



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. High Availability lijkt mij iets dat steeds meer en meer relevant wordt. Zeker met de huidige Corona-crisis, is een stabiele online omgeving, cruciaal om competitief te blijven. Dit onderwerp gaf mij de kans om mijzelf 100% te verdiepen in een onderwerp dat mij interesseerde.

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 dichterbij het einde was.

Zeker wil ik mijn ouders bedanken voor de financiële en mentale steun om deze opleiding tot een goed einde te brengen.

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-relationale 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. De tweede gekozen oplossing komt te liggen bij PostgreSQL Automatic Failover (PAF).

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.





# Inhoudsopgave

<b>1</b>	<b>Inleiding</b>	<b>17</b>
1.1	Probleemstelling	17
1.2	Onderzoeksvraag	17
1.2.1	Hoofdonderzoeksvraag	17
1.2.2	Deelonderzoeksvraag	18
1.3	Onderzoeksdoelstelling	18
1.4	Opzet van deze bachelorproef	18
<b>2</b>	<b>Stand van zaken</b>	<b>21</b>
2.1	Databank	21
2.1.1	Relationele Databank	21
2.1.2	Object-georiënteerde databank	22
2.1.3	SQL	22

2.1.4	Object-relacionele databank .....	22
<b>2.2</b>	<b>PostgreSQL</b>	<b>23</b>
<b>2.3</b>	<b>High Availability</b>	<b>24</b>
2.3.1	Load Balancing .....	25
2.3.2	Schaalbaarheid .....	26
2.3.3	Failover .....	26
2.3.4	Failback .....	27
<b>2.4</b>	<b>Cluster oplossingen</b>	<b>27</b>
2.4.1	Cluster .....	27
2.4.2	Standby mogelijkheden .....	27
2.4.3	Patroni .....	28
2.4.4	Pgpool-II .....	28
2.4.5	PostgreSQL Automatic Failover (PAF) .....	30
2.4.6	Replication Manager (repmgr) .....	31
2.4.7	PgBouncer .....	32
2.4.8	Raima .....	32
2.4.9	EDB .....	33
<b>3</b>	<b>Methodologie .....</b>	<b>35</b>
<b>3.1</b>	<b>Requirementsanalyse</b>	<b>36</b>
3.1.1	Functionele Requirements .....	36
3.1.2	Niet-functionele Requirements .....	36
<b>3.2</b>	<b>Indelen requirements volgens MoSCoW-techniek</b>	<b>36</b>

<b>4</b>	<b>Patroni .....</b>	<b>39</b>
<b>4.1</b>	<b>Inleiding tot Patroni</b>	<b>39</b>
<b>4.2</b>	<b>Requirements</b>	<b>39</b>
4.2.1	Must have .....	39
4.2.2	Should have .....	41
4.2.3	Could have .....	41
<b>5</b>	<b>PostgreSQL Automatic Failover (PAF) .....</b>	<b>43</b>
<b>5.1</b>	<b>Inleiding tot PostgreSQL Automatic Failover (PAF)</b>	<b>43</b>
<b>5.2</b>	<b>Requirements</b>	<b>44</b>
5.2.1	Must have .....	44
5.2.2	Should have .....	45
5.2.3	Could have .....	45
<b>6</b>	<b>Pgpool-II .....</b>	<b>47</b>
<b>6.1</b>	<b>Inleiding tot Pgpool-II</b>	<b>47</b>
<b>6.2</b>	<b>Requirements</b>	<b>47</b>
6.2.1	Must have .....	47
6.2.2	Should have .....	48
6.2.3	Could have .....	49
<b>7</b>	<b>Replication Manager (repmgr) .....</b>	<b>51</b>
<b>7.1</b>	<b>Inleiding tot Replication Manager (repmgr)</b>	<b>51</b>
<b>7.2</b>	<b>Requirements</b>	<b>52</b>
7.2.1	Must have .....	52

7.2.2	Should have .....	53
7.2.3	Could have .....	53
<b>8</b>	<b>Verwerking resultaten .....</b>	<b>55</b>
8.1	Resultaten requirements	55
<b>9</b>	<b>Proof of Concept .....</b>	<b>57</b>
9.1	Omgeving	57
9.2	Prerequisites	58
9.2.1	Vagrant .....	58
9.2.2	VirtualBox .....	59
9.3	pgServer	59
9.4	pgNode1	61
9.5	pgNode2	64
9.6	Persoonlijke conclusie	65
<b>10</b>	<b>Conclusie .....</b>	<b>67</b>
<b>A</b>	<b>Onderzoeksvoorstel .....</b>	<b>69</b>
A.1	Samenvatting	69
A.2	Introductie	70
A.3	State of the art	70
A.4	Methodologie	71
A.5	Verwachte resultaten	71
A.6	Verwachte conclusies	72

<b>Bibliografie</b>	<b>73</b>
---------------------	-----------



## Lijst van figuren

2.1	Voorbeeld klasse CD .....	23
2.2	Voorbeeld overerving .....	23





## Lijst van tabellen

4.1	Overzicht score Patroni .....	42
5.1	Overzicht score PostgreSQL Automatic Failover (PAF) .....	46
6.1	Overzicht score Pgpool-II .....	49
7.1	Overzicht score Replication Manager (repmgr) .....	53
8.1	Requirementanalyse alle oplossingen .....	55
9.1	Ip-adressen Cluster .....	57



# 1. Inleiding

## 1.1 Probleemstelling

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.

## 1.2 Onderzoeksvraag

### 1.2.1 Hoofdonderzoeksvraag

Dit onderzoek zal zich richten rond de vraag: Welke PostgreSQL High Availability cluster oplossing kunnen bedrijven, heden ten dage, gebruiken om garantie te hebben op monitoring, redundantie en failover?

### 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: Stand van zaken 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: Methodologie wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen. Hierin zullen 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 aan de hand van deze requirements punten krijgen. De oplossing met de meeste punten zal gebruikt worden bij het opstellen van de proof of concept.

In Hoofdstuk 4: Patroni, Hoofdstuk 5: PostgreSQL Automatic Failover (PAF), Hoofdstuk 6: Pgpool-II en Hoofdstuk 7: Replication Manager (repmgr) worden de vier oplossingen besproken en wordt gekeken of deze oplossingen aan de verschillende requirements voldoen.

In Hoofdstuk 8: Verwerking resultaten zullen de resultaten van de oplossingen, afgetoetst aan de hand van de requirements, verder onderzocht worden. Hieruit wordt dan één oplos-

sing gekozen die gebruikt zal worden voor de opzet van de proof of cluster, te vinden in Hoofdstuk 9: Proof of Concept. 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. Hierbij zal ook een persoonlijke conclusie te vinden zijn over eigen ervaringen bij het opzetten van deze PostgreSQL High Availability cluster.

Tot slot wordt in Hoofdstuk 10: Conclusie 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. De bedoeling van dit hoofdstuk is een inleiding in de verschillende termen en methodes die in dit onderzoek zullen gebruikt worden. Op deze manier wordt u, de lezer, al wat meer vertrouwd met het onderwerp.

### 2.1 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).

#### 2.1.1 Relationale 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).

### 2.1.2 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).

### 2.1.3 SQL

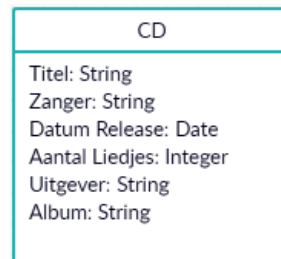
SQL (structured query language) is de eigen programmeertaal specifiek ontwikkelt voor interactie met databanken. Een databank modelleert entiteiten uit het echte leven en slaat deze op in tabellen. Via SQL is het mogelijk om de gegevens in deze tabellen te manipuleren (Carchedi, 2020). SQL wordt door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

### 2.1.4 Object-relationale databank

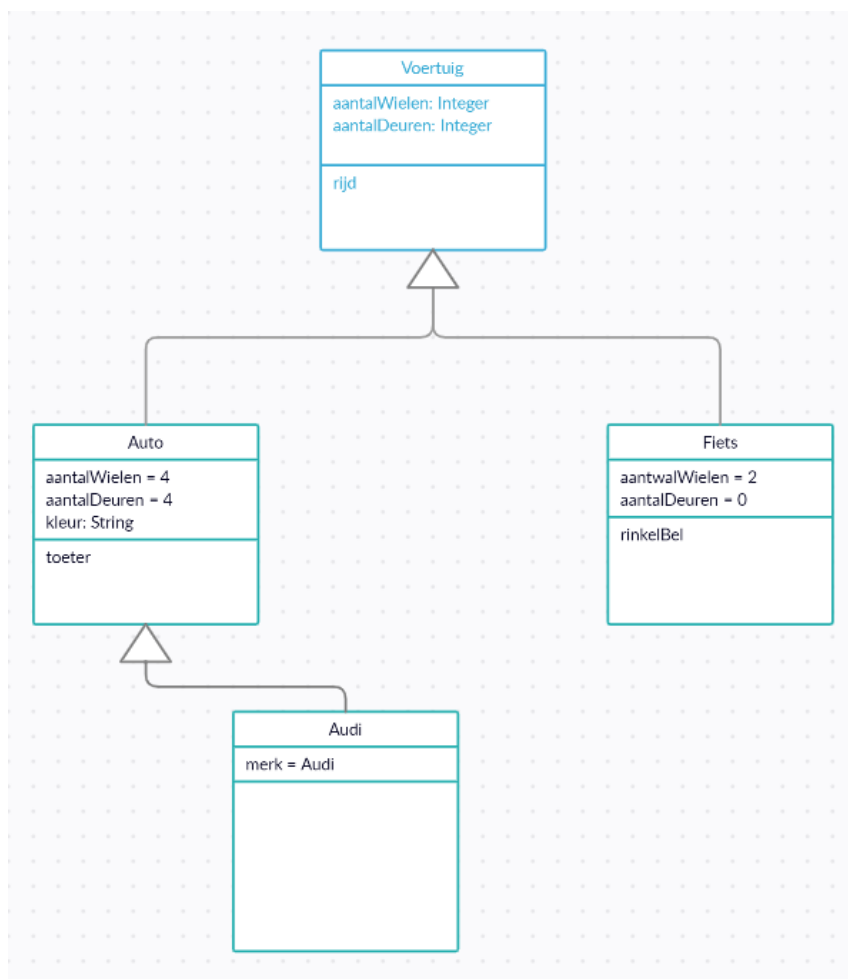
Een object-relationale database (ORD / ORDBMS) is een samenstelling uit zowel een relationele database (RDB / RDBMS), als een object-georiënteerde database (OOD / OODBMS). Samen ondersteunt het de basiscomponenten van elk objectgeoriënteerd databasemodel in zijn schema's en gebruikte querytaal, zoals 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-relationale database, het dichten van de kloof tussen relationele en objectgeoriënteerde modelleringstechnieken en conceptuele datamodelleringstechnieken zoals daar zijn het entiteit-relatiediagram (ERD) en object-relatueel mappen (ORM) (Technopedia, 2021b). Klassen, Overerving, Types, Functies zijn kenmerken van de object-relationale 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. Dit wordt geïllustreerd in onderstaande figuur 2.2: Voorbeeld overerving. Hierbij wordt als voorbeeld een klasse Fiets gebruikt, die overerving doet klasse Vervoermiddel. Hierbij krijgt de klasse Fiets de nodige kenmerken, overgeërfd, van de klasse Vervoermiddel. Hierin is de klasse Fiets een specialisatie van de klasse Vervoermiddel. Zo is een kenmerk van de klasse Vervoermiddel het aantal wielen. Dit datatype wordt dan overgeërfd naar de klasse Fiets, waar de invulling van dit datatype twee is. Bij de klasse Auto heeft het overgeërfde kenmerk vier wielen dan. 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.



Figuur 2.1: Voorbeeld klasse CD



Figuur 2.2: Voorbeeld overerving

## 2.2 PostgreSQL

PostgreSQL is een open source systeem dat zich toelegt op het beheer van object-relationale 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 accommoderen. 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-relationale databank zodat gebruikers het systeem makkelijk kunnen uitbreiden. Deze vier kenmerken, naast nog andere functies maken van Postgres een object-relationale 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).

## 2.3 High Availability

Het doel van High Availability architectuur is ervoor te zorgen dat een server, website of applicatie verschillende soorten netwerkbelastingen 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 ServersAustralia (2017) om in een organisatie maximale productiviteit en betrouwbaarheid te bereiken (ServersAustralia, 2017).

High Availability is het vermogen van een systeem om continu operationeel te blijven gedurende een 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). Maar hier straks wat meer over.

Wanneer is High Availability nu eigenlijk nodig? High Availability is nuttig wanneer bedrijven te maken hebben met dagelijks kritisch applicatiebeheer. Of wanneer er veel traffic 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. **Storingsdetectie:** 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.

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.

### 2.3.1 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 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 garant voor het hebben van High Availability. Door redundantie te implementeren voor de load balancer zelf, kan deze geëlimineerd worden als een single point of failure (Jevtic, 2018))

HAProxy, High Availability Proxy, is hier een voorbeeld van. Dit is een op software gebaseerde TCP/HTTP load balancer. Het verdeelt de werklast over meerdere servers zodat het prestaties kan maximaliseren en het gebruik van resources kan optimaliseren (SeveralNines, 2020).

### 2.3.2 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 schrijfpresetaties 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).

### 2.3.3 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).

### 2.3.4 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).

## 2.4 Cluster oplossingen

### 2.4.1 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).

### 2.4.2 Standby mogelijkheden

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 (PostgreSQL, 2021e). Een Cold Standby systeem is geen operationeel systeem, maar het kan wel beschikbaar worden gemaakt wanneer dit nodig geacht wordt. 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 gearcheiverde logbestanden of door gebruik te maken van log shipping replicatie van Postgres. Log shipping is een proces waarbij de back-up van transactielogbestanden op een primaire database wordt geautomatiseerd en vervolgens op een standby server wordt hersteld (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 gearcheiverde 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.

### 2.4.3 Patroni

Patroni is een open source cluster-technologie, geschreven in Python die zich focust op 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 wordt.

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

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

### 2.4.4 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 replicatie boven streaming replicatie is dat logical toelaat om veranderingen van een oudere versie van PostgreSQL naar een nieuwere versie te repliceren. Streaming replicatie werkt alleen wanneer zowel de master- als de standby node 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 veel gebruikers zijn die veel queries tegelijkertijd uitvoeren.

### 4. Limiting Exceeding Connections

Er is een limiet op het maximum aantal gelijktijdige verbindingen met PostgreSQL. Verbindingen die gemaakt willen worden nadat dit maximum aantal verbindingen overschreden is, worden geweigerd. 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 hier niet geweigerd, maar worden in een wachtrij geplaatst, zodat wanneer een verbinding verdwijnt, de volgende zijn plaats kan innemen.

### 5. Watchdog

Watchdog kan een robuust clustersysteem creëren en het single point of failure of split brain vermijden. Watchdog kan worden ingesteld op de nodes en kan zo een lifecheck uitvoeren op de Pgpool-II nodes, om dan op deze manier fouten in Pgpool-II te detecteren. Wanneer een actieve Pgpool-II node down gaat, kan dan een standby Pgpool-II node 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. Wanneer 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).

### 2.4.5 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:

- aan detectiestoring te doen van een PostgreSQL instance.
- de primary server te herstellen...
- of om aan failover te doen naar een standby server.
- om de best (met de kleinste vertraging) beschikbaarste standby server te selecteren bij failover.
- 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 de cluster 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 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 automatisch terug wilt toevoegen aan de cluster. 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).

### 2.4.6 Replication Manager (repmgr)

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:

#### De implementatie als een PostgreSQL extentie

##### Replicatie cluster monitoring

##### 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 cliënten, 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).

##### Promotie standby server

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).

### 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.

### 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).

### Switchover ondersteuning voor rolswitching tussen primary en standby

Hierin wordt een standby server geüpgevormd 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).

## 2.4.7 PgBouncer

PgBouncer is een open source connection-pooler die gebruikt wordt voor PostgreSQL. Het kan verbindingen naar één of meerdere databases poolen. Het onderhoudt voor een pool van verbindingen per gebruiker. Het wordt vaak geconfigureerd om één van de verbindingen uit te delen aan een nieuwe inkomende cliëntverbinding en terug te geven aan de pool wanneer de cliënt de verbinding verbreekt (Ramachandran, 2019).

## 2.4.8 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).

### 2.4.9 EDB

Voor betrouwbare cross-over biedt EnterpriseDB (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 categorieën 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 twee afzonderlijke datasets met overlap (Kumar, 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 geschikt 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 requirements zullen dan aan de hand van de MoSCoW-methode geprioriteerd worden om zo het belang van elke requirement in te schatten. Deze analyse zal van groot belang zijn in dit onderzoek aangezien ze de basis zullen vormen waarop er een High Available PostgreSQL cluster oplossing zullen beoordelen en kiezen.

Na de verwerking van de resultaten zal er in dit onderzoek één oplossing gebruikt worden waar een proof of concept mee gemaakt zal worden. Hier zal dan gekeken worden om de verschillende functionele requirements te implementeren in de cluster. Op deze manier zal dan worden aangetoond dat deze oplossing een volwaardig PostgreSQL High Availability cluster oplossing is voor bedrijven om te gebruiken.

## 3.1 Requirementsanalyse

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).

### 3.1.1 Functionele Requirements

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

- Replicatie
- Failover
- Monitoring

### 3.1.2 Niet-functionele Requirements

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

- Open source
- Actieve ondersteuning in 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.
- Beperkte manuele interventie  
Een cluster opzetten die nadien niet veel manuele interventie nodig heeft, is ideaal in situaties waar er weinig tot geen kennis is over de cluster. Tijdens en na downtime zal de server automatisch acties uitvoeren die het verdere bestaan van de cluster garandeert.

## 3.2 Indelen requirements volgens MoSCoW-techniek

Na het opstellen van de verschillende requirements kunnen deze nog eens ingedeeld worden 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 wordt hier achterwege gelaten, omdat dit geen invloed zal hebben op de keuze van een oplossing.

- **Must have**

- Ondersteuning van replicatie
- Ondersteuning van failover
- Ondersteuning van monitoring

- **Should have**

- Actieve ondersteuning in 2020-2021
- Open source

- **Could have**

- Grafische interface
- Beperkte manuele interventie





## 4. Patroni

In dit hoofdstuk wordt meer informatie verschaft over Patroni als een PostgreSQL High Availability cluster oplossing. Er zal worden ingegaan op de verschillende requirements, waaraan voldaan moet worden. Hierbij wordt dan telkens ook een score gegeven die in Hoofdstuk 8: Verwerking resultaten verder geanalyseerd zullen worden.

### 4.1 Inleiding tot Patroni

Patroni, als oplossing, is ontstaan uit een fork van Governor, een project van Compose. Het is een in Python geschreven open source oplossing. Het beheert High Availability van PostgreSQL clusters

### 4.2 Requirements

#### 4.2.1 Must have

##### Replicatie

Voor het kopiëren van bestanden of databases, of replicatie genaamd maakt Patroni gebruik 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 gecommiteerde 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 secondary 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 secondary 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.

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.

Hieruit blijkt dat Patroni zeer flexibel en eenvoudig te gebruiken is bij het configureren van replicatie in een High Availability cluster. Op basis hiervan krijgt deze requirement een score van 4.

## 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.

Op deze requirement doet Patroni het zeer goed. Door de hierboven genoemde redenen krijgt Patroni voor failover een 4.

## 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.

Ook op deze requirement scoort Patroni redelijk goed. Er is een uitgebreid aanbod beschikbaar om monitoring te voorzien in de cluster. Hiervoor krijgt Patroni opnieuw een 4.

### 4.2.2 Should have

#### Actieve ondersteuning in 2020-2021

Er worden nog steeds releases gedaan. Op het moment van schrijven was de laatste release op 22 februari 2021. Deze release bevatte nieuwe features waaronder toegevoegde support voor de REST API bij TLS keys en cipher suite limitations. Deze release bevatte ook nog stabiliteitsverbetering en bugfixes.

Op basis hiervan krijgt Patroni voor deze requirement een score van 5.

#### Open source

Patroni is volledig open source. De source code is online te vinden op [github.com](https://github.com) en wordt nog tot op vandaag geüpdate.

Patroni is een open source oplossing, dus voldoet zeker ook aan deze requirement. Hierbij krijgt het ook een score van 5.

### 4.2.3 Could have

#### Grafische interface

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 er op Windows eenvoudig, snel en gebruikersvriendelijk een Patroni cluster opzetten.

**Beperkte manuele interventie**

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

Dit was de laatste requirement waarop Patroni geëvalueerd zou worden. In onderstaande tabel 4.1 is het overzicht terug te vinden van de scores per requirement. Dit zal enkel gaan over de requirements **must have** en **should have**. De could have requirements hebben geen invloed op de eindscore.

Overzicht score Patroni	
<b>Must have</b>	
Ondersteuning van replicatie	4
Ondersteuning van failover	4
Ondersteuning van monitoring	4
<b>Should have</b>	
Open Source	5
Ondersteuning in 2020-2021	5
Totaal	22/25

Tabel 4.1: Overzicht score Patroni

## 5. PostgreSQL Automatic Failover (PAF)

In dit hoofdstuk wordt meer informatie verschaft over PostgreSQL Automatic Failover (PAF) als een PostgreSQL High Availability cluster oplossing. Er zal worden ingegaan op de verschillende requirements, waaraan voldaan moet worden. Hierbij wordt dan telkens ook een score gegeven die in Hoofdstuk 8: Verwerking resultaten verder geanalyseerd zullen worden.

### 5.1 Inleiding tot PostgreSQL Automatic Failover (PAF)

PostgreSQL Automatic Failover (PAF) is een High Availability oplossing ontwikkeld door ClusterLabs die zich bezighoudt met het beheer van High Availability in PostgreSQL clusters. Het maakt vooral gebruik van Postgres synchrone replicatie om te garanderen dat er geen gegevens verloren gaan op het moment van de failover operatie.

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

Pacemaker is een service die in staat is om veel resources te beheren, en doet dit met de hulp van hun resource agents. Resource agents hebben dan de verantwoordelijkheid om een specifieke resource af te handelen, hoe ze zich moeten gedragen, en Pacemaker te informeren over deze resultaten.

## 5.2 Requirements

### 5.2.1 Must have

#### Replicatie

PostgreSQL Automatic Failover (PAF) zal niet 100% kunnen beschermen tegen gegevensverlies. De replicatie wordt geconfigureerd met PostgreSQL. Bij asynchrone replicatie, wat de standaard is voor PostgreSQL, zullen de transacties eerst op primary node gecommitteerd worden voordat ze op de standby nodes zullen worden toegepast. In het geval van failover, zal de meest up-to-date standby node gepromoot worden door PostgreSQL Automatic Failover (PAF). Hierdoor zal gegevensverlies geminimaliseerd worden, maar er zal nog iets van verlies zijn.

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

Hierdoor krijgt PostgreSQL Automatic Failover (PAF) voor deze requirement een score van 4.

#### Failover

Aan de hand van Pacemaker kan er automatische failover plaatsvinden. PostgreSQL Automatic Failover (PAF) communiceert continu met Pacemaker over de status van de cluster en bewaakt het functioneren van de database. In geval van storing, wordt Pacemaker hierover geïnformeerd en zal het kijken om de storing op te lossen. Is de storing onoplosbaar, dan zal Pacemaker een verkiezing starten tussen de standby nodes om een opvolger van de primary node te selecteren. Bij het configureren van de cluster wordt elke node geconfigureerd als standby node voor Pacemaker een keuze maakt van welke node de primay wordt. Dit zorgt er ook voor dat elke node weet hoe hij moet functioneren als een standby node.

'master-max' krijgt bij de configuratie van Pacemaker het aantal PostgreSQL nodes dat als primary zullen dienen op een gegeven moment. Default staat deze ingesteld op 1.

Bij 'clone-max' staat de ingestelde parameter gelijk aan het aantal nodes dat PostgreSQL kunnen draaien, primary of standby. Deze parameter staat normaal gelijk aan het aantal nodes die aanwezig zijn in de cluster.

'notify=true' laat toe om te signaleren wanneer er bepaalde acties gebeuren op de node. Dit zal in contact staan met Pacemaker. Deze staat standaard op inactief, dus moet de parameter liefst expliciet nog eens vermeld worden op 'true'.

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.

Door de hierboven genoemde redenen krijgt PostgreSQL Automatic Failover (PAF) een score van 4 voor deze requirement.

### Monitoring

Via de tool Pacemaker kan er bij PostgreSQL Automatic Failover (PAF) goed aan monitoring gedaan. Het zal signaleren wanneer een node niet bereikbaar is. Het toevoegen van Watchdog als tool is hier mogelijk om ook aan monitoring te doen.

Hierdoor krijgt PostgreSQL Automatic Failover (PAF) voor monitoring een score van 4.

## 5.2.2 Should have

### Actieve ondersteuning in 2020-2021

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.

Hier krijgt PostgreSQL Automatic Failover (PAF) een score van 4. Een score van 5 ging behaald worden moest de laatste release in 2021 zijn.

### Open source

PostgreSQL Automatic Failover (PAF) is een open source High Availability oplossing voor PostgreSQL clusters.

Voor deze requirement krijgt PostgreSQL Automatic Failover (PAF) een score van 5.

## 5.2.3 Could have

### Grafische interface

Op eerste zicht is er geen algemeen gekende grafische interface aanwezig te zijn voor PostgreSQL Automatic Failover (PAF).

### Beperkte manuele interventie

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

Dit was de laatste requirement waarop PostgreSQL Automatic Failover (PAF) geëvalueerd zou worden. In onderstaande tabel 5.1 is het overzicht terug te vinden van de scores per requirement. Dit zal enkel gaan over de requirements **must have** en **should have**. De could have requirements hebben geen invloed op de eindscore.

Overzicht score PostgreSQL Automatic Failover (PAF)	
Must have	
Ondersteuning van replicatie	4
Ondersteuning van failover	4
Ondersteuning van monitoring	4
Should have	
Open Source	5
Ondersteuning in 2020-2021	4
Totaal	21/25

Tabel 5.1: Overzicht score PostgreSQL Automatic Failover (PAF)



## 6. Pgpool-II

In dit hoofdstuk wordt meer informatie verschaft over Pgpool-II als een PostgreSQL High Availability cluster oplossing. Er zal worden ingegaan op de verschillende requirements, waaraan voldaan moet worden. Hierbij wordt dan telkens ook een score gegeven die in Hoofdstuk 8: Verwerking resultaten verder geanalyseerd zullen worden.

### 6.1 Inleiding tot Pgpool-II

### 6.2 Requirements

#### 6.2.1 Must have

##### **Replicatie**

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.

Standaard voor PostgreSQL zal er gebruik worden gemaakt van streaming replication bij Pgpool-II.

Voor deze redenen krijgt Pgpool-II een score van 3 voor replicatie.

## Failover

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_exists`.

Bij Pgpool-II, wanneer het `failover_command` geconfigureerd is, zal deze automatisch tijdens een failover worden uitgevoerd. Hierin zal een nieuwe primary node uit de bestaande standby nodes gekozen worden. Ook kan dit commando signalen uitsturen, zoals een mail om te laten weten dat er failover plaatsvindt binnen de cluster.

Pgpool-II krijgt voor deze requirement een score van 4.

## Monitoring

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

Voor monitoring krijgt Pgpool-II een score van 3.

### 6.2.2 Should have

#### Actieve ondersteuning in 2020-2021

De laatste release van pgpool-II was op 26 november 2020. Hieruit kan worden besloten dat Pgpool-II nog een geldige, bruikbare oplossing is.

Voor deze requirement krijgt Pgpool-II een score van 4. Een score van 5 zou zijn wanneer Pgpool-II in 2021 een release zou gehad hebben.

## Open source

Pgpool-II is een open source project. De source code wordt onderhouden door hun git-repository.

Pgpool-II krijgt voor deze requirement een score van 5.

### 6.2.3 Could have

#### Grafische interface

Op eerste zicht is geen algemeen gekende grafische interface aanwezig voor Pgpool-II.

#### Beperkte manuele interventie

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.

Dit was de laatste requirement waarop Pgpool-II geëvalueerd zou worden. In onderstaande tabel 6.1 is het overzicht terug te vinden van de scores per requirement. Dit zal enkel gaan over de requirements **must have** en **should have**. De could have requirements hebben geen invloed op de eindscore.

Overzicht score Pgpool-II	
<b>Must have</b>	
Ondersteuning van replicatie	3
Ondersteuning van failover	4
Ondersteuning van monitoring	3
<b>Should have</b>	
Open Source	5
Ondersteuning in 2020-2021	4
Totaal	19/25

Tabel 6.1: Overzicht score Pgpool-II



## 7. Replication Manager (repmgr)

In dit hoofdstuk wordt meer informatie verschaft over Replication Manager (repmgr) als een PostgreSQL High Availability cluster oplossing. Er zal worden ingegaan op de verschillende requirements, waaraan voldaan moet worden. Hierbij wordt dan telkens ook een score gegeven die in Hoofdstuk 8: Verwerking resultaten verder geanalyseerd zullen worden.

### 7.1 Inleiding tot Replication Manager (repmgr)

Replication Manager (repmgr) is een open source oplossing ontwikkeld door 2ndQuadrant. Het is ontwikkeld voor het beheer van replicatie en failover in een PostgreSQL cluster.

Replication Manager (repmgr) biedt de tools om PostgreSQL replicatie op te zetten, te configureren, te beheren en te monitoren. Het stelt de gebruiker in staat om handmatige omschakeling en failover taken uit te voeren met behulp van repmgr utility. Dit is een open source tool die PostgreSQL's ingebouwde streaming replicatie ondersteunt en verbetert.

## 7.2 Requirements

### 7.2.1 Must have

#### Replicatie

Replication Manager (RepMgr) maakt standaard gebruik van streaming replicatie in zijn clusters.

Deze streaming replicatie moet vaak manueel geconfigureerd worden in zowel de primary als de standby node. Het zal ook gebruik maken van `pg_basebackup` zodat alle standby nodes kunnen invallen wanneer de primary node uitvalt.

Met repmgr is het mogelijk om deze taken geautomatiseerd uit te voeren, wat veel tijd en mankracht kan sparen.

Voor replicatie krijgt Replication Manager (repmgr) een score van 4.

#### Failover

Aan de hand van de tool repmgrd zal er automatische failover in gang worden gezet wanneer de primary node op down komt te staan. Hier wordt er dan een nieuwe standby node gekozen die komt invallen voor de voormalige primary node.

Het zal in geval van failover of andere situaties ook de nodige notificaties uitsturen om te laten weten wat er aan de hand is. Deze situaties worden dan door de gebruiker zelf geregistreerd.

Om repmgrd te laten werken zal in het repmgr.conf bestand, bij failover 'automatic' worden ingegeven. Hierna zal de repmgrd daemon de cluster actief beginnen te monitoren. Als er een fout optreedt in de primary node, zal het meerdere keren proberen opnieuw verbinding te maken. Als alle pogingen om verbinding te maken met de primary node mislukken, wordt de meest geschikte standby gekozen als de nieuwe primary node door repmgrd.

Door de hierboven genoemde redenen, krijgt Replication Manager (repmgr) een score van 4 voor failover.

#### Monitoring

Over Monitoring is er bij Replication Manager (repmgr) niet veel te vinden. De tools repmgrd zal wel actief de cluster monitoren, en op basis hiervan de nodige acties uitvoeren, naargelang de staat van de cluster.

Voor monitoring krijgt Replication Manager (repmgr) een score van 3.

### 7.2.2 Should have

#### Actieve ondersteuning in 2020-2021

##### Open source

Replication Manager (repmgr) is een open source High Availability tool voor PostgreSQL clustering.

### 7.2.3 Could have

#### Grafische interface

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

#### Beperkte manuele interventie

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

Dit was de laatste requirement waarop Replication Manager (repmgr) geëvalueerd zou worden. In onderstaande tabel 7.1 is het overzicht terug te vinden van de scores per requirement. Dit zal enkel gaan over de requirements **must have** en **should have**. De could have requirements hebben geen invloed op de eindscore.

Overzicht score Replication Manager (repmgr)	
<b>Must have</b>	
Ondersteuning van replicatie	4
Ondersteuning van failover	4
Ondersteuning van monitoring	3
<b>Should have</b>	
Open Source	5
Ondersteuning in 2020-2021	4
Totaal	20/25

Tabel 7.1: Overzicht score Replication Manager (repmgr)





## 8. Verwerking resultaten

In dit hoofdstuk zullen de verschillende resultaten van de requirementanalyse van de oplossingen verwerkt worden om zo tot een statistisch correct antwoord te komen op de hoofdonderzoeksvraag uit Sectie 1.2.1.

### 8.1 Resultaten requirements

Hieronder

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

Tabel 8.1: Requirementanalyse alle oplossingen

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.



## 9. Proof of Concept

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

### 9.1 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 geconfigureerd worden dat er één primary node is met een standby node die asynchrone streaming replicatie verricht.

Ip-adressen	
pgServer	172.16.0.11
pgNode1	172.16.0.22
pgNode2	172.16.0.33

Tabel 9.1: Ip-adressen Cluster

## 9.2 Prerequisites

### 9.2.1 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 bepaald 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 bestand meegegeven die zal worden uitgevoerd in de server. Dit bestand 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
```

---

Bij het gebruik van Vagrant, komen er ook een paar vaak voorkomende commando's bij kijken. Zo is 'vagrant up' het commando dat gebruikt wordt om alle virtuele machines

aan te zetten. Wanneer er één server alleen moet worden aangezet, kan er ook specifiek een virtuele machine aangeroepen worden: `'vagrant up pgnode1'`. Dit zal enkel de virtuele machine met als naam `pgnode1` opstarten. Een volgend veelgebruikt commando is `'vagrant provision'`. Dit commando zal worden gebruikt wanneer er in de configuratiebestanden wijzigingen zijn toegebracht. Dit laat toe om deze wijzigingen uit te voeren op alle virtuele machines. Ook hier kan er specifiek één virtuele machine aangeroepen worden, dit is net zoals hierboven, door een specifieke naam mee te geven als parameter. `'vagrant halt'` zal alle virtuele machines uitzetten. `'vagrant ssh pgnode1'` zal ssh-connectie maken met de meegegeven virtuele machine als parameter.

Na het opzetten van onze Vagrantfile wordt deze via `'vagrant up'` geactiveerd in een command line. Dit kan in Windows Powershell, maar ook in een Linux terminal of een andere soort terminal.

Vagrant is niet standaard geïnstalleerd op een Ubuntu machine. Via `'sudo apt-get install vagrant'` wordt Vagrant geïnstalleerd.

### 9.2.2 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 besturingssystemen kan emuleren. Bij de configuratie van een virtuele machine kunnen de specificaties zoals RAM-geheugen, schijfruimte, etc. bepaald worden.

## 9.3 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 zal 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 wordt verwezen naar het config bestand 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 configuratiebestand 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

```

---

Het pgBouncer.ini bestand 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
```

---

## 9.4 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 wordt verwezen naar het config bestand 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"
}
```

---

## 9.5 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 wordt verwezen naar het config bestand 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"
}
```

---

## 9.6 Persoonlijke conclusie



## 10. Conclusie

In dit onderzoek wordt een antwoord gegeven op de onderzoeksvraag: Welke PostgreSQL High Availability cluster oplossing kunnen bedrijven, de dag van vandaag, gebruiken om garantie te hebben op monitoring, redundantie en failover? Om hierop een antwoord te vinden, is een vergelijkende studie uitgevoerd tussen de verschillende oplossingen uit dit onderzoek. Hier werd dan elke oplossing afgetoetst aan de requirements.

Uit de resultaten van de requirementsanalyse blijkt dat Patroni momenteel de PostgreSQL High Availability cluster oplossing is, die het meest aansluit aan de vooropgestelde requirements. De proof of concept beaamt ook deze vaststelling. Het gebruik van Patroni geeft een garantie op High Availability in een PostgreSQL cluster. Patroni zelf is op het vlak van replicatie zeer flexibel. Het werkt standaard met asynchrone replicatie, maar kan ook zo geconfigureerd worden dat het synchroon gebeurt. Wat betreft failover is Patroni ook zeer ver gevorderd. Er zijn veel geavanceerde opties voor het configureren van failover in de cluster. Een handig iets is dat failover automatisch kan gebeuren en dus geen nood hoeft te hebben aan manuele interventies. Inzake monitoring kent Patroni een zeer rijke REST API die kan gebruikt worden om de cluster en de nodes te monitoren. Hierin kunnen persoonlijke voorkeuren van monitoring toegevoegd worden.

Ook PostgreSQL Automatic Failover (PAF) is een goed alternatief bij het opzetten van een PostgreSQL High Availability cluster.

Uitleg PAF: nog te schrijven

Uitleg andere tools: nog te schrijven

Bij de aanvang van dit onderzoek had ik deze uitslag nog niet verwacht. Dit omdat ik nog geen weet had van welke oplossingen er allemaal waren. Het was pas na het lezen

van Akhtar, H dat ik een idee kreeg van welke PostgreSQL oplossingen er allemaal aanwezig waren. Akhtar haalde Patroni, PostgreSQL Automatic Failover (PAF), pgPool-II en Replication Manager (repmgr) aan als gepaste oplossingen voor High Availability in een PostgreSQL omgeving. Na het schrijven van mijn voorstel en literatuurstudie had ik wel het gevoel dat Patroni een zeer goede kandidaat ging zijn als PostgreSQL High Availability oplossing. De requirementanalyse heeft dit gevoel bij mij dan ook bevestigd en verantwoord.

Dit onderzoek biedt gegarandeerd een meerwaarde voor bedrijven die zich willen inwerken in PostgreSQL High Availability. Er wordt hier een duidelijk beeld gegeven van de verschillende oplossingen en de manier waarop deze oplossingen voldoen aan bepaalde requirements. Deze requirements zijn gekomen uit de bedrijfswereld en zullen zeker voldoen aan de noden van andere bedrijven.

Dit onderzoek vraagt zeker naar een vervolgonderzoek voor de toekomst. Hoe wordt High Availability binnen tien jaar geïmplementeerd in PostgreSQL clusters. Welke nieuwe noden in een cluster zijn er dan van belang.

# 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 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 back-upoplossingen + 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-relacioneel 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<sub>MoSCoW</sub> : A fuzzybasedMoSCoW 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\_archivecleanup*. Percona. Verkregen 13 april 2021, van [https://www.percona.com/blog/2019/07/10/wal-retention-and-clean-up-pg\\_archivecleanup/](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\\_](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-DBhC7ARIsAI8YcWJOxNwWfRDfjloPnAcsWV6QsafwgHI0hqa-gQFH5fj59JLPM04NOxMaAmudEALw\_wcB
- CDNF. (2020, juli 22). *What is a CNF?* cdnf. Verkregen 8 april 2021, van [https://cdnf.io/what\\_is\\_cnf/](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](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](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>
- 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 POSTGRES SQL 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](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). *pgbasebackup*. PostgreSQL. Verkregen 22 april 2021, van <https://www.postgresql.org/docs/10/app-pgbasebackup.html>
- PostgreSQL. (2021c). *pgrewind*. 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. (2021e). *Write-Ahead Logging (WAL)*. PostgreSQL. Verkregen 12 mei 2021, van <https://www.postgresql.org/docs/9.0/wal-intro.html>
- 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/>
- Ramachandran, M. (2019, maart 19). *PostgreSQL Connection Pooling With PgBouncer: Learn more about using PgBouncer to pool PostgreSQL connections*. Database Zone. Verkregen 12 mei 2021, van <https://dzone.com/articles/postgresql-connection-pooling-with-pgbouncer#:~:text=PgBouncer%20is%20an%20open-source,each%20unique%20user,%20database%20pair>
- 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/>

- Roth, P. M. M. R. T. S. C. G. J. (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>
- 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/>
- ServersAustralia. (2017, oktober 31). *What is High Availability (HA) and Do I Need It?* Servers Australia. Verkregen 25 maart 2021, van <https://www.serversaustralia.com.au/resources/blog/what-is-high-availability-ha-and-do-i-need-it/#:~:text=The%20purpose%20of%20HA%20architecture,achieve%20maximum%20productivity%20and%20reliability>
- SeveralNines. (2020). *PostgreSQL Load Balancing with HAProxy: 2. What is HAProxy?* SeveralNines. Verkregen 12 mei 2021, van <https://severalnines.com/resources/database-management-tutorials/postgresql-load-balancing-haproxy>
- 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>
- wiki.postgresql. (2020, december 9). *Repmgr*. wiki.postgresql. Verkregen 13 april 2021, van [https://wiki.postgresql.org/wiki/Repmgr#repmgr\\_5\\_Features](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](https://wiki.postgresql.org/wiki/Streaming_Replication#:~:text=Streaming%20Replication%20(SR)%20provides%20the,some%20out%20of%20date%20information)