri ulterioare ale procesului de recuperare. Baza de date va funționa normal de acum.
8. Se inspectează baza de date pentru a se vedea dacă starea este una consistentă și dorită.

Procesul se bazează pe setarea unui fișier de configurare **recovery.conf** ce specifică cum să se execute procesul și până când anume. Singurul lucru neaparat necesar în fișierul de restaurare este **recovery_command** care specifică ce comandă shell se va executa pentru a recupera fișierele WAL. Aceasta este construită precum **archive_command** prezentată anterior.
De exemplu: **restore_command = 'cp /mnt/server/archivedir/%f %p'**

Logical Volume Manager (LVM)

In zilele de inceput ale serverelor Linux se facea managementul spatiului de stocare prin crearea de partitii pe disc. Chiar daca aceasta abordare functioneaza, ea prezinta anumite dezavantaje, iar cea mai importanta este aceeaa ca discurile sunt inflexibile. Acesta este motivul pentru care a fost introdus Logical Volume Manager. Chiar daca nu este posibil sa schimbam in mod dinamic marimea unei partitii care ramane fara spatiu, acest lucru este rezolvat lucrand cu LVM. LVM ofera multe avantaje ce vor fi prezentate in cele ce urmeaza.

2.1 Arhitectura LVM

In arhitectura LVM se pot distinge mai multe layere. Primul nivel lucreaza direct cu componentele de stocare. Acesta este creat intr-un mod abstract si poate folosi orice medii de stocare.

Mediile de stocare trebuie marcate ca "physical volumes", ceea ce le face sa poata fi folosite de catre LVM. O componenta de stocare marcata ca "physical volume" poate fi adaugata unui "volume group", layer abstract peste toate mediile de stocare. Abstractizarea consta in faptul ca acesta nu este ceva fix, poate fi redimensionat cand este necesar, ceea ce face posibil sa se adauge mai mult spatiu la nivelul "volume group" atunci cand "logical volumes" ce fac parte din el raman fara spatiu. Concret, daca se ramane fara spatiu pe un logical volume se ia din spatiul liber de la nivelul volume group-ului din care face parte. In cazul in care nici la acel nivel nu mai exista spatiu liber se adauga un disc nou in volume group.

Intuitiv, deasupra nivelului de volume group se afla logical volumes. Acestea nu actioneaza ca medii de stocare directe, ele isi iau spatiul necesar din discurile aflate in volume group. Astfel ca, un logical volume poate avea spatiul de stocare distribuit pe mai multe physical volumes (medii de stocare).

2.2 Functionalitati LVM

Exista multe motive pentru care LVM este foarte util. Cel mai important motiv este acela ca ofera flexibilitate pentru managementul spatiului de stocare. Partitiile nu mai sunt limitate de catre spatiul de stocare al discurilor. De asemenea, este posibil sa se reduca marimea unui logical volume, dar doar daca sistemul de fisiere ce a fost creat suporta redimensionare. De exemplu, sistemul de fisiere Ext4 suporta redimensionare in timp ce XFX, care este sistemul de fisiere implicit de pe

RHEL (Red Hat Enterprise Linux) nu suporta.

Un alt motiv pentru care administratorii de sistem folosesc LVM este posibilitatea de a crea salvari. O salvare tine minte starea curenta a logical volume si poate fi folosita pentru il a restaura daca este nevoie.

Al treilea avantaj important este acela ca avem posibilitatea de a schimba usor elementele hardware stricate. Daca un hard disk se strica, datele pot fi mutate in cadrul volume group prin comanda pvmove si un nou harddisk se poate adauga dinamic fara a avea sistemul cazut.

2.3 Crearea Logical Volumes

Crearea Logical Volumes presupune crearea a trei layere in arhitectura LVM. In primul rand trebuie create PV(physical volumes), apoi urmeaza sa se creeze VG(volume group) la care se adauga physical volumes create anterior. In ultimul rand, trebuie create LV(logical volumes) pe care le vom folosi ca si "partitii" efective.

Inainte de a folosi utilitarele LVM trebuie sa marcam partitiile hardware ca fiind de tipul LVM. Daca se foloseste utilitara fdisk impreuna cu un disc de tip MBR, atunci partitia trebuie pusa de tipul 8e. Altfel, daca se foloseste gdisk impreuna cu un disc de tip GUID, atunci partitia este de tip 8300.

2.3.1 Pasi pentru a crea un Physical Volume

Avem nevoie de un harddisk care are spatiu liber (nepartitionat). Pentru scopul explicarii se va folosi un disk /dev/vdb pentru a crea partitia. Fiecare disc are un nume diferit astfel ca pasii vor trebui ajustati pentru fiecare disc.

- Se deschide un shell si se executa **fdisk /dev/vdb**
- Se apasa **n** pentru a crea partitia. Se selecteaza **p** pentru a o marca partitie primara si se accepta numerotarea implicita.
- Se apasa **Enter** de doua ori si astfel se creeza o partitie folosind tot spatiul de pe discul **/dev/vdb**
- Fiind intors la prompter-ul fdisk, se apasa t pentru a schimba tipul partitiei create. Fiind o singura partitie ce foloseste intreg spatiul aceasta va fi selectata implicit, altfel ar putea intreba care pentru care partitie sa se schimbe tipul.
- Utilitara fdisk intreaba ce tip va avea partitia aleaa. Scrie 8e. Apasa w pentru a scrie schimbarile efectuate in acesti pasi. Daca se primeste un

mesaj conform caruia tabela de partitii nu a putut fi actualizata trebuie dat un restart.

Acum ca partitia a fost creata trebuie marcata ca LVM physical volume. Pentru asta se va executa **pvcreate /dev/vdb1**. Ar trebui primit urmatorul mesaj: Physical volume "/dev/vbd1" successfully created. Pentru a verifica existenta physical volume nou creat putem executa in shell comanda pvs ce listeaza physical volume existente.

2.3.2 Pasi pentru a crea un Volume Group

Acum ca am creat un physical volume trebuie sa il asignam unui volume group. Este posibil sa adauga physical volumes la un volume group existent (a se citi manualul din linux pentru comanda **vgextend**), insa acum un volume group nou si vom adauga un physical volume la el. Aceasta este procedura relativ usoara, o singura comanda, vgcreate, ce primeste numele noului volume group precum si physical volume ce va fi adaugat initial. De exemplu: **vgcreate vgdata /dev/vdb1**.

Este complet la alegerea utilizatorului ce nume alege pentru volume groups. In acest exemplu am ales sa incep numele cu vg, ceea ce le face usor de gasit daca vre-odata avem foarte multe.

Dupa ce s-a create volume group putem executa comanda **vgs** pentru a lista volume groups existente.

2.3.3 Pasi pentru a crea Logical Volumes

Acum ca s-a creat un volume group putem incepe sa cream logical volumes. Procedura este un pic mai complicata decat crearea physical volumes sau volume groups, deoarece sunt mai multe alegeri ce trebuie facute. Cand se creeaza un logical volumes trebuie specificate numele si marimea.

Marimea logical volume poate fi specificata prin valoare absoluta folosind optiunea **-L**. De exemplu folosind **-L 5G** se creeaza un logical volume cu marimea de 5GB. O alternativa este sa folosim optiunea **- l** pentru a specifica o marime relativa. De exemplu, se foloseste **- l 50% FREE** pentru a folosi jumatate din spatiul liber la nivelul volume group. Urmatorul argument este numele volume group la care va fi asignat logical volume nou creat si din care se va lua spatiu necesar. Un parametru optional este **- n** ce specifica numele noului logical volume. De exemplu, comanda **lvcreate -n lvvol1 -L 100M vgdata** creeaza un logical volume cu numele lvvol folosind 100MB spatiu din volume group cu numele vgdata (creat anterior).

2.3.4 Modificarea marimii unui Logical Volume

Motivul folosirii logical volumes in locul utilizarii directe a unor discuri este acela ca la nevoie putem foarte usor sa adaugam spatiu intr-un logical volume, in

timp ce daca de exemplu avem de adaugat in cazul discurilor trebuie sa adaugam un alt disc mai mare si sa copiem totul din discul initial, moment in care sistemul trebuie sa fie oprit. Daca de exemplu avem un folder /tenant1 in care tinem datele din baza de date pentru un client. Sa ne imaginal ca acesta trimite la un disc fizic de 100GB. Pentru a putea sa oferim mai mult spatiu in acest director va fi nevoie sa oprim baza de date, sa mutam toate datele pe un alt disc de 200GB de ex si apoi sa repornim sistemul. In cazul in care /tenant1 este un director la care a fost montat un logical volume putem folosi urmatoare procedura pentru a mari spatiul fara a opri sistemulȘ

- Verificam ca in volume group din care face parte logical volume ce va fi extins se afla spatiu liber. Daca nu exista spatiu liber adauga un nou physical volume.
- Executam comanda **vgextend** pentru a extinde spatiul din logical volume.

Putem utiliza comanda **lvs** ce listeaza date despre logical volumes existente pentru a vedea spatiul liber al logical volume. Pentru detalii despre comanda **vgextend** a se executa comanda **man vgextend** ce afiseaza manualul din Linux.

2.4 Explicatii despre conventia de numire a LVM

Din acest moment putem incepe sa folosim logical volume nou creat. Pentru a face asta trebuie sa stim cum sa ne adresam acestuia. Entitatile LVM pot fi adresate in moduri multiple. Cea mai simpla metoda este sa adresam un logical volume precum: /dev/vgname/lvname. Astfel ca, daca am creat un logical volume cu numele lvdata, ce isi ia spatiu dintrun volume group cu numele vgdata, calea entitatii va fi /dev/vgdata/lvdata.

Device mapper poarta un rol important in denumirea LVM logical volumes. Device mapper (abreviat dm) este o interfata generica folosita de kernelul Linux pentru a adresa device de stocare. Device mapper este folosit de catre multiple sisteme: LVM volumes, dar si de software RAID etc. Aceste device sunt create doua locatii: device numerotate in secventa plasate in directorul /dev, precum /dev/dm-0, /dev/dm-1 si asa mai departe. Deoarece aceste denumiri nu ofera nicio informatie despre deviceuri ci confuzeaza, se folosesc scurtaturi create in directorul /dev/mapper. Acestea folosesc sablonul vgname-lvname pentru nume. Astfel ca, daca avem device /dev/vgdata/lvdata putem sa ne adresam folosind si /dev/mapper/vgdata-lvdata. Cand lucram cum LVM logical volumes putem folosi oricare din aceste tipuri de adresare.

Database Point in time recovery Manager

Din punctul de vedere al unui dezvoltator software am observat in timpul experientei faptul ca pentru echipa efectiva de dezvoltare era destul de dificil de intretinut baza de date (ma refer chiar si la pornit/oprit, mai ales pentru juniori acest lucru putea pune probleme daca baza de date se afla instalata pe o platforma UNIX). In cazul in care se dorea aplicarea unei strategii de point in time recovery pentru restabilirea bazei de date la o stare existenta la un moment dat, era nevoie de interventia echipei de devOps, ceea ce se factura diferit la client, prin urmare trebuia convins in prealabil Product Owner-ul de necesitatea acelei interventii si planificat totul.

Astfel ca, am observat necesitatea existentei unei aplicatii care sa usureze operatiile asupra bazei de date, incepand cu pornirea/oprirea/verificarea statusului pana la intreaga operatie de backup si point in time recovery(intoarcere la o stare existenta la un anumit moment dat). Aceasta aplicatie trebuia sa fie intuitiva, usor de folosit chiar si pentru cineva care nu avea cunostinte de devOps, nefiind necesar accesul efectiv pe masina de UNIX unde se afla clusterul de baze de date. Este de la sine inteles ca aplicatia ar trebui sa detecteze toate bazele de date din clusterul de pe masina pe care este incarcata si ar trebui sa permita operatiile prezentate mai sus pentru fiecare dintre instanta in parte.

3.1 Arhitectura sistemului de fisiere

Am propus o solutie pentru bazele de date PostgreSQL folosite pe platforme UNIX (la momentul de fata). Pentru a prezenta solutia software am avut nevoie de un mediu UNIX pe care sa ruleze un numar de instante PostgreSQL. Am ales o masina CentOS, instanta EC2 oferita de Amazon. Sistemul de fisiere a fost organizat folosind LVM (logical volume manager) dupa cum urmeaza:

- am cerut 3 discuri de cate 50GiB, Amazon EBS;
- am creat cate un physical volume din fiecare din cele 3 discuri;
- am creat un volume group si le-am adaugat acestuia;
- am creat 6 logical volumes, 3 seturi de cate 2 logical volumes; un logical volume de 20GiB pentru datele din db si un logical volume de 10GiB pentru stocarea fisierelor WAL.

Au mai ramas 60GiB nealocati, acestia putand fi folositi ulterior. Daca se observa ca un client are nevoie de mai mult spatiu i se pot extinde logical volumes folosind din spatiul nealocat, acesta fiind mare avantaj al LVM. Bazele de date nu vor fi intrerupte cat timp se face extinderea logical volumes.

Am setat astfel sistemul de fisiere pentru 3 instante diferite de baze de date PostgreSQL denumite lv-pgdata-1, lv-pgdata-2 si lv-pgdata-3. In *Figura 1* este prezentata diagrama sistemului de fisiere.

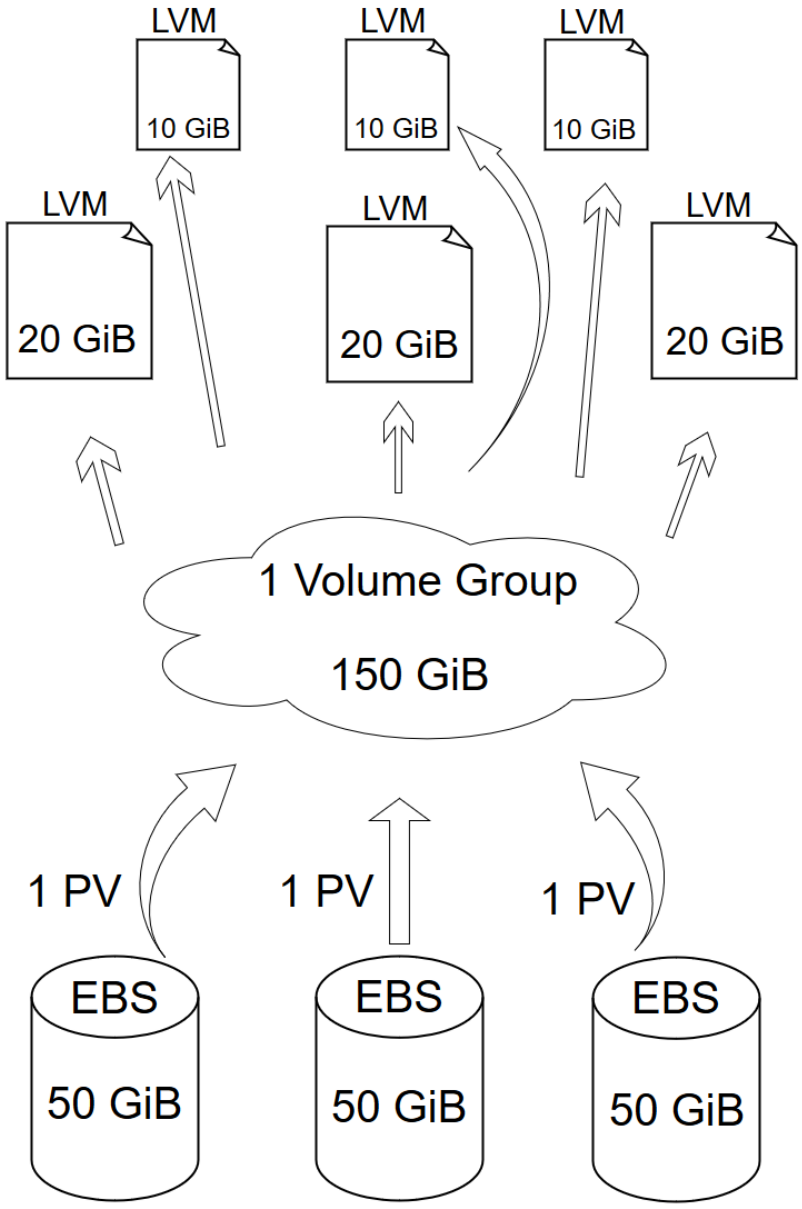

Figura 1: Diagrama sistemului de fisiere

DevOps as a Service

In cele ce urmeaza voi prezenta solutia software propusa pentru managementul bazelor de date in maniera descrisa anterior. Voi trece in revista arhitectura software, urmand apoi sa explic fiecare din componentele aplicatiei, precum si modul de interactionare a acestora.

4.1 Arhitectura Sistemului Software

Sistemul este impartit in 3 componente importante:

- Scripturi UNIX ce interactioneaza cu instantele de baze de date;
- RESTful API folosit pentru a expune functionalitatea oferita de scripturi prin intermediul protocolului HTTP;
- Aplicatie Web ce comunica prin intermediul protocolului HTTP cu API-ul RESTful expus.

In *Figura 2* este prezentata diagrama arhitecturii software.

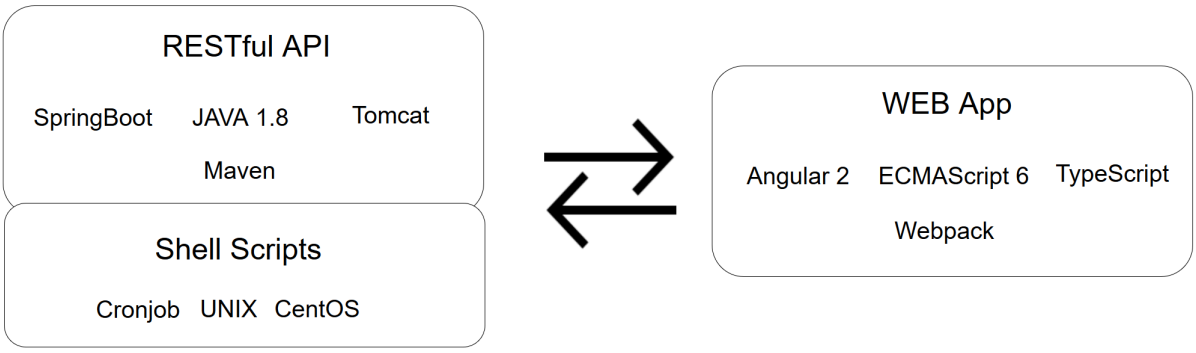

Figura 2: Diagrama arhitecturii software

4.2 Scripturile UNIX

Functionalitatea de baza a aplicatiei consta in interactionarea cu instantele de baze de date. Pentru asta am facut 3 scripturi ce au urmatoarele responsabilitati:

1. **pg-script.sh** pentru interactionarea cu instantele (pornire, oprire, interogare status);

```
#!/bin/bash

name="$(basename $0)"
pgdata="$(find /pgdata -type d -name $name)"
logFile="$pgdata/pg_log/startup.log"

if [ ! -f $logFile ]
then
        "/bin/touch" "$logFile"
fi

"/usr/lib64/pgsql95/bin/pg_ctl" $1 "-D" $pgdata "-l" $logFile
```

Cod sursă 1: `pg-script.sh`

2. **basebackup.sh** pentru crearea de basebackup a unei instante (checkpoint de la care sa se poate incepe procesul de restaurare);

```
#!/bin/bash

if [ "$#" -ne 2 ]
then
        echo "Illegal number of parameters. 2 params: ip port"
        exit 1
fi

name="$(basename $0)"
pgdata="$(find /pgdata -type d -name $name)"
if [ ${#pgdata} == 0 ]
then
        echo "tenant with name $name has no pgdata folder"
        exit 2
fi

backup="$(find /backup -type d -name $name)"
if [ ${#backup} == 0 ]
then
        echo "tenant with name $name has no backup folder"
        exit 3
fi

basebackup="$backup/basebackup"
if [ -d $basebackup ]
then
        rm -rf $basebackup/*
        rm -rf $basebackup/.[a-z]*
else
        mkdir $basebackup
        chmod 700 $basebackup
fi

"pg_basebackup" "-h" $1 "-p" $2 "-D" $basebackup
```

Cod sursă 2: `basebackup.sh`

3. **recovery.sh** pentru initierea procesului de restaurare a bazei de date

```
#!/bin/bash

if [ "$#" -ne 2 ]
then
        echo "Illegal number of parameters. 2 params: date e.g.
                        '2017-06-05 14:00:00 EEST' true/false if you want to start the db also"
        exit 1
fi

name="$(basename $0)"
pgdata="$(find /pgdata -type d -name $name)"
if [ ${#pgdata} == 0 ]
then
        echo "tenant with name $name has no pgdata folder"
        exit 2
fi

backup="$(find /backup -type d -name $name)"
if [ ${#backup} == 0 ]
then
        echo "tenant with name $name has no backup folder"
        exit 3
fi

basebackup="$backup/basebackup"
if [ ${#basebackup} == 0 ]
then
        echo "tenant with name $name has no basebackup folder"
        exit 4
fi

wal="$backup/wal"
if [ ${#wal} == 0 ]
then
        echo "tenant with name $name has no WAL files folder"
        exit 5
fi

# TO DO: make sure that the server instance is stopped
"/home/ec2-user/disertatie/server/$name" "stop"

# TO DO: backup logs and other things of interes before empty pgdata dir
rm -rf $pgdata/*
rm -rf $pgdata/.[a-z]*

"cp" "-a" "-r" "$basebackup/." $pgdata

result="restore_command = 'cp $wal/%f %p'\n
recovery_target_time = '$1'"

echo -e $result > $pgdata/recovery.conf

if [ $2 == true ]
then
        # Start the db server which will trigger recovery
        "/home/ec2-user/disertatie/server/$name" "start"
fi
```

Cod sursă 3: `recovery.sh`

Toate cele trei scripturi au fost gandite cu scopul de a se crea symbolic links (scurtaturi) catre ele. Acestea vor avea numele instantei bazei de date, iar scriptul principal foloseste numele fisierului scurtatura pentru a accesa datele instantei respective. Scopul este acela de a crea un singur set de scripturi, iar daca avem n instante de baza de date doar creem symbolic links catre scripturi, fiecare cu numele instante de baza de date respective.

4.3 Server-ul: RESTful API

Pentru partea de server (backend) a fost aleasa solutia crearii unui RESTful API cu scopul de a se expune functionalitatea oferita de catre scripturile bash prezentate anterior. Am utilizat Java Enterprise Edition, framework-ul Spring, ce facilteaza crearea unui API prin intermediul librariilor SpringWeb.

Un alt motiv pentru care am ales Spring a fost suportul oferit pentru a injecta dependinte urmand sablonul de proiectare Dependency Injection. Astfel, am decuplat controller-ul REST ce primeste cererea web de catre serviciul care o executa. Se poate injecta orice implementare de serviciu in controller, iar codul acestuia nu trebuie schimbat. Versiunea de Java utilizata este 1.8.

```java
package pgapp.api;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import pgapp.dto.DatabaseManagementDetailsObject;
import pgapp.service.DatabaseManagementService;

import java.util.List;

/**
 * @author Stefan Stan on 06.06.2017.
 */
@RestController
@RequestMapping("/pgapp")
public class DatabaseManagementAPI {

    @Autowired
    private DatabaseManagementService dbService;

    @GetMapping("/server")
    public List<String> getDBsNames() {
        return dbService.getDBsNames();
    }

    @PostMapping("/server")
    public DatabaseManagementDetailsObject changeDB(
            @RequestParam("dbServerName") String dbServerName,
            @RequestParam("action") String action
    ) {
        return dbService.changeDB(dbServerName, action);
    }

    @PostMapping("/basebackup")
    public DatabaseManagementDetailsObject basebackupDB(
            @RequestParam("dbServerName") String dbServerName
    ) {
        return dbService.basebackupDB(dbServerName);
    }

    @PostMapping("/pitr")
    public DatabaseManagementDetailsObject recoverDB(
            @RequestParam("dbServerName") String dbServerName,
            @RequestParam("dateTime") String dateTime
    ) {
        return dbService.recoverDB(dbServerName, dateTime, true);
    }
}
```

Cod sursă 4: `DatabaseManagementAPI.java`

Pentru managementul dependintelor si crearea artefactului s-a folosit Maven. Se ofera ca si exemplu fisierul **pom.xml**:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.2.RELEASE</version>
    </parent>

    <groupId>ro.stefanstan.apps</groupId>
    <artifactId>pgApp</artifactId>
    <version>1.0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.jayway.jsonpath</groupId>
            <artifactId>json-path</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

    <repositories>
        <repository>
            <id>spring-releases</id>
            <url>https://repo.spring.io/libs-release</url>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>spring-releases</id>
            <url>https://repo.spring.io/libs-release</url>
        </pluginRepository>
    </pluginRepositories>
</project>
```

Cod sursă 5: `pom.xml`

4.4 Interfata grafica cu utilizatorul

Scopul proiectului este crearea unei aplicatii care sa faciliteze manipularea serverelor de baze de date PostgreSQL de catre dezvoltatori care nu trebuie sa aiba acces la masinile pe care se afla bazele de date/ cunostinte de UNIX. In acest sens, avand deja server-ul de Java care expune prin servicii Web functionalitatile respective, trebuie apelate si oferite intr-o maniera cat mai intuitiva.

Ca si framework am folosit Angular 2, limbajul de programare fiind Javascript, standardul ECMAScript6, impreuna cu TypeScript. Angular 2 are urmatoarele avantaje:

- este un framework open-source JavaScript construit și întreținut de Google, ceea ce usurează dezvoltarea aplicațiilor web.
- standardizează aplicațiile la nivel de client (browser), oferind o structură robustă și ușor de implementat în dezvoltarea site-urilor.
- este alegerea potrivită pentru orice aplicație web, în special dacă se preferă efectele vizuale spectaculoase. Rezultatul este un site fluid și rapid. Acesta este motivul pentru care se pretează SPA-urilor (Single Page Application) și aplicațiilor web destinate dispozitivelor mobile.
- se integrează ușor cu Bootstrap, permite crearea de aplicații responsive care pot fi accesate atât din browserul calculatorului cât și de pe dispozitivele mobile. Astfel, este ușor să creezi o singură aplicație web care să se vadă perfect pe orice dispozitiv de pe care este accesată – PC, tabletă, telefon, plasmă.

Ca si manager de pachete am utilizat **npm** de la NodeJS.

4.5 Setarea aplicatiei in Amazon Web Services

Avand ca si obiectiv crearea unui mediu cat mai apropiat de cel aflat pe masinile din productie, am inchiriat un server EC2 de la Amazon.

Sistemul de fisiere a fost stabilit conform descrierii din Capitolul 2. Am inchiriat 3 discuri EBS(Elastic Block Storage) pe care le-am configurat precum a fost prezentat, pentru a seta 3 instante de baza de date.

Odata ce sistemul de fisiere a fost configurat, am creat scripturile ce vor interactiona cu bazele de date. Urmatorul pas a fost pornirea celor doua servere (backend si frontend). In timpul acestui proces a fost necesara configurarea unui rol de securitate oferit masinii de calcul EC2, pentru a permite traficul pe portul 3001 utilizat de aplicatia frontend.

Ghid de utilizare a aplicaţiei

Solutia propusa pentru a rezolva problema prezentata in capitolele procedente este o aplicatie web folosita pentru managementul bazelor de date.

Aceasta ofera functionalitatile de vizualizare a starii instantelor de baze de date prin apasarea butonului de "Refresh". Pentru a restaura starea bazei de date este necesara crearea unui checkpoint, iar pentru aceasta se apasa butonul "Create basebackup".

Odata existent un checkpoint, se poate restaura baza de date respectiva prin alegerea unei date/ore folosind componenta vizuala de tip calendar, urmand sa se apese butonul "Revert".

Direcţii de dezvoltare viitoare

Aplicatia propusa pentru a rezolva necesitatea restaurarii bazelor de date la anumite momente in timp poate fi folosita doar pentru baza de date PostgreSQL.

Astfel ca, o prima idee de dezvolatare viitoare ar fi investigarea suportului oferit alte tipuri de baze de date Oracle, MySql etc. urmand ca apoi sa creez un design abstract in ceea ce priveste baza de date si care sa fie integrat cu solutia existenta oferita pentru PostgreSQL.

Cat despre interactiunea cu ultilizatorul, am folosit ultimele tehnologii actuale Angular 2, EcmaScript 6 etc. insa din punct de vedere al design, ar putea fi imbunatatit in ceea ce priveste html si css existent.

Concluzii

Am creat o aplicatie pornind de la necesitatea prezentata in capitolul *Introducere*. Scopul acesteia este de a oferi o metoda facila de a restaura o baza de date la un moment de timp oferit de catre utilizator. Acesta nu necesita cunostinte de baze de date, sau de devOps (nu e nevoie sa interactioneze cu scripturi de UNIX).

Pentru a realiza cele mentionate anterior am studiat modalitatea bazei de date PostgreSQL de management intern a transactiilor si am utilizat-o intro maniera ce imi permite restaurarea bazei de date la un moment in timp cu o granularitate fina, la secunda.

De asemenea, am studiat modalitatea sistemului de operare UNIX ce permite creare de partitii virtuale ce pot fi oricand modificate in ceea ce priveste marimea fara a inchide baza de date. Am combinat aceasta functionaliate a sistemului de operare impreuna cu solutia propusa, pentru a realiza cele prezentate cu un timp minim de inaccesibilitate a bazei de date. *Capitolul 3* prezinta in detaliu aceste aspecte.

Am propus o arhitectura a sistemului de fisiere bazat pe studiul prezentat anterior si am utilizat-o in implementarea propriu-zisa. Cat despre aceasta, am folosit serviciile Amazon Cloud (EC2, EBS) pentru a face deploy solutiei (scripturi ce interactioneaza cu baza de date, aplicatie backend si aplicatie frontend).

In opinia mea, prin studiul realizat si prin intermediul solutiei propuse am reusit sa realizez ce mi-am propus si am oferit o modalitate de management a bazelor de date si de restaurare a acestora la starea existenta un moment de timp. Prin urmare am realizat o aplicatie usor de utilizat de catre persoane ce nu au cunostinta de baze de date, sistem de operare UNIX.

Bibliografie

[1] Sander van Vugt, Red Hat® RHCSA™/RHCE® 7 Cert Guide: Red Hat Enterprise Linux 7 (EX200 and EX300)

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2005.

[3] Postgres Documentation

https://www.postgresql.org/docs/9.5/static/continuous-archiving.html

[4] Craig Walls, Spring in Action, Third Edition

[5] Martin Fowler. *GUI Architectures*. 18 July 2006.