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.

Studiu *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'**

Studiu *Logical Volume Manager (LVM)*

În zilele de început ale serverelor Linux se făcea managementul spațiului de stocare prin crearea de partiții pe disc. Chiar dacă această abordare funcționează, ea prezintă anumite dezavantaje, iar cea mai importantă este aceeaa că discurile sunt inflexibile. Acesta este motivul pentru care a fost introdus Logical Volume Manager. Chiar dacă nu este posibil să schimbăm în mod dinamic marimea unei partiții care ramane fără spațiu, acest lucru este rezolvat lucrând cu LVM. LVM oferă multe avantaje ce vor fi prezentate în cele ce urmează.

2.1 Arhitectura LVM

În arhitectura LVM se pot distinge mai multe layere. Primul nivel lucrează direct cu componentele de stocare. Acesta este creat într-un mod abstract și poate folosi orice medii de stocare.

Mediile de stocare trebuie marcate ca "physical volumes", ceea ce le face să poată fi folosite de către LVM. O componentă de stocare marcată ca "physical volume" poate fi adăugata unui "volume group", layer abstract peste toate mediile de stocare. Abstractizarea constă în faptul că acesta nu este ceva fix, poate fi redimensionat când este necesar, ceea ce face posibil să se adauge mai mult spațiu la nivelul "volume group" atunci când "logical volumes" ce fac parte din el rămân făra spațiu. Concret, dacă se rămâne fără spațiu pe un logical volume se ia din spațiul liber de la nivelul volume group-ului din care face parte. În cazul în care nici la acel nivel nu mai există spațiu liber se adaugă un disc nou în volume group.

Intuitiv, deasupra nivelului de volume groups se află logical volumes. Acestea nu actionează ca medii de stocare directe, ele își iau spațiul necesar din discurile aflate în volume group. Astfel că, un logical volume poate avea spațiul de stocare distribuit pe mai multe physical volumes (medii de stocare).

2.2 Funcționalități LVM

Există multe motive pentru care LVM este foarte util. Cel mai important motiv este acela că oferă flexibilitate pentru managementul spațiului de stocare. Partițiile nu mai sunt limitate de către spațiul de stocare al discurilor. Deasemenea, este posibil să se reducă mărimea unui logical volume, dar doar dacă sistemul de fișiere ce a fost creat suportă redimensionare. De exemplu, sistemul de fișiere Ext4 suportă redimensionare în timp ce XFS, care este sistemul de fișiere implicit de pe

RHEL (Red Hat Enterprise Linux) nu suportă.

Un alt motiv pentru care administratorii de sistem folosesc LVM este posibilitatea de a crea salvări. O salvare ține minte starea curentă a logical volume și poate fi folosită pentru a îl restaura dacă este nevoie.

Al treilea avantaj important este acela că avem posibilitatea de a schimba ușor elementele hardware stricate. Dacă un hard disk se strică, datele pot fi mutate în cadrul volume group prin comanda pvmove și 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 în arhitectura LVM. În primul rând trebuie create PV(physical volumes), apoi urmează să se creeze VG(volume group) la care se adaugă physical volumes create anterior. În ultimul rând, trebuie create LV(logical volumes) pe care le vom folosi ca și "partiții" efective.

Înainte de a folosi utilitarele LVM trebuie să marcăm partițiile hardware ca fiind de tipul LVM. Dacă se folosește utilitara fdisk împreună cu un disc de tip MBR, atunci partiția trebuie pusă de tipul 8e. Altfel, dacă se folosește gdisk împreună cu un disc de tip GUID, atunci partiția este de tip 8300.

2.3.1 Pași pentru a crea un Physical Volume

Avem nevoie de un harddisk care are spațiu liber (nepartiționat). Pentru scopul explicării se va folosi un disk /dev/vdb pentru a crea partiția. Fiecare disc are un nume diferit astfel că pașii vor trebui ajustați pentru fiecare disc.

- Se deschide un shell și se execută **fdisk /dev/vdb**
- Se apasă **n** pentru a crea partiția. Se selectează **p** pentru a o marca partiție primară si se acceptă numerotarea implicită.
- Se apasă **Enter** de două ori și astfel se creează o partiție folosind tot spațiul de pe discul **/dev/vdb**
- Fiind întors la prompter-ul fdisk, se apasă t pentru a schimba tipul partiției create. Fiind o singură partiție ce folosește întreg spatiul aceasta va fi selectată implicit, altfel ar putea întreba pentru care partiție să se schimbe tipul.
- Utilitara fdisk întreabă ce tip va avea partiția alesă. Scrie 8e. Apasă w pentru a scrie schimbările efectuate în acești pași. Dacă se primește un

mesaj conform căruia tabela de partiții nu a putut fi actualizată trebuie dat un restart.

Acum că partiția a fost creată trebuie marcată ca LVM physical volume. Pentru asta se va executa **pvcreate /dev/vdb1**. Ar trebui primit următorul mesaj: Physical volume "/dev/vbd1" successfully created. Pentru a verifica existența physical volume nou creat putem executa în shell comanda pvs ce listează physical volume existente.

2.3.2 Pași pentru a crea un Volume Group

Acum că am creat un physical volume trebuie să îl asignam unui volume group. Este posibil să adăugăm physical volumes la un volume group existent (a se citi manualul din linux pentru comanda **vgextend**), însă avem un volume group și vom adăuga un physical volume la el. Aceasta este procedura relativ ușoară, o singură comanda, vgcreate, ce primește numele noului volume group precum și physical volume ce va fi adăugat inițial. De exemplu: **vgcreate vgdata /dev/vdb1**.

Este complet la alegerea utilizatorului ce nume alege pentru volume groups. În acest exemplu am ales să încep numele cu vg, ceea ce le face ușor de găsit dacă vre-odată avem foarte multe.

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

2.3.3 Pași pentru a crea Logical Volumes

Acum că s-a creat un volume group putem începe să cream logical volumes. Procedura este un pic mai complicată decât crearea physical volumes sau volume groups, deoarece sunt mai multe alegeri ce trebuie făcute. Cand se creeaza un logical volumes trebuie specificate numele și mărimea.

Mărimea logical volume poate fi specificată prin valoare absolută folosind opțiunea **-L**. De exemplu, folosind **-L 5G** se creează un logical volume cu mărimea de 5GB. O alternativă este să folosim opțiunea **- l** pentru a specifica o mărime relativă. De exemplu, se folosește **- l 50% FREE** pentru a folosi jumătate din spațiul liber la nivelul volume group. Următorul argument este numele volume group la care va fi asignat logical volume nou creat și din care se va lua spațiu necesar. Un parametru opțional este **- n** ce specifică numele noului logical volume. De exemplu, comanda **lvcreate -n lvvol1 -L 100M vgdata** creează un logical volume cu numele lvvol folosind 100MB spațiu din volume group cu numele vgdata (creat anterior).

2.3.4 Modificarea mărimii unui Logical Volume

Motivul folosirii logical volumes în locul utilizării directe a unor discuri este acela că la nevoie putem foarte ușor să adăugam spațiu într-un logical volume, în

timp ce dacă de exemplu, avem de adăugat în cazul discurilor trebuie să adăugam un alt disc mai mare și să copiem totul din discul inițial, moment în care sistemul trebuie să fie oprit. Daca de exemplu avem un folder /tenant1 în care ținem datele din baza de date pentru un client. Să ne imaginăm că acesta trimite la un disc fizic de 100GB. Pentru a putea să oferim mai mult spațiu în acest director va fi nevoie să oprim baza de date, să mutăm toate datele pe un alt disc de 200GB de ex și apoi să repornim sistemul. În cazul în care /tenant1 este un director la care a fost montat un logical volume putem folosi urmatoare procedura pentru a mări spațiul fără a opri sistemul:

- Verificăm că în volume group din care face parte logical volume ce va fi extins se află spațiu liber. Dacă nu există spațiu liber, adaugă un nou physical volume.
- Executăm comanda **vgextend** pentru a extinde spațiul din logical volume.

Putem utiliza comanda **lvs** ce listează date despre logical volumes existente pentru a vedea spațiul liber al logical volume. Pentru detalii despre comanda **vgextend** sa se execute comanda **man vgextend** ce afișeaza manualul din Linux.

2.4 Convenția de numire a LVM

Din acest moment putem începe să folosim logical volume nou creat. Pentru a face asta trebuie să stim cum să ne adresăm acestuia. Entitățile LVM pot fi adresate în moduri multiple. Cea mai simplă metodă este să adresăm un logical volume precum: /dev/vgname/lvname. Astfel că, dacă am creat un logical volume cu numele lvdata, ce își ia spațiu dintrun volume group cu numele vgdata, calea entității va fi /dev/vgdata/lvdata.

Device mapper poartă un rol important în denumirea LVM logical volumes. Device mapper (abreviat dm) este o interfața generică folosită de kernelul Linux pentru a adresa device de stocare. Device mapper este folosit de către multiple sisteme: LVM volumes, dar și de software RAID etc. Aceste device sunt create în două locații: device numerotate în secvență plasate în directorul /dev, precum /dev/dm-0, /dev/dm-1 și așa mai departe. Deoarece aceste denumiri nu oferă nicio informație despre deviceuri ci confuzează, se folosesc scurtături create în directorul /dev/mapper. Acestea folosesc șablonul vgname-lvname pentru nume. Astfel că, dacă avem device /dev/vgdata/lvdata putem să ne adresăm folosind și /dev/mapper/vgdata-lvdata. Când lucrăm cum LVM logical volumes putem folosi oricare din aceste tipuri de adresare.

Arhitectura sistemului de fișiere

Din punctul de vedere al unui dezvoltator software am observat în timpul experienței faptul că pentru echipa efectivă de dezvoltare era destul de dificil de întretinut baza de date (mă refer chiar și la pornit/oprit, 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, ceea ce se factură diferit la client, prin urmare trebuia convins în prealabil Product Owner-ul de necesitatea acelei intervenții si planificat totul.

Astfel că, am observat necesitatea existenței unei aplicații care să ușureze operațiile asupra bazei de date, începând cu pornirea/oprirea/verificarea statusului până la întreaga operație de backup și point in time recovery(întoarcere la o stare existentă la un anumit moment dat). Aceasta aplicație trebuia să fie intuitivă, ușor de folosit chiar și pentru cineva care nu avea cunoștinte 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 încărcată și ar trebui să permită operațiile prezentate mai sus pentru fiecare instanță în parte.

Am propus o soluție pentru bazele de date PostgreSQL folosite pe platforme UNIX (la momentul de față). Pentru a prezenta soluția software am avut nevoie de un mediu UNIX pe care să ruleze un numar de instanțe PostgreSQL. Am ales o mașină CentOS, instanța EC2 oferită de Amazon. Sistemul de fișiere a fost organizat folosind LVM (logical volume manager) dupa cum urmeaza:

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

Au mai ramas 60GiB nealocați, aceștia putând fi folosiți ulterior. Dacă se observă că un client are nevoie de mai mult spațiu i se pot extinde logical volumes folosind

din spațiul nealocat, acesta fiind marele avantaj al LVM. Bazele de date nu vor fi întrerupte cât timp se face extinderea logical volumes.

Am setat astfel sistemul de fișiere pentru 3 instanțe diferite de baze de date PostgreSQL denumite lv-pgdata-1, lv-pgdata-2 si lv-pgdata-3. În *Figura 1* este prezentată diagrama sistemului de fișiere.

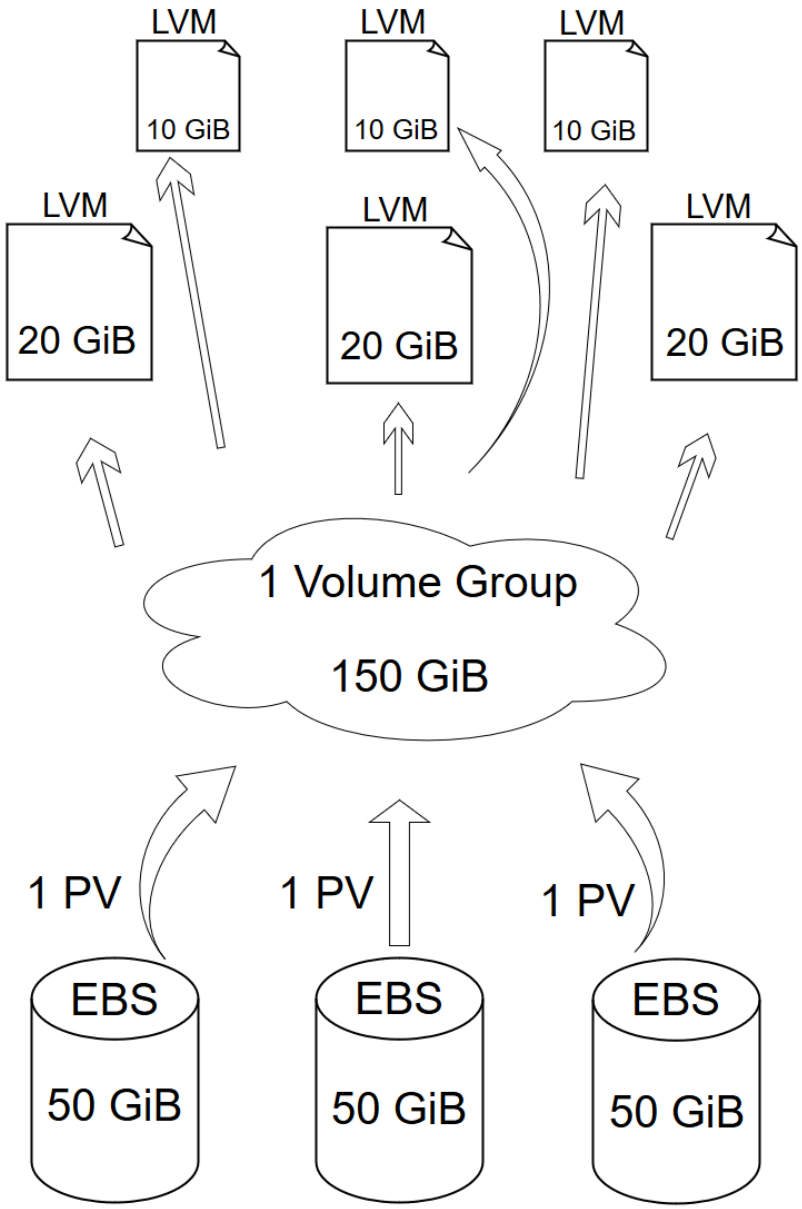

Figura 1: Diagrama sistemului de fisiere

DevOps as a Service

În cele ce urmează voi prezenta soluția software propusă pentru managementul bazelor de date în maniera descrisă anterior. Voi trece în revistă arhitectura software, urmând apoi să explic fiecare din componentele aplicației, precum și modul de interacționare a acestora.

4.1 Arhitectura sistemului software

Sistemul este împărtit în 3 componente importante:

- Scripturi UNIX ce interacționează cu instanțele de baze de date;
- RESTful API folosit pentru a expune funcționalitatea oferită de scripturi prin intermediul protocolului HTTP;
- Aplicație Web ce comunică prin intermediul protocolului HTTP cu API-ul RESTful expus.

În *Figura 2* este prezentată diagrama arhitecturii software.

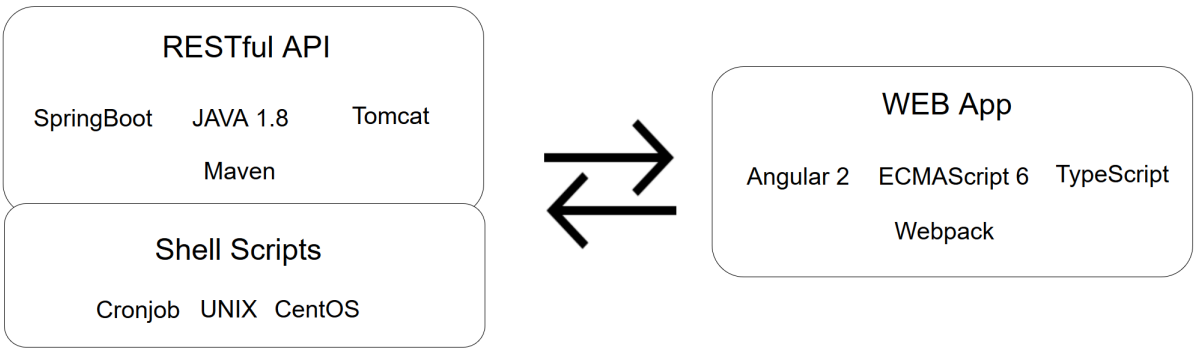

Figura 2: Diagrama arhitecturii software

4.2 Scripturile UNIX

Funcționalitatea de bază a aplicației constă în interacționarea cu instanțele de baze de date. Pentru asta am făcut 3 scripturi ce au urmatoarele responsabilități:

1. **pg-script.sh** pentru interacționarea cu instanțele (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 instanțe (checkpoint de la care să se poate începe 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 inițierea 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 gândite cu scopul de a se crea symbolic links (scurtături) către ele. Acestea vor avea numele instanței bazei de date, iar scriptul principal folosește numele fișierului scurtatură pentru a accesa datele instanței respective. Scopul este acela de a crea un singur set de scripturi, iar dacă avem n instanțe de baza de date doar creem symbolic links către scripturi, fiecare cu numele instanței de baza de date respective.

4.3 Server-ul: RESTful API

Pentru partea de server (backend) a fost aleasă soluția creării unui RESTful API cu scopul de a se expune funcționalitatea oferită de către scripturile bash prezentate anterior. Am utilizat Java Enterprise Edition, framework-ul Spring, ce facilitează crearea unui API prin intermediul librăriilor SpringWeb.

Un alt motiv pentru care am ales Spring a fost suportul oferit pentru a injecta dependințe urmând șablonul de proiectare Dependency Injection. Astfel, am decuplat controller-ul REST ce primește cererea web de către serviciul care o execută. Se poate injecta orice implementare de serviciu în controller, iar codul acestuia nu trebuie schimbat. Versiunea de Java utilizată 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 dependințelor și crearea artefactului s-a folosit Maven. Se oferă ca și exemplu fișierul **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 Interfața grafică cu utilizatorul

Scopul proiectului este crearea unei aplicații care să faciliteze manipularea serverelor de baze de date PostgreSQL de către dezvoltatori care nu trebuie să aibă acces la mașinile pe care se află bazele de date/ cunoștinte de UNIX. În acest sens, având deja server-ul de Java care expune prin servicii Web funcționalitățile respective, trebuie apelate și oferite într-o manieră cât mai intuitivă.

Ca și framework am folosit Angular 2, limbajul de programare fiind Javascript, standardul ECMAScript6, împreună 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 și manager de pachete am utilizat **npm** de la NodeJS.

4.5 Setarea aplicației în Amazon Web Services

Având ca și obiectiv crearea unui mediu cât mai apropiat de cel aflat pe mașinile din producție, am închiriat un server EC2 de la Amazon.

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

Odată ce sistemul de fișiere a fost configurat, am creat scripturile ce vor interacționa cu bazele de date. Urmatorul pas a fost pornirea celor două servere (backend și frontend). În timpul acestui proces a fost necesară configurarea unui rol de securitate oferit mașinii de calcul EC2, pentru a permite traficul pe portul 3001 utilizat de aplicația frontend.

Ghid de utilizare a aplicaţiei

Soluția propusă pentru a rezolva problema prezentată în capitolele procedente este o aplicație web folosită pentru managementul bazelor de date.

Aceasta oferă funcționalitățile de vizualizare a stării instanțelor de baze de date prin apăsarea butonului de "Refresh". Pentru a restaura starea bazei de date este necesară crearea unui checkpoint, iar pentru aceasta se apasă butonul "Create basebackup".

Odată existent un checkpoint, se poate restaura baza de date respectivă prin alegerea unei date/ore folosind componenta vizuală de tip calendar, urmând să se apese butonul "Revert".

Direcţii de dezvoltare viitoare

Aplicația propusă pentru a rezolva necesitatea restaurării bazelor de date la anumite momente în timp poate fi folosită doar pentru baza de date PostgreSQL.

Astfel că, o prima idee de dezvolatare viitoare ar fi investigarea suportului oferit alte tipuri de baze de date Oracle, MySql etc. urmând ca apoi să creez un design abstract în ceea ce privește baza de date și care să fie integrat cu soluția existentă oferită pentru PostgreSQL.

Cât despre interacțiunea cu utilizatorul, am folosit tehnologii actuale Angular 2, EcmaScript 6 etc. însa din punct de vedere al designului, ar putea fi îmbunătățit în ceea ce privește html și css existent.

Concluzii

Am creat o aplicație pornind de la necesitatea prezentată în capitolul *Introducere*. Scopul acesteia este de a oferi o metodă facila de a restaura o baza de date la un moment de timp oferit de către utilizator. Acesta nu necesită cunoștințe de baze de date, sau de devOps (nu e nevoie să interacționeze cu scripturi de UNIX).

Pentru a realiza cele menționate anterior am studiat modalitatea bazei de date PostgreSQL de management intern a transacțiilor și am utilizat-o întro manieră ce îmi permite restaurarea bazei de date la un moment în timp cu o granularitate fină, la secundă.

De asemenea, am studiat modalitatea sistemului de operare UNIX ce permite creare de partiții virtuale ce pot fi oricând modificate în ceea ce privește mărimea fără a închide baza de date. Am combinat aceasta funcționaliate a sistemului de operare împreuna cu soluția propusă, pentru a realiza cele prezentate cu un timp minim de inaccesibilitate a bazei de date. *Capitolul 3* prezinta în detaliu aceste aspecte.

Am propus o arhitectură a sistemului de fișiere bazat pe studiul prezentat anterior și am utilizat-o în implementarea propriu-zisa. Cât despre aceasta, am folosit serviciile Amazon Cloud (EC2, EBS) pentru a face deploy soluției (scripturi ce interacționează cu baza de date, aplicație backend și aplicație frontend).

In opinia mea, prin studiul realizat și prin intermediul soluției propuse am reușit să realizez ce mi-am propus și am oferit o modalitate de management a bazelor de date și de restaurare a acestora la starea existentă un moment de timp. Prin urmare am realizat o aplicație usor de utilizat de catre persoane ce nu au cunoștințe 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.