HO
GENT

Faculteit Departement IT en Digitale Innovatie

High Availability oplossingen voor PostgreSQL: een vergelijkende studie en proof of concept

Elias Ameye

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Thomas Aelbrecht
Co-promotor:
Ruben Demey

Instelling: —

Academiejaar: 2020-2021

Tweede examenperiode

Faculteit Departement IT en Digitale Innovatie

High Availability oplossingen voor PostgreSQL: een vergelijkende studie en proof of concept

Elias Ameye

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Thomas Aelbrecht
Co-promotor:
Ruben Demey

Instelling: —

Academiejaar: 2020-2021

Tweede examenperiode

Woord vooraf

Deze bachelorproef werd geschreven in het kader van het voltooien van de opleiding Toegepaste Informatica afstudeerrichting Systeem- en Netwerkbeheer. Ik heb gekozen voor dit onderwerp omdat High Availability clustering mij tijdens de opleiding altijd al interesseerde.

In dit onderzoek zal ik het hebben over High Availability oplossingen en het belang ervan in PostgreSQL clusters. Met de Coronapandemie die nog overal aanwezig is, is er een grote stijging in het digitale gebruik. Online winkelen heeft een enorme groei gekend. Koerierbedrijven hebben overuren moeten draaien om pakjes en post te brengen bij de mensen thuis. Technologie bedrijven kennen een extra druk omdat er meer beroep wordt gedaan op IT-services. Deze digitale (r)evolutie toont ons dat beschikbaarheid van diensten zeker heel relevant is. Stel je voor dat een server van Zalando, door software problemen, uitvalt. Alle aankopen van het laatste uur zijn niet doorgekomen. Een financieel drama. De oplossing? Een standby server die inspringt in geval van downtime. Resultaat? Geen downtime, geen financieel drama, geen geknoei met corrupte data. Met dit voorbeeld wil ik op een simpele manier het belang aantonen van High Availability in clusters. Stel, er loopt iets mis, kan het probleem snel opgelost worden, zonder dat de klant of het bedrijf er iets van nadelige ervaringen aan overhoudt.

Ik wil graag Thomas Aelbrecht, mijn promotor, bedanken voor de goede begeleiding en de duidelijke feedback. Ongeveer tweewekelijks kwamen we samen om eens te overlopen hoever ik zat. Hierdoor gaf ik mijzelf telkens een deadline tegen wanneer ik bepaalde zaken verricht wou hebben.

Ook wil ik Ruben Demey, co-promotor, bedanken om bij de vragen die ik had, duidelijke antwoorden te geven waardoor ik telkens een stap dichter was bij het einde.

Tot slot wil ik ook nog mijn vriendin bedanken voor de vele steun en toeverlaat.

Veel leesplezier toegewenst!

Samenvatting

Dit onderzoek kan dienen als basis voor de keuze tussen verschillende PostgreSQL High Availability cluster oplossingen. Doordat PostgreSQL een zeer populair open source object-relationele databank systeem is, en het geen tekenen geeft van in gebruikers te minderen, is het zeker interessant om te kijken hoe een bedrijf in 2020-2021 zijn High Availability clusters kan updaten, of creëren. In dit onderzoek ligt de focus vooral op High Availability in een PostgreSQL omgeving. Er wordt gekeken naar verschillende oplossingen zoals Patroni, PostgreSQL Automatic Failover (PAF), Pgpool-II en Replication Manager (repmgr). Deze oplossingen worden aan de hand van verschillende requirements met elkaar vergeleken. Hieruit brengt dit onderzoek één oplossing voor die gebruikt kan worden om High Availability te implementeren. In dit onderzoek vind u een inleiding tot High Availability in een PostgreSQL cluster met hieraan verbonden de huidige stand van zaken. Hierna worden vier oplossingen met elkaar vergeleken aan de hand van vooropgestelde requirements. Aan de hand van deze requirements krijgt elke oplossing een score toegewezen waarop bepaald zal worden met welke oplossing er verder gewerkt zal worden. De ultieme oplossing uit dit onderzoek blijkt Patroni te zijn die zeer goed aan de verschillende requirements voldoet. Tweede keus komt te liggen bij PostgreSQL Automatic Failover (PAF).

Nog af te werken.

Inhoudsopgave

1 Inleiding 15

1.1 Probleemstelling 15

1.2 Onderzoeksvraag 15

1.2.1 Hoofdonderzoeksvraag 15

1.2.2 Deelonderzoeksvraag 16

1.3 Onderzoeksdoelstelling 16

1.4 Opzet van deze bachelorproef 16

2 Stand van zaken 19

2.1 PosgreSQL 19

2.1.1 Databank 19

2.1.2 Relationele Databank 19

2.1.3 Object-georiënteerde databank 20

2.1.4 SQL 20
2.1.5 Object-relationele databank 20
2.1.6 PostgreSQL 21
2.2 High Availability 21
2.2.1 Load Balancing 23
2.2.2 Schaalbaarheid 23
2.3 Cluster oplossingen 24
2.3.1 Cluster 24
2.3.2 Failover 24
2.3.3 Failback 24
2.3.4 Patroni 24
2.3.5 Pgpool-II 25
2.3.6 PostgreSQL Automatic Failover (PAF) 27
2.3.7 RepMgr (Replication Manager) 27
2.3.8 PgCluster 29
2.3.9 Raima 29
2.3.10 EDB 29
2.4 Puppet 30
3 Methodologie 33
4 Schifting 35
4.1 Requirements 35
4.1.1 Functionele Requirements 35
4.1.2 Niet-functionele Requirements 36

4.2 Indelen requirements volgens MoSCoW-techniek 36

4.3 Hoe beantwoorden de verschillende oplossingen aan de bovenstaande requirements 37

4.3.1 Oplossing 1: Patroni 37

4.3.2 Oplossing 2: Pgpool-II 38

4.3.3 Oplossing 3: PostgreSQL Automatic Failover (PAF) 39

4.3.4 Oplossing 4: Replication Manager (RepMgr) 40

4.4 Resultatenanalyse 41

4.4.1 Resultaten requirements 41

5 Oplossing 1: Patroni 43

5.1 Redundantie/Replicatie 43

5.2 Failover 44

5.3 Monitoring 44

6 Oplossing 2: PostgreSQL Automatic Failover (PAF) 45

6.1 Redundantie/Replicatie 45

6.2 Failover 45

6.3 Monitoring 45

7 Conclusie 47

8 Proof of Concept 49

8.1 Omgeving 49

8.2 Prerequisites 49

8.2.1 Vagrant 49

8.2.2 VirtualBox 50

8.3 pgServer 51

8.4 pgNode1 52

8.5 pgNode2 55

A Onderzoeksvoorstel 59

A.1 Samenvatting 59

A.2 Introductie 60

A.3 State of the art 60

A.4 Methodologie 61

A.5 Verwachte resultaten 61

A.6 Verwachte conclusies 62

Bibliografie 63

Lijst van figuren

Lijst van tabellen

4.1 Requirementanalyse Patroni 41
4.2 Requirementanalyse Pgpool-II 41
4.3 Requirementanalyse PostgreSQL Automatic Failover (PAF) 42
4.4 Requirementanalyse Replication Manager (repmgr) 42
4.5 Requirementanalyse alle oplossingen 42

8.1 Ip-adressen Cluster 57

1. Inleiding

1.1 Probleemstelling

Deze vergelijkende studie kan een meerwaarde bieden aan bedrijven die werken met een kleine of grote PostgreSQL cluster waarin zij High Availability willen implementeren. Het kan ook een meerwaarde zijn voor bedrijven die al High Availability implementaties hebben in hun cluster, maar die een frisse blik nodig hebben, of willen upgraden naar een meer hedendaagse oplossing. Hiermee is dus vooral de focus gericht op systeem- en netwerkbeheerders die dagelijks bezig zijn met het onderhouden en/of beheren van servers, meer specifiek database servers.

1.2 Onderzoeksvraag

1.2.1 Hoofdonderzoeksvraag

Dit onderzoek zal zich bezighouden rond de vraag: Welke PostgreSQL High Availability cluster oplossing kunnen bedrijven, de dag van vandaag, gebruiken om garantie te hebben op monitoring, redundantie en failover?

In de conclusie (zie Hoofdstuk 7) van dit onderzoek zal worden antwoord gegeven op deze onderzoeksvraag.

1.2.2 Deelonderzoeksvraag

Naast de hoofdonderzoek vraag zijn er ook enkele ondersteunende deelonderzoeksvragen die bij dit onderzoek horen. Deze deelonderzoeksvragen zullen doorheen het onderzoek beantwoord worden. Bij de conclusie (zie Hoofdstuk 7) zal er een samenvatting van te vinden zijn.

· Wat zijn de voor- en nadelen van deze oplossing?
· Wanneer is High Availability nodig?

1.3 Onderzoeksdoelstelling

Het beoogde resultaat van deze bachelorproef is om uit de vergelijkende studie één oplossing voor te stellen die kan gebruikt worden voor implementatie van een High Available PostgreSQL cluster. Hieraan gekoppeld wordt ook een eerste poging tot een proof of concept getoond waarin deze oplossing geïmplementeerd zit. Het belangrijkste in dit onderzoek zal zijn wanneer er een PostgreSQL High Availability cluster oplossing is die voldoet aan de gevraagde requirements. Wanneer deze oplossing gevonden is, zal dit onderzoek geslaagd zijn.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie. Hierin zal ik mij vooral focussen op High Availability, PostgreSQL, Puppet, clustering en de reeds aanwezige oplossingen voor High Availability.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen. Hierin zal de schiftingsmethode die gebruikt zal worden om een oplossing te kiezen kort uitgelegd worden.

In Hoofdstuk 4 worden de functionele en niet-functionele requirements aan de hand van de MoSCoW-methode geprioriteerd om dan met elkaar vergelijken te worden. Hierbij zal elke oplossing punten krijgen. De twee oplossingen met meeste punten worden verder besproken.

In Hoofdstuk 5 en in Hoofdstuk 6 worden de twee verkozen oplossingen verder benaderd. Hier zal al wat specifieker worden ingegaan op hoe concreet aan de requirements kan voldaan worden.

Verder zal in Hoofdstuk 8 een proof of concept gemaakt worden met behulp van Patroni. Hierin zullen de verschillende stappen doorlopen worden en zijn code snippets in terug te vinden die gebruikt zijn geweest bij het opbouwen van de cluster.

In Hoofdstuk 7, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

Dit onderzoek is een vergelijkende studie tussen verschillende PostgreSQL oplossingen. In dit hoofdstuk zullen al verschillende oplossingen aan bod komen. Hiermee zal worden gekeken naar de nadruk die gelegd wordt bij elke oplossing. Op deze manier kan gekeken worden naar welke requirements regelmatig terugkomen, en dus welke kunnen gebruikt worden om later in het onderzoek verschillende oplossingen met elkaar te vergelijken. Aan de hand van deze requirements kan geschift worden tussen de bestaande High Availability PostgreSQL cluster oplossingen om hier dan twee oplossingen uit te halen.

2.1 PosgreSQL

2.1.1 Databank

Oracle (2021) omschrijft een databank als een georganiseerde verzameling van gestructureerde informatie, of gegevens, die meestal elektronisch in een computersysteem wordt opgeslagen. Een database wordt gewoonlijk beheerd door een databasebeheersysteem (DBMS). Samen worden de gegevens en het DBMS, samen met de toepassingen die ermee verbonden zijn, een databasesysteem genoemd, vaak afgekort tot gewoon database (Oracle, 2021).

2.1.2 Relationele Databank

Een relationele databank is een type databank waarin gebruik wordt gemaakt van een structuur die het mogelijk maakt om gegevens te identificeren en te benaderen in relatie tot

een ander deeltje data in diezelfde databank. Deze gegevens worden vaak georganiseerd in tabellen. Deze tabellen kunnen honderden, duizenden, miljoenen rijen en kolommen aan data hebben. Een kolom kent vaak ook een specifiek gegevenstype. Deze gegevenstypes kunnen getallen (integers), woorden (strings) of andere soorten bevatten (Codecademy, 2018).

2.1.3 Object-georiënteerde databank

Een objectgeoriënteerde database (OODBMS) is een type databank die zich baseert op objectgeoriënteerd programmeren (OOP). De gegevens worden hier voorgesteld en opgeslagen in de vorm van objecten. OODBMS worden ook objectdatabases of objectgeoriënteerde databasemanagementsystemen genoemd (CSharp-Corner, 2019).

2.1.4 SQL

SQL (structured query language) is de eigen programmeertaal specifiek ontwikkelt voor interactie met databanken. Een databank modelleert entiteiten uit het echte leven en slaat deze op in tabellen. Via SQL is het mogelijk om de gegevens in deze tabellen te manipuleren (Carchedi, 2020). SQL wordt door bijna alle relationele databanken gebruikt zoals MySQL, PostgreSQL, OracleDB, SQLite... (Codecademy, 2018).

2.1.5 Object-relationele databank

Een object-relationele database (ORD / ORDBMS) is een samenstelling uit zowel een relationele database (RDB / RDBMS), als een object-georiënteerde database (OOD / OODBMS). Samen ondersteunt het de basiscomponenten van elk objectgeoriënteerd databasemodel in zijn schema's en gebruikte querytaal, zoals klassen, overerving en objecten. Het bevat aspecten en kenmerken van bovenstaande genoemde modellen. Zo wordt het relationele duidelijk in de manier van opslaan van gegevens. Deze worden opgeslagen in een traditionele database en worden dan met behulp van SQL query's gemanipuleerd en benaderd. Aan de andere kant is ook het objectgeoriënteerde gedeelte merkbaar, namelijk dat de database beschouwd wordt als een objectopslag. Kort gezegd is één van de voornaamste doelstellingen van een object-relationele database, het dichten van de kloof tussen relationele en objectgeoriënteerde modelleringstechnieken en conceptuele datamodelleringstechnieken zoals daar zijn het entiteit-relatiediagram (ERD) en object-relationeel mappen (ORM) (Technopedia, 2021b). Klasses, Overerving, Types, Functies zijn kenmerken van de object-relationele database en zullen de basisconcepten vormen voor PostgreSQL. Een klasse is een verzameling van gegevenstypes die bij eenzelfde soort iets horen. Bijvoorbeeld een klasse CD kan als kenmerken hebben: Titel, zanger, Datum uitgave, aantal liedjes... . Overerving is wanneer een klasse bepaalde kenmerken overerft, krijgt van een superklasse. Bijvoorbeeld een klasse Olifant erft van de klasse Zoogdier kenmerken. Hierin is de klasse Olifant een specialisatie van de klasse Zoogdier. Een type is hierboven al eens genoemd geweest. Dit gaat over de verschillende soorten data die er zijn. Getallen, woorden, objecten zijn hier voorbeelden van. Functies

zijn een reeks SQL-statements die een specifieke taak uitvoeren. Functies bevorderen de herbruikbaarheid van code.

2.1.6 PostgreSQL

PostgreSQL is een open source systeem dat zich toelegt op het beheer van object-relationele databases. Het heeft meer dan 30 jaar actieve ontwikkeling en heeft een sterke reputatie op vlak van betrouwbaarheid, robuustheid van functies en prestaties (postgresql, 2021). PostgreSQL biedt een uitgebreide set van functionaliteiten die een hoge mate van customisatie mogelijk maakt binnen het systeem. Dit gaat van data administratie, beveiliging, tot backup en herstel . PostgreSQL wordt regelmatig bijgewerkt door de PostgreSQL Global Development Group en bijdragers uit de community. Deze community ondersteunt zichzelf en zijn gebruikers door het aanbieden van online educatieve bronnen en communicatiekanalen, zoals daar zijn PostgreSQL wiki, online forums en officiële documentatie. Er zijn ook bedrijven die commerciële support bieden aan een prijs (Nethosting, 2019). Volgens DB-Engines is PostgreSQL de vierde database die vandaag de dag het meest gebruikt wordt en de tweede meest gebruikte open source database, na MySQL (DB-Engines, 2021a). DB-Engines verklaarde PostgreSQL in 2017, 2018 en 2020 het DBMS (Database management system) van het jaar (DB-Engines, 2021b). PostgreSQL biedt veel mogelijkheden om ontwikkelaars te helpen bij het bouwen van applicaties; om beheerders te helpen bij het beschermen van data-integriteit; en bij het bouwen van fouttolerante omgevingen. Het helpt ook bij het beheren van data, hoe groot of hoe klein de dataset ook is. PostgreSQL voldoet sinds september 2020 aan 170 van de 179 verplichte functies voor SQL:2016 Core conformiteit. Schaalbaarheid valt ook toe te schrijven aan PostgreSQL, dit zowel in de hoeveelheid data die het kan beheren, als in het aantal gelijktijdige gebruikers dat het kan accomoderen. Er zijn actieve PostgreSQL clusters in productie omgevingen die terabytes aan data beheren, en gespecialiseerde systemen die zelfs petabytes beheren (PostgreSQL, 2021a). Enkele voorbeelden om te tonen hoe schaalbaar PostgreSQL wel is.

Postgres speelt in op de bovenvernoemde vier basisconcepten van een object-relationele databank zodat gebruikers het systeem makkelijk kunnen uitbreiden. Deze vier kenmerken, naast nog andere functies maken van Postgres een object-relationele database. Bovenstaande vermelde kenmerken zouden doen blijken dat Postgres voornamelijk een object-georiënteerde database is, maar de ondersteuning van de traditionele relationele databases, toont duidelijk aan dat, ondanks de object-georiënteerde kenmerken, Postgres stevig verankerd is in de relationele database wereld (PostgreSQL, 2021d).

2.2 High Availability

Het doel van High Availability architectuur is ervoor te zorgen dat een server, website of applicatie verschillende vraagbelastingen en verschillende soorten storingen kan verdragen. En dit met de minst mogelijke downtime. Door gebruik te maken van best practices die zijn ontworpen om hoge beschikbaarheid te garanderen, helpt dit volgens

Australia2017 om in een organisatie maximale productiviteit en betrouwbaarheid te bereiken (**Australia2017**).

High Availability is het vermogen van een systeem om continu operationeel te blijven te zijn gedurende een wenselijk lange tijd. Men kan Availability meten ten opzichte van 100% operationeel, als in, nooit uitvallen. Beschikbaarheid wordt vaak uitgedrukt als een percentage van uptime in een bepaald jaar op basis van de SLA's, Service Level Agreements (Kumar, 2020). Vaak duidt men deze norm aan als de Five 9's, namelijk 99,999% beschikbaarheid (Lutkevich, 2021). High availability impliceert dat delen van een systeem volledig zijn getest en dat er voorzieningen zijn voor storingen/failures in de vorm van redundante componenten. Servers kunnen worden ingesteld om in geval van nood de verantwoordelijkheden over te dragen aan een externe server, in een back-up proces. Hier spreekt men dan van failover (Technopedia, 2021a).

Wanneer is High Availability nu eigenlijk nodig? High Availability is nuttig wanneer bedrijven te maken hebben met dagelijk kritisch applicatiebeheer. Of wanneer er veel traffiek is op de website, is het niet te veroorloven om downtime te hebben. Een simpele reden als gewoon goede service aanbieden, is ook een geldige reden om aan High Availability te doen (CriticalCase, 2020).

Belangrijke principes van High Availability zijn:

1. **Het elimineren van single point of failure**: Toevoeging van redundantie zorgt er voor zodat het falen van een onderdeel in het systeem niet leidt tot het volledige falen van een geheel systeem.

2. **Betrouwbare cross-over**: In een redundant systeem wordt het kruispunt zelf een single point of failure. Fouttolerante systemen moeten voorzien in een betrouwbaar crossover- of automatisch omschakelingsmechanisme om storingen te voorkomen.

3. **Storingdetectie**: Als bovenstaande principes proactief bewaakt worden, dan zal een gebruik misschien nooit een systeemstoring zien. Postgres biedt de bouwstenen om bovenstaande principes volledig uit te werken zodat er op deze manier High Availability verzekerd kan worden.

Bij het elimineren van single points of failure ondersteunt Postgres de volgende fysieke stand-by's:

1. **Cold Standby**: Dit is een back-up server die beschikt over back-ups en alle nodige WAL-bestanden voor herstel. WAL is de afkorting voor Write Ahead Log. Het logt elke transactie die uitgevoerd wordt op een database voordat het wordt uitgevoerd. Een Cold Standby systeem is geen operationeel systeem, maar het kan wel beschikbaar worden gemaakt als dat nodig is. Voornamelijk worden dan backup servers en WAL bestanden gebruikt voor het maken van een nieuwe PostgreSQL node als onderdeel van disaster recovery.

2. **Warm Standby**: Hierin draait Postgres in herstelmodus en ontvangt updates door gebruik te maken van gearchiveerde logbestanden of door gebruik te maken van log shipping

replicatie van Postgres. Log shipping is een proces waarbij de back-up van transactielog-bestanden op een primaire database wordt geautomatiseerd en vervolgens op een standby server wordt hersteld (cawrites; Craig Guyer; Jason Roth, 2016). In deze modus aanvaardt Postgres geen verbindingen en queries.

3. **Hot Standby**: Ook bij Hot Standby draait Postgres in herstelmodus en ontvangt het updates door gebruik te maken van gearchiveerde logbestanden of door gebruik te maken van log shipping van Postgres. Het verschil met Warm Standby is dat in deze herstelmodus Postgres hier wel verbindingen ondersteunt en read-only queries.

Bovenstaande voorbeelden zijn mogelijkheden die kunnen helpen bij het elimineren van single points of failure. Afhankelijk van het overeengekomen niveau van beschikbaarheid, kunnen gebruikers voor een van de bovenstaande kiezen.

In geval van een volledige uitval van een systeem is geografische redundantie algemeen zeer wenselijk. Op deze manier worden servers verdeeld over meerdere locaties verdeeld over de wereld. Bij downtime door een natuurramp bijvoorbeeld zijn standby servers op meerdere fysieke (ongetroffen) locaties beschikbaar om in te vallen. Dit type van redundantie kan zeer duur uitdraaien, waarbij het een verstandige beslissing kan zijn om te kiezen voor een gehoste oplossing, waarbij de provider datacenters heeft over heel de wereld.

2.2.1 Load Balancing

Load Balancing is ook een manier om High Availability te waarborgen. Het doel van een load balancer is om toepassingen en/of netwerkverkeer te verdelen over meerdere servers en componenten. Het zal binnenkomende verzoeken routeren naar verschillende servers. Hiermeer wil het optionele prestaties en betrouwbaarheid verbeteren. Enkele voorbeelden van load balancing is Round Robin die ervoor zorgt dat de verzoeken van de load balancer naar de eerste server gaan in de rij. De verzoeken gaan deze rij af, tot hij op het einde komt, waarna hij terug van het eerste element in de rij begint. Een tweede manier van load balancing is Least Connection. Hierbij zal er gekozen worden om gebruik te maken van de server met het minst aantal actieve verbindingen. Load balancers spelen een rol bij het tot stand brengen van een infrastructuur met High Availability, maar het hebben van een load balancer staat niet garantie voor het hebben van High Availability. Door redundantie te implementeren voor de load balancer zelf, kan deze geelimineerd worden als een single point of failure (Jevtic, 2018))

2.2.2 Schaalbaarheid

Schaalbaarheid is de eigenschap van een systeem om te kunnen voldoen aan een groeiend aantal eisen door bronnen toe te voegen. De redenen voor een groei kunnen tijdelijk zijn, bijvoorbeeld wanneer een bedrijf net een nieuw product op de markt brengt, of wanneer er een substantiële groei is in het aantal klanten of personeel. Er zijn twee manieren om aan schalen te doen. Horizontaal en verticaal (Insausti, 2019).

Horizontaal schalen

Horizontaal schalen richt zich op het toevoegen van nodes in een systeem of cluster. Hierin is het handig om meer slave nodes toe te voegen aan de cluster. Dit zal helpen om lees- en schrijfprestaties te verbeteren, doordat het meer gebalanceerd kan gebeuren. Hierbij is een load balancer nodig, die dit verkeer zal regelen. Eén load balancer zal niet voldoende zijn om single point of failure preventief te voorkomen. Beter hier is dan twee of meerdere load balancers toevoegen. Op deze manier blijft High Availability een garantie (Insausti, 2019).

Verticaal schalen

Verticaal schalen is het toevoegen van hardware resources aan bestaande nodes. Deze resources kunnen gaan over RAM-geheugen, schijfgeheugen, CPU (Insausti, 2019).

2.3 Cluster oplossingen

2.3.1 Cluster

Een cluster is een groepering van servers die met elkaar samenwerken om één geheel te vormen. Op deze manier kan een cluster High Availability mogelijk maken (TechTarget, 2017).

2.3.2 Failover

Failover is het automatisch overschakelen naar een back-upsysteem. Wanneer een primair systeemonderdeel faalt, wordt failover ingeschakeld om de negatieve gevolgen te elimineren of te beperken (AVINetworks, 2020).

2.3.3 Failback

Failback is het proces van het herstellen van operaties naar een primaire machine nadat ze zijn verschoven geweest naar een secundaire machine wegens failover (TechTarget, 2020).

2.3.4 Patroni

Patroni is een open source cluster-technologie, geschreven in Python die zich bezighoudt met automatische failover en High Availability voor een PostgreSQL databank. Het dient als een soort cluster manager die de implementatie en het onderhoud van High Availability in PostgreSQl clusters zal aanpassen en automatiseren. Het maakt gebruik van

gedistribueerde configuratieopslagplaatsen zoals etcd, Consul, ZooKeeper of Kubernetes voor maximale toegankelijkheid (Markwort, 2018).

Patroni biedt cloud-native netwerkfuncties en geavanceerde opties voor failback en failover. Een cloud-native netwerkfunctie is een software-implementatie van een netwerkfunctie, die wordt uitgevoerd in een linux-container, die traditioneel wordt uitgevoerd door een fysiek apparaat (CDNF, 2020).

Als oplossing zorgt Patroni voor een end-to-end setup van High Available PostgreSQL clusters, inclusief streaming replicatie. Streaming replicatie biedt de mogelijkheid om continu WAL logs naar standby servers te sturen en deze toe te passen om de servers op deze manier up to date te houden (wiki.postgresql.org, 2020). Het biedt verschillende manieren waarop een standby node kan aangemaakt worden, en dient als sjabloon dat kan worden aangepast naar wat gewenst word.

Bij het aanmaken van een standby node maakt Patroni gebruik van pg_basebackup en ondersteunt het ook methodes zoals Barman, pgBackRest en meer die gebruikt worden voor het aanmaken van standby nodes.

Na het opzetten van de cluster, zal Patroni zich actief bezighouden met het monitoren. Patroni zal werken aan de hand van een leader lock. Deze wordt om de zoveel tijd vernieuwt, en wanneer de master node er niet in slaagt om deze te vernieuwen, zal Patroni een nieuwe node verkiezen tot master node (ScaleGrid, 2018a).

2.3.5 Pgpool-II

Pgpool-II omschrijft zichzelf als een middleware die werkt tussen PostgreSQL servers en een PostgreSQL database client. Het ondersteunt High Availability, geautomatiseerde load balancing, etc. Pgpool-II biedt ook logical replication. Logische replicatie, of logical replication stelt gebruikers in staat om een selectieve replica van bepaalde tabellen uit te voeren en deze uit te schrijven naar een standby node. Met logical replication kan een standby node replicatie ingeschakeld hebben van meerdere master nodes. Dit kan handig zijn in situaties waar je gegevens van verschillende PostgreSQL databases moet repliceren naar een enkele PostgreSQL server voor rapportage en data warehousing. Eén van de grootste voordelen van logische replication boven streaming replication is dat logical toelaat om veranderingen van een oudere versie van PostgreSQL naar een nieuwere versie te repliceren. Streaming replicatie werkt alleen als zowel de master als de standby van dezelfde versie zijn. Wat niet altijd ideaal is in grote opstellingen (Ihalainen, 2019). Pgpool-II biedt de volgende features:

1. Connection Pooling

Pgpool-II bewaart verbindingen naar de PostgreSQL servers, en hergebruikt ze wanneer een nieuwe verbinding met dezelfde eigenschappen zoals gebruikersnaam, database of protocol versie binnenkomt. Dit vermindert de overhead van verbindingen, en verbetert de totale doorvoer van het systeem.

2. Replication

Pgpool-II kan meerdere PostgreSQL servers beheren. Door gebruik te maken van de replicatiefunctie kan een realtime backup worden gemaakt op 2 of meerdere fysieke schijven, zodat de dienst kan worden voortgezet zonder servers te stoppen in geval van een schijfstoring.

3. Load Balancing

Als een database wordt gerepliceerd, zal het uitvoeren van een query op elke server hetzelfde resultaat opleveren. Pgpool-II maakt gebruik van de replicatie mogelijkheid om de belasting op elke PostgreSQL server te verminderen door queries over meerdere servers te verdelen, waardoor de totale throughput van het systeem verbetert. In het meest gunstige geval verbetert de prestatie evenredig met het aantal PostgreSQL servers. Load balance werkt het beste in een situatie waarin er veel gebruikers zijn die veel queries tegelijkertijd uitvoeren.

4. Limiting Exceeding Connections

Er is een limiet op het maximum aantal gelijktijdige verbindingen met PostgreSQL, en verbindingen worden geweigerd na dit maximaum aantal verbindingen. Het instellen van een max aantal verbindingen verhoogt echter het verbruik van bronnen en beïnvloedt de systeemprestaties. pgpool-II heeft ook een limiet op het maximum aantal verbindingen, maar extra verbindingen worden dan in een wachtrij geplaatst..

5. Watchdog

Watchdog kan een robuust clustersysteem creëren en het single point of failure of split brain vermijden. Watchdog kan een lifecheck uitvoeren tegen andere Pgpool-II nodes, om een fout van Pgpool-II te detecteren. Als actieve Pgpool-II down gaat, kan dan een standby Pgpool-II gepromoveerd worden tot actief, en zal deze dan virtueel het IP overnemen.

6. In Memory Query Cache

In memory query cache maakt het mogelijk om een paar SELECT statements en zijn resultaat op te slaan. Als een identieke SELECT binnenkomt, retourneert Pgpool-II de waarde uit de cache. Omdat er geen SQL parsing of toegang tot PostgreSQL aan te pas komt, is het gebruik van in memory cache extreem snel. Aan de andere kant kan het in sommige gevallen langzamer zijn dan het normale pad, omdat het wat overhead toevoegt van het opslaan van cache gegevens.

Pgpool-II praat met de backend en de frontend protocollen van PostgreSQL, en legt een verbinding tussen beide. Daarom denkt een database applicatie (frontend) dat Pgpool-II de eigenlijke PostgreSQL server is, en de server (backend) ziet Pgpool-II als een van zijn clients. Omdat Pgpool-II transparant is voor zowel de server als de client, kan een bestaande databasetoepassing met Pgpool-II worden gebruikt vrijwel zonder de broncode aan te passen (pgpool, 2021).

2.3.6 PostgreSQL Automatic Failover (PAF)

PostgreSQL Automatic Failover (PAF) is een nieuwe resource agent, speciaal voor PostgreSQL. Door Pacemaker en Corosync, is het voor PAF mogelijk om:

1. Aan detectiestoring te doen van een PostgreSQL instance.

2. de primary server te herstellen...

3. of om aan failover te doen naar een standby server.

4. Om de best (met de kleinste vertraging) beschikbaarste standby server te selecteren bij failover.

5. Rollen te wisselen in de cluster tussen standby en primary nodes.

De oorspronkelijke wens is om een duidelijke grens te houden tussen de Pacemaker administratie en de PostgreSQL administratie, om dingen eenvoudig, gedocumenteerd en toch krachtig te houden.

Zodra een PostgreSQL cluster is opgebouwd met behulp van interne streaming replicatie, is PAF in staat om aan Pacemaker te laten zien wat de huidige status is van de PostgreSQL instantie op elke node: primary, standby, gestopt, etc. Mocht er dan een storing optreden op de primary node, dan zal Pacemaker standaard proberen deze te herstellen. Mocht de storing niet te herstellen zijn, dan zorgt PAF ervoor dat de standby servers de beste van hen kunnen kiezen (de dichtstbijzijnde bij de oude primary) en deze promoveren tot de nieuwe primary. Dit doet PAF dankzij Pacemaker (de Rorthais, 2020).

Postgres Automatic Failover (PAF) biedt verschillende voordelen in het omgaan met PostgreSQL High Availability. PAF gebruikt IP adres failover in plaats van het herstarten van de standby om verbinding te maken met de nieuwe master tijdens een failover event. Dit blijkt voordelig te zijn in scenario's waarbij de gebruiker de standby nodes niet opnieuw wil opstarten. PAF heeft ook zeer weinig manuele tussenkomst nodig en beheert de algemene gezondheid van alle Postgres databankbronnen. Het enige geval waarin manuele tussenkomst een vereiste is, is dan in het geval van een tijdlijn data divergentie waarbij de gebruiker kan kiezen om pg_rewind te gebruiken (ScaleGrid, 2018b).Pg_rewind is een tool die helpt bij het synchroniseren van een PostgreSQL cluster met een andere kopie van diezelfde cluster, met als enige verschil de tijd. Voorbeeld hierbij is een oude master node terug online brengen na failover als een standby node die de nieuwe master node volgt (PostgreSQL, 2021c).

2.3.7 RepMgr (Replication Manager)

Repmgr is een open-source oplossing die zich bezighoudt met het beheren van replicatie en failover van servers in PostgreSQl clusters. Het verbetert de ingebouwde hot-standby opties van PostgreSQL met extra features zoals tools om standby servers op te zetten, replicatie te monitoren en administratieve taken uit te voeren, zoals failover (repmgr,

2021b).

De features die repmgr 5 aanbiedt zijn:

1. De implementatie als een PostgreSQL extentie.

2. Replicatie cluster monitoring.

3. Standby klonen aan de hand van pg_basebackup of Barman

pg_basebackup: pg_basebackup wordt gebruikt om basisbackups te maken van een draaiende PostgreSQL databank cluster. Deze back-ups worden gemaakt zonder de aanwezige cliënts, die in verbinding staan met de databank, te beïnvloeden. Deze back-ups kunnen gebruikt worden voor zowel point-in-time recovery, maar ook als een startpunt voor log shipping of streaming replicatie standby servers. Bij point-in-time recovery wordt verwezen naar herstel van dataveranderingen tot een bepaald punt in de tijd (MySQL, 2021).

pg_basebackup maakt een binaire kopie van de database cluster bestanden, terwijl het ervoor zorgt dat het systeem automatisch in en uit backup modus wordt gezet. Backups worden altijd gemaakt van de gehele databasecluster; het is niet mogelijk om een backup te maken van afzonderlijke databases of databaseobjecten.

De backup wordt gemaakt over een gewone PostgreSQL verbinding maakt gebruik van het replicatieprotocol. De server moet ook worden geconfigureerd om ten minste één sessie beschikbaar te laten voor de backup (PostgreSQL, 2021b).

Barman: Barman of pgbarman staat voor Backup en Recovery Manager. Het is een open-source beheertool voor disaster recovery van PostgreSQL servers. Het is geschreven in Python. Barman laat toe om van op afstand van meerdere servers in bedrijfskritische omgevingen back-ups uit te voeren. Het helpt ook database beheerders tijdens een herstelfase (Barman, 2020b) (Barman, 2020a).

4. Standby server die kan worden gepromoveerd tot een primary server zonder herstart. Andere standby servers die verbinding kunnen maken met de nieuwe master zonder opnieuw gesynchroniseerd te worden (wiki.postgresql, 2020).

5.Cascading Standby Support

Standby servers die niet direct verbonden zijn met de master node worden niet beïnvloed tijdens failover van de primary naar een andere standby mode.

6. Vereenvoudigen van het beheer van WAL-retentie en ondersteuning voor replicatiesleuven

Door de vereenvoudiging van WAL-retentie zal het dus eenvoudiger zijn om WAL-bestanden op te ruimen vanaf elke bestandssysteemlocatie. Ook in standby servers kan het gebruikt worden om bestanden die niet meer nodig zijn, te verwijderen uit de standby server (Augustine, 2019).

7. Switchover ondersteuning voor rolswitching tussen primary en standby

Hierin wordt een standby server geprovomeerd tot een primary server en zal deze primary server degraderen naar een standby server. Wanneer andere standby servers verbonden zijn met de degradatiekandidaat, kan repmgr deze instrueren om de nieuwe primary server te volgen en niet de oude, die net gedegradeerd is (repmgr, 2021a).

2.3.8 PgCluster

2.3.9 Raima

Databasereplicatie is het proces waarin gegevens worden gekopieerd van een database naar één of meerdere replica's. Dit om de toegankelijkheid van gegevens en fouttolerantie te verbeteren. In de context van replicatie gebruikt men vaak ook de termen actief-actief en actief-passief. Raima Database Manager (RDM) ondersteunt beide technieken. Bij Raima wordt actieve replicatie gewoon replicatie genoemd en passieve replicatie spiegelen (mirroring). Spiegelen zal resulteren in identieke replica's zoals de originele database, terwijl replicatie zal resulteren in replica's die niet identiek zijn aan de originele database. Deze replica's zullen alle records bevatten die van de originele database zijn overgebracht, maar de fysieke organisatie van de records in de databasebestanden (of in het geheugen) kan verschillen. Om terug te komen op de termen actief-actief en actief-passief zullen die vaker verwijzen naar andere concepten dan dewelke juist omschreven (replicatie en spiegelen). Actieve-actieve replicatie betekent replicatie in twee richtingen van gegevens tussen twee databases die beide actief worden bijgewerkt. Actieve-passieve replicatie betekent replicatie in één richting van een actief bijgewerkte master node naar een slave node die niet wordt bijgewerkt, behalve door het replicatieproces. Hier verwijst men soms ook naar master-slave replicatie. In RDM is replicatie altijd actief-passief (Raima, 2021).

2.3.10 EDB

Voor betrouwbare cross-over biedt EDB een technologie genaamd EDB Postgres Failover Manager (EFM). Dit maakt automatische failover van de Postgres master node naar een standby node mogelijk in geval van een software- of hardwarefout op de master. EFM maakt gebruik van JGroups, die een betrouwbare, gedistribueerde en redundante infrastructuur biedt zonder een single point of failure. EDB Postgres Failover Manager kan ook gebruikt worden voor de detectie van storingen. Het bewaakt de server continu en zal storingen op verschillende niveaus detecteren. Het is ook capabel om om failover uit te voeren van de master node naar één van de replica nodes om het systeem beschikbaar te maken voor het accepteren van databaseverbindingen en queries. Wanneer EFM goed geconfigureerd is, kan het storingen detecteren en direct failover uitvoeren.

Verlies van service kan in twee categoriëen opdelen. Geplande uitval of downtime en ongeplande uitval of downtime. Geplande downtime is vaak het gevolg van onderhoudsactiviteiten. Dit kan zijn door een softwarepatches die een herstart van het systeem of van de database vereist. In het algemeen is deze uitval niet onverwachts en zal deze uitval geen

grootschalige gevolgen hebben. Een ongeplande downtime is vaak het resultaat van een of andere fysieke gebeurtenis, zoals hardware- of softwarestoring, of een anomalie in de omgeving. Stroomuitval, defecte CPU- of RAM-componenten (of eventueel andere hardwarecomponenten), netwerkstoringen, inbreuken op de beveiliging, of diverse defecten in toepassingen, middleware en besturingssystemen resulteren bijvoorbeeld in ongeplande uitval. In geval van (on)geplande downtime kan EFM helpen om de downtime zoveel mogelijk te minimaliseren. Voor een geplande downtime kan een gebruiker bijvoorbeeld eerst alle standby nodes patchen en EFM gebruiken om over te schakelen voordat de master node gepatcht wordt. Bij een ongeplande downtime kan EFM ervoor zorgen dat de storingen gedetecteerd worden en failover uitvoeren naar de juiste standby node, om dan deze node de nieuwe master node te maken. EFM zal na dit proces er ook voor zorgen dat de oude master node niet terugkomt om een split-brain situatie te voorkomen. Split-brain duidt op de inconsistenties in beschikbaarheid en data. Hierdoor ontstaan er twee afzonderlijke datasets met overlap (Kumar, 2020).

2.4 Puppet

Puppet CTO Deepak Giridharagopal zei dat in het kielzog van de economische neergang als gevolg van de COVID-19 pandemie, meer IT-teams zwaar zullen moeten vertrouwen op automatisering. De meeste IT-teams zullen ofwel even groot blijven of worden ingekrompen. De IT-omgeving zal echter steeds complexer worden. De enige manier om IT-teams in staat te stellen meer te doen met minder is het automatiseren van meer routinetaken (Vizard, 2020).

Puppet is een cross-platform client-server gebaseerde toepassing die wordt gebruikt voor configuratiebeheer. Het behandelt de software en zijn configuraties op meerdere servers. Er zijn hierbij twee versies beschikbaar. De ene is open-source, de andere is een betalende, commerciële versie. Het werkt op zowel Linux als op Windows. Het gebruikt een declaratieve aanpak om updates, installaties en andere taken te automatiseren. De software kan systemen configureren met behulp van bestanden die manifesten worden genoemd. Een manifest bevat instructies voor een groep of type server(s) die wordt/worden beheerd (Kelly, 2020).

Wat is configuratiebeheer nu juist? Configuratiebeheer onderhoudt en bepaalt productkenmerken door fysieke en functionele attributen, ontwerp, vereisten en operationele informatie op te slaan gedurende de levenscyclus van een server.

Puppet maakt gebruik van de beschrijvende programmeertaal Ruby. Ruby is een dynamische, open source programmeertaal met de nadruk op eenvoud en productiviteit (Ruby, g.d.).

Vroeger werden software en systemen door systeembeheerders manueel opgezet en geconfigureerd. Maar toen het te beheren aantal servers snel toenam, moest er gezocht worden naar een manier om die processen te automatiseren, om dan zo tijd te besparen en de nauwkeurigheid te vergroten. Puppet is uit deze zoektocht ontstaan.

Puppet werkt aan de hand van een eenvoudig client/server architectuur workflow proces. Hierin bestaat er een master server die alle informatie bevat over de configuraties van de verschillende nodes aanwezig. Het slaat deze configuraties op in manifestbestanden op een een centrale server, genaamd de Puppet master, en voert deze manifesten uit op de remote client servers genaamd agents (Kelly, 2020).

3. Methodologie

Zoals in vele andere onderzoeken ook het geval is, is dit onderzoek gestart met een diepgaande en extensieve literatuurstudie over PostgreSQL, High Availability, Puppet en de reeds bestaande High Available PostgreSQL cluster oplossingen. Deze literatuurstudie is terug te vinden in Hoofdstuk 2: Stand van zaken.

Na de literatuurstudie zal worden geduid hoe de verschillende High Available PostgreSQL cluster oplossingen geschift zullen worden. In deze schifting zullen verschillende requirements opgezet worden waaraan de verschillende oplossingen zullen afgetoetst worden. Dit zal gebeuren aan de hand van een requirementsanalyse waarin de verschillende requirements aan bod zullen komen. Deze analyse zal van groot belang zijn in dit onderzoek aangezien ze de basis zullen vormen waarop we een High Available PostgreSQL cluster oplossing zullen beoordelen en kiezen. Uit deze schifting, aan de hand van de requirements worden dan de beste twee kandidaten verkozen.

Na de schifting en de verdieping van de twee High Available PostgreSQL cluster oplossingen zal er in dit onderzoek één oplossing gebruikt worden waar een proof of concept mee gemaakt zal worden. Hier zal dan, door gebruik te maken van Puppet de reproduceerbaarheid van de High Available PostgreSQL cluster zeer eenvoudig en snel moeten verlopen.

4. Schifting

4.1 Requirements

Om te kunnen bepalen welke requirements er nodig zijn,

Elk van deze High Available PostgreSQL cluster oplossingen zal worden afgetoetst aan de requirements om ze op deze manier te evalueren. In samenspraak met Ruben Demey zijn er drie functionele requirements. Bij elke oplossing worden de requirements afgetoetst en krijgen ze een score van 1 (slecht) tot 5 (uitstekend).

4.1.1 Functionele Requirements

Een functionele requirement beschrijft hoe het systeem moet werken en wat het moet kunnen.

Redundancy

Om High Availability te bereiken moeten clusters aan bepaalde eisen voldoen. Het moet over redundantie beschikken om single points of failure te vermijden.

Failover

Het opzetten van failover (replicatie) biedt de nodige redundantie om High Availability mogelijk te maken door ervoor te zorgen dat standby nodes beschikbaar zijn als de master of primary node ooit uitvalt.

Monitoring

Monitoring is belangrijk omdat het op deze manier actief storingen kan opmerken en opsporen. Monitoring checkt de algemene gezondheid en beschikbaarheid van een systeem.

4.1.2 Niet-functionele Requirements

Een niet-functionele requirement is een kwaliteitseis voor het systeem. Dit gaat dan meer over voorkeuren.

Open source

Open source verwijst naar source code die publiekelijk toegankelijk is en die door iedereen gebruikt mag worden zonder nodige licentie (opensource.com, 2021).

Anno 2020-2021

Een belangrijke requirement is dat de oplossing up-to-date moet zijn. In dit onderzoek zal er geen gebruik gemaakt worden van achterhaalde, oude tools. Het is belangrijk dat de tool futureproof kan zijn en zeker nog jaren kan meegaan.

Grafische interface

Een gemakkelijk te gebruiken interface is zeker een meerwaarde bij het monitoren en beheren van een cluster. Als er geen specialist aanwezig is, kan de grafische interface soms voor wat extra duidelijkheid zorgen.

4.2 Indelen requirements volgens MoSCoW-techniek

Na het opstellen van de verschillende requirements kunnen we deze nog eens indelen volgens de MoSCoW-techniek (Ahmad e.a., 2017). Hierin worden de requirements geprioritiseerd in 4 categorieën. Must have, Should have, Could have, Won't have Hierin zijn de Must have-requirements verplicht aanwezig. De Should have-requirements zijn geen verplichting, maar zijn het liefst wel aanwezig in de keuze van een oplossing. De Could have-requirements zijn volledig optioneel en zijn dus niet geheel relevant bij het kiezen van de oplossing. Het Won't have aspect laten we hier achterwege, omdat dit geen invloed zal hebben op de keuze van een oplossing.

- **Must have**
  - Ondersteuning van redundancy
  - Ondersteuning van failover
  - Ondersteuning van monitoring
- **Should have**

- Ondersteuning in 2020-2021
- Open source

• **Could have**

- Grafische interface
- Zo weinig mogelijk manuele interventie

4.3 Hoe beantwoorden de verschillende oplossingen aan de bovenstaande requirements

In dit gedeelte worden de verschillende oplossingen vergeleken aan de hand van de voor-opgestelde requirements. Wanneer een oplossing een zeer brede implementatie heeft van een requirement zal deze aan de hand van een puntensysteem een 5/5 krijgen. Wanneer er niet voldoende implementatie aanwezig is voor deze oplossing, zal dit een 1/5 zijn. De functionele requirements zullen zwaarder doorwegen dan de niet-functionele requirements. Hierbij zal het totaal aantal punten op 20 staan, waarbij de functionele requirements 75% van de totaalscore bevatten. Dit staat gelijk aan 15/20. De niet-functionele requirements staan dus maar op 25%, wat gelijk staat aan 5/20 van de totaalscore. Hierbij kan er een goed beeld gevormd worden welke oplossing er functioneler is. Belangrijke niet-functionele requirement is dat de oplossing up-to-date moet zijn, anno 2020-2021. Als de oplossing hier niet aan voldoet, dan zal deze niet doorkomen, of gebruikt worden als proof of concept.

4.3.1 Oplossing 1: Patroni

Must have

Voor replicatie opties bestaan er verschillende tools zoals barman, Wal-E, die zullen helpen om nodes toe te voegen aan de cluster. Er kan ook gekozen worden van waar sommige nodes hun data zullen halen. Patroni Patroni gebruikt PostgreSQL streaming replicatie en ondersteunt zowel synchrone als asynchrone replicatie. Standaard wordt asynchrone replicatie geconfigureerd, maar dit kan gewijzigd worden naar de noden van de cluster.

Patroni heeft een automatische failover-functie. Hierbij wordt automatisch gekeken welke node de te vervangen node zal vervangen. Na failover zal ook automatisch de gefaalde node terug in de cluster komen.

Patroni heeft een zeer uitgebreide REST API, die gebruikt kan worden om failover, switchover, reloads, restarts, health checks, etc. uit te voeren. Patroni heeft een tool genaamd, ETCD waarmee de gezondheid van een PostgreSQL cluster, de status van knooppunten en andere informatie van de cluster wordt bijgehouden. Tools die hiervoor ook gebruikt worden en semi-gelijkaardige functionaliteiten hebben zijn o.a. Consul en Zookeeper. Er zijn ook tools die uitsluitend als doel monitoring hebben. Hiervan is PGWatch een goed voorbeeld van. PGWatch geeft op een visuele manier informatie weer over de Postgres nodes.

Patroni bevat veel antwoorden op de verschillende 'Must haves'. Redundantie/replicatie kan hier vele vormen aannemen, aangepast aan de noden van de cluster. Failover kan op een automatische manier gebeuren en hoeft niet manueel gedaan te worden. Ook voor monitoring zijn er verschillende tools beschikbaar. Hierbij zijn ETCD, Consul en Zookeeper goed omschreven voorbeelden. Op basis hiervan krijgt Patroni 12/15 (3 x 4) punten.

Should have

Patroni is ontwikkeld door Zalando en is volledig open source. De volledige source code is online te vinden op github.com en wordt nog steeds geüpdate. Dit is ook een belangrijke vereiste waaraan de oplossing moet doen. Er worden nog steeds releases gedaan. De laatste release was, tijdens dit schrijven, op 22 februari 2021. Deze release bevatte sommige nieuwe features zoals toegevoegde support voor de REST API bij TLS keys en cipher suite limitations, maar ook door stabiliteitsverbetering en bugfixes.

Patroni voldoet door zijn open source en recente relevante updates aan de Should have requirements. Hierdoor krijgt Patroni 5/5 punten bij Should have.

Could have

Patroni kent ook grafische interfaces die gebruikt kunnen worden bij het opzetten van een Patroni PostgreSQL High Availability cluster. Een voorbeeld hiervan is Patroni Environment Setup (PES). Hiermee kan je op Windows eenvoudig, snel en gebruikersvriendelijk een Patroni cluster opzetten.

Patroni kan opgesteld worden op een manier waarbij niet veel manuele interventie nodig is.

4.3.2 Oplossing 2: Pgpool-II

Must have

Pgpool-II beschikt over een tool, genaamd Watchdog, die een subproces is van Pgpool-II dat dient om High Availability toe te voegen. Watchdog wordt gebruikt om single points of failure op te lossen door meerdere Pgpool-II nodes te coördineren. Het coördineert meerdere Pgpool-II nodes door informatie met elkaar uit te wisselen. Op die manier kan het zeker zijn dat de database service onaangetast blijft.

Watchdog kan ook remote de gezondheid controleren van de node waarop het is geïnstalleerd door de verbinding met upstream nodes te monitoren. Als de monitoring faalt, behandelt watchdog dit als het falen van een lokale Pgpool-II node en zal het de nodige acties ondernemen.

Pgpool-II kan meerdere nodes beheren en hierin replicatie opzetten om High Availability te verkrijgen.

Pgpool-II voorziet ook automatische failover binnenin Watchdog. Pgpool-II kan ook andere soorten van failover configuraties aan zoals failover_requires_consensus of failover_when_quorum_exi

Pgpool-II kan aan de hand van Watchdog aan de vooropgestelde Must haves voldoen. Pgpool-II krijgt hierdoor 10/15 punten.

Should have

Pgpool-II is een open source project. De source code wordt onderhouden door hun git-repository. De laatste release van pgpool-II was op 26 november 2020. Hieruit kunnen we ook zeggen dat Pgpool-II nog een geldige, bruikbare oplossing.

Aan de hand hiervan krijgt Pgpool-II 4/5 punten voor Should have. Geen 5/5 omdat er (nog) geen update in 2021 is geweest.

Could have

Op eerste zicht is er niet direct een alomgekende grafische interface aanwezig voor Pgpool-II.

Via de tool Watchdog is het mogelijk om bepaalde elementen in Pgpool-II te automatiseren. Automatische failover is hier een voorbeeld van. Doch andere elementen zijn wel nog manueel te configureren.

4.3.3 Oplossing 3: PostgreSQL Automatic Failover (PAF)

Must have

PostgreSQL Automatic Failover (PAF) werkt nauw samen met de tool Pacemaker. PAF is in staat om aan Pacemaker te laten zien wat de huidige status is van een node. Bij het optreden van een storing zal Pacemaker automatisch proberen dit te herstellen. Als de storing niet te herstellen valt, dan zal PAF zorgen voor automatische failover. PAF maakt gebruik van ip-adres failover in plaats van het herstarten van de standby node om verbinding te maken met de nieuwe master tijdens een failover event, wat voordelig is in scenario's waar een gebruiker de standby nodes niet wil herstarten.

PostgreSQL Automatic Failover (PAF) maakt gebruik van synchrone replicatie om te garanderen dat er geen data verloren gaat tijdens een failover.

PostgreSQL Automatic Failover (PAF) voldoet aan de nodige requirements, zeker met de tool Pacemaker is er heel veel mogelijk qua monitoring, failover en replicatie. De score hierbij voor Must have is 13/15 punten.

Should have

Ook PostgreSQL Automatic Failover (PAF) is open source. De laatste release was op 10 maart 2020. Hieruit kunnen we concluderen dat ook deze oplossing nog up-to-date is en bruikbaar voor dit onderzoek. Doordat er (nog) geen nieuwe release is in 2021, krijgt PostgreSQL Automatic Failover (PAF) ook 4/5 punten voor Should have, en geen 5/5.

Could have

Op eerste zicht lijkt er niet direct een alomgekende grafische interface aanwezig te zijn voor PostgreSQL Automatic Failover (PAF).

PostgreSQL Automatic Failover (PAF) kan zo geconfigureerd worden dat het weinig tot geen manuele interventie vereist. Aan de hand van bijvoorbeld Pacemaker is dit mogelijk.

4.3.4 Oplossing 4: Replication Manager (RepMgr)

Must have

Replication Manager (RepMgr) is een oplossing ontwikkeld voor het beheren van replicatie en failover van PostgreSQL clusters. Het biedt de tools aan om replicatie van PostgreSQL op te zetten, te configureren, te beheren en te monitoren. Het laat ook toe om handmatige omschakeling en failover taken uit te voeren met behulp van repmgr utility. Dit is een gratis tool die ondersteuning en verbetering biedt van PostgreSQL's ingebouwde streaming replicatie. Voorbeelden van tools zijn repmgr en repmgrd. Replication Manager (RepMgr) biedt ook de tools om primary en standby nodes op te zetten, en om in geval van faalscenario automatische failover te doen.

De functionaliteiten van Replication Manager (RepMgr) sluiten voldoende aan op de vooropgestelde Must haves en krijgt hiervoor 12/15 punten.

Should have

Replication Manager (RepMgr) is open source, ontwikkeld door 2ndQuadrant. De laatste release was op 22 oktober 2020.

Replication Manager (RepMgr) krijgt hierdoor een score van 4/5 voor Should have. Doordat er (nog) geen release is in 2021, krijgt het geen 5/5.

Could have

Op eerste zicht lijkt er niet direct een alomgekende grafische interface aanwezig te zijn voor Replication Manager (repmgr).

Bij Replication Manager (repmgr) is er nog veel dat manueel moet gedaan worden.

Requirementanalyse Patroni	
Must have	
Ondersteuning van redundancy	4
Ondersteuning van failover	4
Ondersteuning van monitoring	4
Should have	
Open Source	2.5
Ondersteuning in 2020-2021	2.5
Totaal	17/20

Tabel 4.1: Requirementanalyse Patroni

Requirementanalyse Pgpool-II	
Must have	
Ondersteuning van redundancy	3
Ondersteuning van failover	4
Ondersteuning van monitoring	3
Should have	
Open Source	2.5
Ondersteuning in 2020-2021	1.5
Totaal	14/20

Tabel 4.2: Requirementanalyse Pgpool-II

4.4 Resultatenanalyse

4.4.1 Resultaten requirements

In onderstaande tabellen wordt de onderverdeling van de punten van de verschillende oplossingen duidelijk gemaakt. Bij Must have zijn er 3 items waar per item 5 punten te verdienen zijn. Should have kent 2 items, die samen voor 5 punten meetellen (2 x 2.5). De Could have items worden niet meegerekend omdat deze van geen belang zijn geweest in de keuze naar een oplossing. Het totaal van de punten staat op 20. De twee oplossingen met de hoogste score zullen gebruikt worden voor een nog meer verdiepende studie die meer in detail gaat bij deze twee oplossingen. De oplossing met de hoogste score zal ook gebruikt worden bij het opzetten van de proof-of-concept.

In de laatste tabel is er een duidelijke onderverdeling in score van de oplossingen. In dit onderzoek zal verder verdiept worden in Patroni en PostgreSQL Automatic Failover en bijhorende tools. Voor proof of concept zal gebruik worden gemaakt van Patroni.

Requirementanalyse PostgreSQL Automatic Failover (PAF)	
Must have	
Ondersteuning van redundancy	4
Ondersteuning van failover	4
Ondersteuning van monitoring	4
Should have	
Open Source	2.5
Ondersteuning in 2020-2021	1.5
Totaal	16/20

Tabel 4.3: Requirementanalyse PostgreSQL Automatic Failover (PAF)

Requirementanalyse Replication Manager (repmgr)	
Must have	
Ondersteuning van redundancy	4
Ondersteuning van failover	4
Ondersteuning van monitoring	3
Should have	
Open Source	2.5
Ondersteuning in 2020-2021	1.5
Totaal	15/20

Tabel 4.4: Requirementanalyse Replication Manager (repmgr)

Requirementanalyse Alle Oplossingen	
Patroni	17/20
Pgpool-II	14/20
PostgreSQL Automatic Failover (PAF)	16/20
Replication Manager (repmgr)	15/20

Tabel 4.5: Requirementanalyse alle oplossingen

5. Oplossing 1: Patroni

In dit hoofdstuk worden nog eens voor Patroni alle punten uit de functionele requirements overlopen. Hier kunnen al meer specifieke commando's en tools benoemd worden die gebruikt kunnen worden voor de opstelling van de proof of concept.

5.1 Redundantie/Replicatie

Voor het kopiëren van bestanden of databases, of replicatie genaamd maakt Patroni gebruikt van streaming replicatie. Standaard maakt Patroni gebruik van asynchrone replicatie. Hierbij worden gegevens eerst geschreven naar een primaire opslagarray en committeert dan de te repliceren gegevens naar het geheugen. Vervolgens worden deze gegevens in real-time of met geplande tussenpozen naar de replicatiedoelen gekopieerd. Wanneer gewerkt wordt met asynchrone replicatie, kan de cluster het zich permitteren om een aantal gecommitteerde transacties te verliezen, om High Availability te garanderen. Wanneer de primary node faalt, en Patroni een standby node regelt, zullen alle transacties die niet naar de standby node zijn gerepliceerd, verloren gaan. Het aantal transacties dat verloren mag gaan, kan geregeld worden aan de hand van maximum_lag_on_failover. Dit zal bij failover het aantal verloren transacties beperken.

PostgreSQL synchrone replicatie is ook mogelijk bij Patroni. Synchrone replicatie zorgt voor meer consistentie van data in een cluster door te bevestigen dat schrijfacties naar een secundary node worden geschreven voordat ze naar de verbindende client terugkeren met een succes. De nadelen van synchrone replicatie houden in dat er een verminderde verwerkingscapaciteit is voor schrijfacties. Deze verwerkingscapaciteit is volledig gebaseerd op netwerkprestaties. Het gebruik van PostgreSQL synchrone replicatie garandeert niet dat

er onder alle omstandigheden geen transacties verloren zullen gaan. Wanneer de primary node en de secundary node die op dat moment synchrone replicatie draaien, gelijktijdig falen, zal een derde node, die mogelijk niet alle transacties bevat, worden gepromoveerd tot primary node.

5.2 Failover

Patroni clusters kennen een automatische failover wanneer de primary node (onverwachts) niet beschikbaar is. De failover opties kunnen zeer geavanceerd aangepast worden naar de wensen van de cluster. Wanneer, na failover, een primary node terug online komt, kan deze aan de hand van pg_rewind terug toegevoegd worden aan de cluster als standby node. Patroni werkt aan de hand van endpoints. Dit kunnen switchover of failover endpoints zijn. Een failover endpoint laat toe om een manuele failover uit te voeren wanneer er in de cluster geen 'gezonde' nodes meer aanwezig zijn. Deze endpoint wordt gebruikt door patronictl failover. Aan de hand van dynamische configuratie instellingen, opgeslagen in het Distributed Configuration Store (DCS), die geldig zijn voor alle nodes, worden parameters zoals ttl, maximum_lag_on_failover, retry_timeout, etc. ingesteld. Bij ttl wordt dan de tijd ingesteld die nodig is om de leader lock in te nemen als standby node. Normaal bevat de primary node altijd deze lock, maar als er iets misloopt en de primary node door één of andere reden deze lock misloopt, kan een standby node deze leader lock overpakken en op deze manier zichzelf promoten tot primary node en zo failover initiëren. Patroni kan bijvoorbeeld worden verteld om nooit een standby node te promoten die meer dan een configureerbare hoeveelheid log achterloopt op de primary node.

5.3 Monitoring

Patroni kent een zeer uitgebreide REST API die gebruikt kan worden om de cluster en zijn nodes te monitoren en voor applicaties om te selecteren met welke PostgreSQL instantie verbinding moet worden gemaakt. Aan de hand van Consul kan je monitoring aanzetten in de cluster. Dit moet dan wel op elke node aangezet worden. Er zijn ook tools beschikbaar waarbij monitoring de enige functionaliteit is. PGWatch/PGWatch2 zijn hier voorbeelden van. PGWatch maakt gebruik van data die al verzameld wordt in PostgreSQL queries. Dus toevoegen of wijzigen van deze queries kan zeer eenvoudig gebeuren.

6. Oplossing 2: PostgreSQL Automatic Failove

In dit hoofdstuk worden nog eens voor PostgreSQL Automatic Failover (PAF) alle punten uit de functionele requirements overlopen. Hier kunnen al meer specifieke commando's en tools benoemd worden die gebruikt kunnen worden voor de opstelling van een PostgreSQL Automatic Failover (PAF) cluster.

6.1 Redundantie/Replicatie

6.2 Failover

Aan de hand van Pacemaker kan er automatische failover plaatsvinden.

6.3 Monitoring

7. Conclusie

Hierin zetten welke tool er gewonnen heeft, in dit geval Patroni.

1. Antwoord op Hoofdonderzoeksvraag

2. Antwoord op deelonderzoekvragen

8. Proof of Concept

In dit hoofdstuk wordt de opbouw van de Patroni cluster uitgelegd. Hierin zal worden gekeken hoe de cluster aan de functionele requirements, namelijk redundantie/replicatie, failover en monitoring voldoet.

8.1 Omgeving

De opzet van deze proof of concept zal gebeuren in een lokale Linux-omgeving, namelijk Ubuntu 21.04. De cluster zal bestaan uit drie virtuele machines die zullen draaien op Ubuntu Xenial 16.04. De eerste virtuele machine (pgServer) is een servernode waarop Consul zal draaien, samen met pgBouncer en een HAProxy. De andere twee virtuele machines (pgNode1 en pgNode2) zullen PostgreSQL, Patroni en een Consul Agent draaien.

De cluster zal dus zo geconfigueerd worden dat er één primary node is met een standby node die asynchrone streaming replicatie verricht.

8.2 Prerequisites

8.2.1 Vagrant

Bij de opzet van cluster zal gebruik worden gemaakt van Vagrant. Vagrant is een tool voor het bouwen en beheren van virtuele machine-omgevingen (Kalow, 2020).

In onderstaande code snippet is de code voor de Vagrantfile zichtbaar. Hierin wordt be-

paald welke servers aangemaakt worden en welke opties zij bezitten. Zo zal elke node een RAM-geheugen hebben van 1 GB (1024 MB). Ook het ip-adres zal hierin worden toegewezen per node. Er wordt ook een file meegegeven die zal worden uitgevoerd in de server. Deze file bevat de opbouw van de cluster, zoals het installeren van Consul, het configureren van Patroni op de nodes, en de setup van HAProxy en pgBouncer.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = "ubuntu/xenial64"
    config.vm.provider "virtualbox" do |v|
        v.customize ["modifyvm", :id, "--memory", 1024]
        v.gui = true
    end

    config.vm.define :pgServer do |pgServer_config|
        pgServer_config.vm.hostname = 'pgServer'
        pgServer_config.vm.network :private_network, ip: "172.16.0.11"
        pgServer_config.vm.provision :shell, :path =>
            "postgresql-cluster-setup.sh"
    end
    config.vm.define :pgNode1, primary: true do |pgNode1_config|
        pgNode1_config.vm.hostname = 'pgNode1'
        pgNode1_config.vm.network :private_network, ip: "172.16.0.22"
        pgNode1_config.vm.provision :shell, :path =>
            "postgresql-cluster-setup.sh"
    end
    config.vm.define :pgNode2 do |pgNode2_config|
        pgNode2_config.vm.hostname = 'pgNode2'
        pgNode2_config.vm.network :private_network, ip: "172.16.0.33"
        pgNode2_config.vm.provision :shell, :path =>
            "postgresql-cluster-setup.sh"
    end
end
```

8.2.2 VirtualBox

In deze proof of concept zal gebruik gemaakt worden van virtuele machines via VirtualBox. VirtualBox is een open source virtualisatiesoftware. Het werkt als een hypervisor en kan op deze manier virtuele machines aanmaken, waarin de gebruiker naar eigen keuze verschillende besturingssytemen kan emuleren. Bij de configuratie van een virtuele machine kunnen de specificaties zoals RAM-geheugen, schijfruimte, etc. bepaald worden.

8.3 pgServer

Tijdens de configuratie van de cluster zal pgServer '172.16.0.11' als ip adres hebben.

Als eerste stap moeten de juiste packages geïnstalleerd worden op pgServer. Hier gaat het over Python, Consul, pgBouncer en HAProxy.

Hierna zullen we voor Consul een service toevoegen die bij het opstarten van de server wordt uitgevoerd. Belangrijk is dat bij het aanmaken van deze service, de service gestart en ge-enabled moet zijn. In deze service verwijzen we naar de config file voor de Consul server. Deze ziet er als volgt uit.

```
{
    "advertise_addr": "172.16.0.11",
    "bind_addr": "172.16.0.11",
    "bootstrap": false,
    "bootstrap_expect": 1,
    "server": true,
    "client_addr": "0.0.0.0",
    "node_name": "pgServer",
    "datacenter": "dc1",
    "data_dir": "/var/consul/server",
    "domain": "consul",
    "encrypt": "/q/vkVS+My2nl8Zk/8csuQ==",
    "log_level": "INFO",
    "enable_syslog": true,
    "rejoin_after_leave": true,
    "ui_dir": "/var/consul/ui",
    "leave_on_terminate": false,
    "skip_leave_on_interrupt": false
}
```

Bij het installeren van de HAProxy wordt doorverwezen naar een configuratiefile waarin de verschillende opties worden meegegeven.

```
global
    maxconn 100

defaults
    log global
    mode tcp
    retries 2
    timeout client 30m
    timeout connect 4s
    timeout server 30m
    timeout check 5s

listen stats
```

```
    mode http
    bind *:7000
    stats enable
    stats uri /

listen postgres
    bind *:5000
    option httpchk
    http-check expect status 200
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
    server postgresql_pgNode1_172.0.16.22 172.16.0.22:5432 maxconn 100
        check port 8008
    server postgresql_pgNode2_172.0.16.33 172.16.0.33:5432 maxconn 100
        check port 8008
```

De pgBouncer.ini file ziet er als volgt uit:

```
[databases]
postgres = host=172.16.0.11 port=5000 pool_size=6
template1 = host=172.16.0.11 port=5000 pool_size=6
test = host=172.16.0.11 port=5000 pool_size=6

[pgbouncer]
logfile = /var/log/postgresql/pgbouncer.log
pidfile = /var/run/postgresql/pgbouncer.pid
listen_addr = *
listen_port = 6432
unix_socket_dir = /var/run/postgresql
auth_type = trust
auth_file = /etc/pgbouncer/userlist.txt
admin_users = postgres
stats_users =
pool_mode = transaction
server_reset_query =
server_check_query = select 1
server_check_delay = 10
max_client_conn = 1000
default_pool_size = 12
reserve_pool_size = 5
log_connections = 1
log_disconnections = 1
log_pooler_errors = 1
```

8.4 pgNode1

Tijdens de configuratie van de cluster zal pgNode1 ’172.16.0.22’ als ip adres hebben.

Idem als bij pgServer is de eerste stap het installeren van de juiste packages. Hier gaat het over Python, Consul Agent, Patroni en PostgreSQL uiteraard.

Bij het configureren van Patroni zal verwezen worden naar pgNode1Patroni.yml waarin de configuratie te vinden is. Zoals eerder al vermeld is geweest, is de configuratie van Patroni te vinden in het DCS, het Distributed Configuration Store. Hierin staat onder andere de configuratie voor pg_rewind, maximum_lag_on_failover, etc.:

```
scope: PatroniCluster
namespace: /nsPatroniCluster
name: pgNode1

log:
    level: INFO
    dir: /var/log/patroni
    file_size: 10485760 #staat gelijk aan ongeveer 10MB

restapi:
    listen: 172.16.0.22:8008
    connect_address: 172.16.0.22:8008

consul:
    host: 172.16.0.11:8500

bootstrap:
    dcs:
        ttl: 30
        loop_wait: 10
        retry_timeout: 10
        maximum_lag_on_failover: 1048576 #staat gelijk aan ongeveer 1MB
        master_start_timeout: 300
        postgresql:
            use_pg_rewind: true
            parameters:
                archive_command: 'exit 0'
                archive_mode: 'on'
                autovacuum: 'on'
                checkpoint_completion_target: 0.6
                checkpoint_warning: 300
                datestyle: 'iso, mdy'
                default_text_search_config: 'pg_catalog.english'
                effective_cache_size: '128MB'
                hot_standby: 'on'
                include_if_exists: 'repmgr_lib.conf'
                lc_messages: 'C'
                listen_addresses: '*'
                log_autovacuum_min_duration: 0
                log_checkpoints: 'on'
                logging_collector: 'on'
```

```
                log_min_messages: INFO
                log_filename: 'postgresql.log'
                log_connections: 'on'
                log_directory: '/var/log/postgresql'
                log_disconnections: 'on'
                log_line_prefix: '%t [%p]: [%l-1] user=%u,db=%d,app=%a '
                log_lock_waits: 'on'
                log_min_duration_statement: 0
                log_temp_files: 0
                maintenance_work_mem: '128MB'
                max_connections: 101
                max_wal_senders: 5
                port: 5432
                shared_buffers: '128MB'
                shared_preload_libraries: 'pg_stat_statements'
                unix_socket_directories: '/var/run/postgresql'
                wal_buffers: '8MB'
                wal_keep_segments: '200'
                wal_level: 'replica'
                work_mem: '128MB'

    initdb:
    - encoding: UTF8
    - data-checksums

    pg_hba:
    - host replication replicator 127.0.0.1/32 md5
    - host replication replicator 172.16.0.22/0 md5
    - host replication replicator 172.16.0.33/0 md5
    - host all postgres 172.16.0.11/32 trust
    - host all postgres 172.16.0.22/32 trust
    - host all postgres 172.16.0.33/32 trust
    - host all all 0.0.0.0/0 md5
    - local all all peer

    users:
        admin:
            password: admin
            options:
                - createrole
                - createdb

postgresql:
    listen: 127.0.0.1,172.16.0.22:5432
    connect_address: 172.16.0.22:5432
    data_dir: /var/lib/postgresql/patroni
    pgpass: /tmp/pgpass
    use_unix_socket: true
    authentication:
```

```
        replication:
            username: replication
            password: replication
        superuser:
            username: postgres
            password: postgres
        parameters:
            unix_socket_directories: '/var/run/postgresql'
            wal_compression: on

tags:
    nofailover: false
    noloadbalance: false
    clonefrom: false
    nosync: false

watchdog:
    mode: off
```

Hierna zal opnieuw voor Consul een service toegevoegd worden die bij het opstarten van de server wordt uitgevoerd. Belangrijk is dat bij het aanmaken van deze service, de service gestart en ge-enabled moet zijn. In deze service verwijzen we naar de config file voor de Consul agent (cliënt). Deze ziet er als volgt uit.

```
    {
        "server": false,
        "datacenter": "dc1",
        "data_dir": "/var/consul/client",
        "ui_dir": "/var/consul/ui",
        "encrypt": "/q/vkVS+My2nl8Zk/8csuQ==",
        "log_level": "INFO",
        "enable_syslog": true,
        "start_join": ["172.16.0.11"],
        "bind_addr": "172.16.0.22"
    }
```

8.5 pgNode2

Tijdens de configuratie van de cluster zal pgNode2 '172.16.0.33' als ip adres hebben.

Idem als bij pgServer is de eerste stap het installeren van de juiste packages. Hier gaat het over Python, Consul Agent, Patroni en PostgreSQL uiteraard.

Bij het configureren van Patroni zal verwezen worden naar pgNode2Patroni.yml waarin de configuratie te vinden is. Dit bestand is zeer gelijkend aan wat er bij pgNode1 staat,

maar kent toch een paar kleine wijzigingen, maar enkel in ip-adressen:

```
restapi:
    listen: 172.16.0.33:8008
    connect_address: 172.16.0.33:8008

consul:
    host: 172.16.0.11:8500
```

```
postgresql:
    listen: 127.0.0.1,172.16.0.33:5432
    connect_address: 172.16.0.33:5432
    data_dir: /var/lib/postgresql/patroni
    pgpass: /tmp/pgpass
    use_unix_socket: true
    authentication:
        replication:
            username: replication
            password: replication
        superuser:
            username: postgres
            password: postgres
    parameters:
        unix_socket_directories: '/var/run/postgresql'
        wal_compression: on
```

Hierna zal opnieuw voor Consul een service toegevoegd worden die bij het opstarten van de server wordt uitgevoerd. Belangrijk is dat bij het aanmaken van deze service, de service gestart en ge-enabled moet zijn. In deze service verwijzen we naar de config file voor de Consul agent (cliënt). Deze ziet er als volgt uit.

```
    {
        "server": false,
        "datacenter": "dc1",
        "data_dir": "/var/consul/client",
        "ui_dir": "/var/consul/ui",
        "encrypt": "/q/vkVS+My2nl8Zk/8csuQ==",
        "log_level": "INFO",
        "enable_syslog": true,
        "start_join": ["172.16.0.11"],
        "bind_addr": "172.16.0.33"
    }
```

Ip-adressen	
pgServer	172.16.0.11
pgNode1	172.16.0.22
pgNode2	172.16.0.33

Tabel 8.1: Ip-adressen Cluster

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Samenvatting

In deze bachelorproef zal onderzoek gedaan worden naar de huidige staat van open-source database high availability (HA) tooling. De vraag naar PostgreSQL (pgSQL) wordt steeds groter, waardoor een onderzoek naar open-source database high availability (HA) tooling zeker niet onmisbaar is. In het eerste deel van dit onderzoek zal er geduid worden wat open-source database high availability (HA) tooling precies is en wat de toepassingen ervan zijn. Hierna wordt een vergelijkende studie uitgevoerd over de verschillende open-source database high availability (HA) oplossingen. Na de vergelijkende studie wordt als proof-of-concept een PostgreSQL (pgSQL) cluster opgezet met behulp van de automation tool Puppet. Dit zorgt voor automatisering en reproduceerbaarheid. In dit onderzoek zal ik worden bijgestaan door Ruben Demey, Global IT Operations Manager bij ST Engineering iDirect. Het verwachte resultaat van dit onderzoek is aantonen dat er een PostgreSQL high availability (ha) cluster oplossing aanwezig is die voldoet aan de requirements, opgesteld in dit onderzoek. In de toekomst is het interessant om soortgelijk onderzoek uit te voeren, om te kijken of de tools die in dit onderzoek worden aangehaald, nog steeds even waardevol zullen zijn.

A.2 Introductie

Database high availability (HA) staat voor de garantie van het behouden van gegevens in geval er zich een defect of storing voordoet aan de databank server. Een storing aan een databank server kan te wijten zijn aan verschillende factoren. Voorbeelden hiervan zijn het verlies van netwerkconnectie en een defect in de software of hardware van de databankserver. Ook menselijke factoren en omgevingsfactoren moeten in rekening genomen worden. Voorbeelden hiervan zijn een menselijke vergissing en een wijziging in temperatuur. Investeren in high availability geeft meer zekerheid over de beschikbaarheid van data en biedt verschillende mogelijkheden voor failover en systeembescherming (IBM, 2019). Met behulp van clusters kan er één actieve en een of meerdere standby instanties van de databank server zijn. Een cluster is een groep van servers en computers die samenwerken met elkaar alsof het één systeem is. Deze standby instanties zullen, in het ideale geval, dezelfde gegevens bevatten als de actieve server (Bermingham, 2019). Wanneer dan een actieve server faalt, kan een standby instantie inspringen waardoor dataverlies en server downtime gereduceerd worden.

A.3 State of the art

Singer spreekt over high availability (HA) clustering als een groep van servers die applicaties en services ondersteunen die op een betrouwbare manier gebruikt kunnen worden met een minimaal aantal aantal downtimes. Hij bespreekt de cluster architectuur en wat de best practice is voor high availability (HA) binnen een cluster. De conclusie die hier getrokken wordt, is dat het primaire doel van een high availability (HA) systeem het voorkomen en elimineren van alle single points of failure zijn. Dit systeem moet beschikken over meerdere geteste actieplannen. Dit zodat ze in geval van storing, verstoring en defect in dienstverlening direct, gepast en onafhankelijk kunnen reageren. Zorgvuldige planning + betrouwbare implementatiemethoden + stabiele softwareplatforms + degelijke hardware-infrastructuur + vlotte technische operaties + voorzichtige managementdoelstellingen + consistente databeveiliging + voorspelbare redundantiesystemen + robuuste backupoplossingen + meerdere herstelopties = 100% uptime (Singer, 2020). Ook Jevtic heeft het over veel van de punten die hierboven reeds zijn aangehaald. Jevtic spreekt over een highly available architectuur waarin meerdere componenten samenwerken om een ononderbroken service gedurende een bepaalde periode te garanderen. Dit omvat ook de reactietijd op verzoeken van gebruikers. Jevtic kenmerkt een highly available (HA) infrastructuur aan de hand van: 1. Hardware redundantie; 2. Software en applicatie redundantie; 3. Gegevens redundantie; 4. Elimineren van storingspunten (Jevtic, 2018). Akhtar heeft vier van de meest gebruikte database high availability (HA) oplossingen opgelijst. Deze vier zijn "PgPool-II", "PostgreSQL Automatic Failover (PAF)", "RepMgr [Replication Manager] ", en "Patroni"(Akhtar, 2020). Akhtar vergelijkt deze verschillende oplossingen kort met elkaar. Akhtar definieert high availability (HA) als niet alleen de continuïteit van een bepaalde service, maar volgens hem gaat high availability (HA) ook over het vermogen van een systeem om een (hogere) werkdruk te kunnen schalen en te beheren. Dit systeem moet volgens Akhtar de gemiddelde werkdruk, maar ook de piekmomenten aankunnen.

Aldus Andersen zijn de top drie open-source databanken van 2019, in volgorde van top 3, MySQL met 31.7%, PostgreSQL met 13.4% en MongoDB met 12.2% (Anderson, 2020) van het totaal aantal open-source databank gebruikers.

A.4 Methodologie

In de eerste fase van het onderzoek zal er een vergelijkende studie gebeuren over de huidige, database high availability (HA) oplossingen. Deze verschillende tools/oplossingen zullen dan met elkaar vergeleken worden. Hierbij wordt gekeken naar welke elementen er allemaal (meermaals) voorkomen. Hiervan komt er een lijst die gebruikt zal worden om te schiften tussen de verschillende oplossingen. Op deze manier zal er dan een oplossing gekozen worden. Via deze methode wordt er gekeken om maximum 3 verschillende oplossingen uit te werken, waarbij één oplossing gebruikt zal worden bij de proof of concept. In de tweede fase van het onderzoek wordt de focus gelegd op het opzetten van de PostgreSQL (pgSQL) cluster als proof of concept. PostgreSQL is een open-source, object-relationeel databank systeem (PostgreSQL, 2020). Bij Inuits, een Belgisch open-source bedrijf met verschillende vestigingen in Europa, merken ze een stijging in de vraag naar het PostgreSQL verhaal. Deze zal vooraf gegaan worden door een literatuurstudie over PostgreSQL (pgSQL). Aan de hand hiervan zal er gewerkt worden aan het opbouwen van de PostgreSQL (pgSQL) cluster. Vooraleer dit geautomatiseerd wordt, zal de opbouw manueel verlopen. De opbouw zal gebeuren via virtuele machines (VirtualBox) waarop Linux-distributies staan. In het onderzoek zal dus gebruik gemaakt worden van Linux-servers. De keuze van Linux-distributie zal onderbouwd worden in dit onderzoek. De opbouw van de cluster zal telkens grondig gedocumenteerd worden. Alle commando's zullen hierbij overlopen worden. Hierna zal er een inleidende literatuurstudie zijn over Puppet en zal er aansluitend via Puppet gewerkt worden om deze PostgreSQL (pgSQL) cluster te reproduceren. Ook hier zal alles grondig gedocumenteerd worden. Na het opzetten van de PostgreSQL (pgSQL) cluster zal er getest worden of de gebruikte database high availability (HA) oplossing functioneel is. Deze testen zullen uitgebreid beschreven worden.

A.5 Verwachte resultaten

Uit het onderzoek zal blijken dat verschillende database high availability (HA) oplossingen mogelijk zijn binnen een PostgreSQL (pgSQL) cluster. De opbouw van deze cluster zal gebeuren aan de hand van virtuele machines. Wanneer de virtuele machine, waarop de PostgreSQL server staat, uitvalt, zal er een standby instantie van deze server het werk van de actieve, uitgevallen server overnemen. Hierdoor zal er geen downtime of dataverlies zijn. De data zal beschikbaar en onverstoord blijven.

Uit dit onderzoek zullen ook best practices volgen die gehanteerd kunnen worden om op die manier het risico op verlies van gegevens te verminderen. De kans om offline te zijn zal lager liggen met een high available systeem.

Door gebruik te maken van Puppet zal de reproduceerbaarheid van de high available (HA) PostgreSQL (pgSQL) cluster zeer eenvoudig en snel moeten verlopen.

A.6 Verwachte conclusies

Uit het onderzoek zal blijken dat database high availability (HA) een blijvend topic is waar voldoende aandacht aan besteed moet worden in kleine en grote bedrijven. Zonder de implementatie van een high available (HA) architectuur kan een storing of downtime van de databank (SQL) server grote gevolgen hebben op een bedrijf/organisatie. Gevolgen zoals verlies van vertrouwen bij klanten, verlies van inkomen, verlies van informatie. Door middel van hardware-, software- en gegevensredundantie en het elimineren van mogelijke storingspunten zal er high availability gegarandeerd worden. De kost en tijd die geïnvesteerd moet worden in het onderhouden van een high available systeem zal lager liggen dan de kost en tijd die geïnvesteerd moet worden in geval van een downtime. Met de proof of concept wordt dan aangetoond dat database high availability (HA) eenvoudig te implementeren valt in een PostgreSQL (pgSQL) cluster.

Bibliografie

Ahmad, K. S., Ahmad, N., Tahir, H. & Khan, S. (2017). Fuzzy$_{MoSCoW}$: *A fuzzy based MoSCoW method for 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*.

Akhtar, H. (2020, augustus 10). *PostgreSQL High Availability: The Considerations and Candidates*. Verkregen 2 januari 2021, van https://www.highgo.ca/2020/08/10/postgresql-high-availability-the-considerations-and-candidates/

Anderson, K. (2020, januari 17). *2019 Open Source Database Report*. DZone. Verkregen 31 december 2020, van https://dzone.com/articles/2019-open-source-database-report-top-databases-pub

Augustine, J. (2019, juli 10). *PostgreSQL WAL Retention and Clean Up: pg$_a$rchivecleanup*. Percona. Verkregen 13 april 2021, van https://www.percona.com/blog/2019/07/10/wal-retention-and-clean-up-pg_archivecleanup/

AVINetworks. (2020, september 26). *Failover Definition*. AVI Networks. Verkregen 1 april 2021, van https://avinetworks.com/glossary/failover/

Barman. (2020a). *Barman*. Barman. Verkregen 13 april 2021, van https://www.pgbarman.org/

Barman. (2020b). *Bringing you the Espresso Backup! About Barman*. Barman. Verkregen 13 april 2021, van https://www.pgbarman.org/about/

Bermingham, D. (2019, mei 9). *Clustering for SQL Server High Availability*. Big Data Quarterly (BDQ). Verkregen 31 december 2020, van https://www.dbta.com/BigDataQuarterly/Articles/Clustering-for-SQL-Server-High-Availability-131639.aspx

Carchedi, N. (2020, oktober 23). *What is SQL?* https://www.datacamp.com/community/tutorials/what-is-sql?utm_source=adwords_ppc&utm_campaignid=898687156&utm_adgroupid=48947256715&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=229765585183&utm_

targetid=dsa-429603003980&utm_loc_interest_ms=&utm_loc_physical_ms=1001208&gclid=Cj0KCQjwse-DBhC7ARIsAI8YcWJOxNwWfRDfjloPnAcswV6QsafwgHI0hqagQFH5fj59JLPM04NOxMaAmudEALw_wcB

cawrites; Craig Guyer; Jason Roth, P. M. M. R. T. S. M. (2016, mei 17). *About Log Shipping (SQL Server)*. Microsoft. Verkregen 25 maart 2021, van https://docs.microsoft.com/en-us/sql/database-engine/log-shipping/about-log-shipping-sql-server?view=sql-server-ver15

CDNF. (2020, juli 22). *What is a CNF?* cdnf. Verkregen 8 april 2021, van https://cdnf.io/what_is_cnf/

Codecademy. (2018). What is a Relational Database Management System? Learn about RDBMS and the language used to access large datasets SQL. https://www.codecademy.com/articles/what-is-rdbms-sql#:~:text=A%5C%20relational%5C%20database%5C%20is%5C%20a,database%5C%20is%5C%20organized%5C%20into%5C%20tables

CriticalCase. (2020). *5 REASONS WHY YOU NEED HIGH-AVAILABILITY FOR YOUR BUSINESS: What does it mean High-Availability infrastructure (HA)?* CriticalCase. Verkregen 12 mei 2021, van https://www.criticalcase.com/blog/5-reasons-why-you-need-high-availability-for-your-business.html

CSharp-Corner. (2019, september 6). *What Are Object-Oriented Databases And Their Advantages*. https://www.c-sharpcorner.com/article/what-are-object-oriented-databases-and-their-advantages2/#:~:text=An%20object-oriented%20database%20(OODBMS,object-oriented%20database%20management%20systems

DB-Engines. (2021a). *DB-Engines Ranking*. DB-Engines. Verkregen 25 maart 2021, van https://db-engines.com/en/ranking

DB-Engines. (2021b). *PostgreSQL System Properties*. DB-Engines. Verkregen 25 maart 2021, van https://db-engines.com/en/system/PostgreSQL

de Rorthais, M. R. J.-G. (2020). *PostgreSQL Automatic Failover*. clusterlabs. Verkregen 15 april 2021, van https://clusterlabs.github.io/PAF/

IBM. (2019). *High availability for databases*. International Business Machines Corporation (IBM). Verkregen 31 december 2020, van https://www.ibm.com/support/knowledgecenter/SSANHD_7.6.1.2/com.ibm.mbs.doc/gp_highavail/c_ctr_ha_for_databases.html

Ihalainen, A. V. F. L. C. J. A. N. (2019, april 4). *Replication Between PostgreSQL Versions Using Logical Replication*. Percona. Verkregen 15 april 2021, van https://www.percona.com/blog/2019/04/04/replication-between-postgresql-versions-using-logical-replication/#:~:text=Logical%20replication%20in%20PostgreSQL%20allows,is%20not%20open%20for%20writes.

Insausti, S. (2019, juli 18). *Scaling PostgreSQL for Large Amounts of Data*. Several Nines. Verkregen 1 april 2021, van https://severalnines.com/database-blog/scaling-postgresql-large-amounts-data

Jevtic, G. (2018, juni 22). *What is High Availability Architecture? Why is it Important?* Verkregen 25 maart 2021, van https://phoenixnap.com/blog/what-is-high-availability

Kalow, B. (2020). *Introduction to Vagrant*. HashiCorp. Verkregen 9 mei 2021, van https://www.vagrantup.com/intro

Kelly, K. (2020, oktober 1). *What is Puppet and What Role Does it Play in DevOps? What is Puppet?* Liquid Web. Verkregen 23 maart 2021, van https://www.liquidweb.com/kb/what-is-puppet-and-what-role-does-it-play-in-devops/

Kumar, V. (2020, april 7). *What Does "Database High Availability"Really Mean?* EDB. Verkregen 25 maart 2021, van https://www.enterprisedb.com/blog/what-does-database-high-availability-really-mean

Lutkevich, B. (2021). *high availability (HA): What is high availability?* (A. S. Gillis, Red.). TechTarget. Verkregen 25 maart 2021, van https://searchdatacenter.techtarget.com/definition/high-availability

Markwort, J. (2018, september 27). *PATRONI : SETTING UP A HIGHLY AVAILABLE POSTGRESQL CLUSTER*. Cybertec. Verkregen 8 april 2021, van https://www.cybertec-postgresql.com/en/patroni-setting-up-a-highly-available-postgresql-cluster/

MySQL. (2021). *1.5 Point-in-Time (Incremental) Recovery*. MySQL. Verkregen 22 april 2021, van https://dev.mysql.com/doc/mysql-backup-excerpt/8.0/en/point-in-time-recovery.html#:~:text=Point-in-time%20recovery%20refers,time%20the%20backup%20was%20made

Nethosting. (2019, mei 13). *MySQL vs. PostgreSQL: 2019 Showdown*. https://nethosting.com/mysql-vs-postgresql-2019-showdown/

opensource.com. (2021). *What is open source?* Red Hat Inc. Verkregen 29 april 2021, van https://opensource.com/resources/what-open-source

Oracle. (2021). *Database defined*. Oracle. Verkregen 15 april 2021, van https://www.oracle.com/database/what-is-database/

pgpool. (2021). *What is Pgpool-II?* pgpool. Verkregen 15 april 2021, van https://www.pgpool.net/mediawiki/index.php/Main_Page

PostgreSQL. (2020). PostgreSQL. Verkregen 31 december 2020, van https://www.postgresql.org/

PostgreSQL. (2021a). *About: What is PostgreSQL?* PostgreSQL. Verkregen 25 maart 2021, van https://www.postgresql.org/about/

PostgreSQL. (2021b). *pg_basebackup*. PostgreSQL. Verkregen 22 april 2021, van https://www.postgresql.org/docs/10/app-pgbasebackup.html

PostgreSQL. (2021c). *pg_rewind*. PostgreSQL. Verkregen 15 april 2021, van https://www.postgresql.org/docs/12/app-pgrewind.html

PostgreSQL. (2021d). *What is Postgres?* PostgreSQL. Verkregen 25 maart 2021, van https://www.postgresql.org/docs/6.3/c0101.htm

postgresql. (2021). *PostgreSQL: The World's Most Advanced Open Source Relational Database*. PostgreSQL. Verkregen 25 maart 2021, van https://www.postgresql.org/

Raima. (2021). *High-Availability Database (HA DB)*. Raima. Verkregen 20 april 2021, van https://raima.com/rdme-high-availability-database/

repmgr. (2021a). *repmgr standby switchover*. repmgr. Verkregen 13 april 2021, van https://repmgr.org/docs/4.0/repmgr-standby-switchover.html

repmgr. (2021b). *WHAT IS REPMGR?* repmgr. Verkregen 15 april 2021, van https://repmgr.org/

Ruby. (g.d.). *About Ruby*. Ruby. Verkregen 23 maart 2021, van https://www.ruby-lang.org/en/about/

ScaleGrid. (2018a, augustus 22). *Managing High Availability in PostgreSQL Part III: Patroni*. ScaleGrid. Verkregen 8 april 2021, van https://scalegrid.io/blog/managing-high-availability-in-postgresql-part-3/#:~:text=Patroni%20ensures%20the%20end-to,be%20customized%20to%20your%20needs.

ScaleGrid. (2018b, november 27). *Managing PostgreSQL High Availability Part I: PostgreSQL Automatic Failover*. ScaleGrid. Verkregen 15 april 2021, van https://scalegrid.io/blog/managing-high-availability-in-postgresql-part-1/

Singer, D. (2020, augustus 6). *What is High Availability? A Tutorial*. Liquid Web. Verkregen 2 januari 2021, van https://www.liquidweb.com/kb/what-is-high-availability-a-tutorial/

Technopedia. (2021a). *High Availability (HA): What Does High Availability (HA) Mean?* Technopedia. Verkregen 25 maart 2021, van https://www.techopedia.com/definition/1021/high-availability-ha

Technopedia. (2021b, april 28). *Object-Relational Database (ORD)*. https://www.techopedia.com/definition/8714/object-relational-database-ord

TechTarget. (2017, november 16). *Cluster*. TechTarget. Verkregen 1 april 2021, van https://whatis.techtarget.com/definition/cluster

TechTarget. (2020). *Failback*. TechTarget. Verkregen 1 april 2021, van https://whatis.techtarget.com/definition/failback

Vizard, M. (2020, juli 21). *Puppet Brings Orchestration to IT Automation*. DevOps.com. Verkregen 25 maart 2021, van https://devops.com/puppet-brings-orchestration-to-it-automation/

wiki.postgresql. (2020, december 9). *Repmgr*. wiki.postgresql. Verkregen 13 april 2021, van https://wiki.postgresql.org/wiki/Repmgr#repmgr_5_Features

wiki.postgresql.org. (2020, juli 6). *Streaming Replication*. wiki.postgresql.org. Verkregen 8 april 2021, van https://wiki.postgresql.org/wiki/Streaming_Replication#:~:text=Streaming%20Replication%20(SR)%20provides%20the,some%20out%20of%20date%20information