Afstudeeropdracht Topicus

Afstudeerverslag

Klas:
DHI4V.AADa1

Opleiding:
Software Engineering(SE)

Opdrachtgever:
Topicus Education BV

Student(en):
Rob van Heuven (467542)

Datum:
10 juni 2022

Versie:
V1.0

Versie	Datum	Omschrijving
V0.1	25-03-2022	Eerste opzet.
V0.2	24-05-2022	Feedback is verwerkt van mijn bedrijfsbegeleiders en afstudeerdocent.
V1.0	10-06-2022	Feedback is verwerkt van mijn examinatoren.

Inhoudsopgave

1. Samenvatting

Tijdens mijn afstudeertraject heb ik een opdracht uitgevoerd voor Topicus.Education B.V. Voor deze opdracht heb ik onderzocht hoe Topicus voor Somtoday, een online leerling-informatiesysteem en elektronische leeromgeving voor het voortgezet onderwijs, de productieomgeving kan aanpassen zonder dat de gebruikers daar last van hebben.

Momenteel worden bij productie onderhoud aan Somtoday alle gebruikers uitgelogd. Om de dienstverlening te verbeteren naar haar klanten wil Somtoday de productieomgeving kunnen onderhouden terwijl de gebruikers door kunnen werken zonder uitgelogd te worden.

Omdat alle sessie-data in het geheugen van de applicatie en op de applicatieserver wordt vastgehouden gaan deze verloren bij een herstart. Als dit behouden kan worden hoeven gebruikers van Somtoday hierbij niet uitgelogd te worden.

Topicus Education B.V. verwacht een Proof of Concept oftewel PoC, die als basis gebruikt kan worden voor het bewerken van Somtoday. Dit PoC moet kunnen voldoen aan de essentiële eisen gesteld in Requirements - Afstudeeropdracht Topicus[2]. Met dit PoC kan er dan onderzocht worden hoe deze oplossing toegepast kan worden in Somtoday.

Voor deze opdracht heb ik een Onderzoeksrapport[4] opgezet waarin ik aan de hand van de Advanced Application Development onderzoeksmethode en de DOT onderzoeksmethode[7] een Longlist heb opgesteld. Deze bestaat uit meerdere technologieën waarmee het verlies van sessiegegevens verholpen kan worden. Deze technologieën zijn vervolgens kort onderzocht of ze wel echt geschikt zijn. Hieruit is een Shortlist samengesteld met technologieën die verder onderzocht zijn en aan de hand van een aantal criteria becijfert zijn.

Hierna wordt er met de 2 best scorende technologieën een PoC ontwikkeld. Vervolgens is aan de hand van de PoC’s bepaald welke technologie het beste geschikt is.

De gekozen PoC is daarna verder doorontwikkeld om ook te werken met de door Somtoday gebruikte WildFly applicatieserver waarvoor ook een apart onderzoek is opgezet.

Na dat dit PoC was opgeleverd is een Technisch ontwerp[9] geschreven over hoe de oplossingen uit het PoC toegepast zou kunnen worden op Somtoday. En dit ontwerp is vervolgens toegepast op Somtoday zelf.

2. Introductie

De opleiding HBO-ICT Software Engineering wordt afgerond met een afstudeerstage. De duur hiervan is ongeveer 5 maanden, van 7 februari 2022 tot 1 juli 2022.

Ik deed mijn afstudeerstage bij Topicus.Education B.V. in Deventer. Topicus is een bedrijf dat zich op het gebied van software bevindt in meerdere sectoren. Hieronder vallen zorg, finance, onderwijs en sociale domeinen. Topicus is gevestigd in Deventer. De vestiging waar ik mijn stage doorbreng heeft ongeveer 800 werknemers.

2.1 Inleiding

Voor mijn afstudeerproject voor de studie HBO-ICT Software Engineering aan de Saxion Hogeschool in Deventer onderzocht ik de volgende vraag voor Topicus.Education B.V., de exploitant van Somtoday:

- Hoe kan Somtoday, een online leerling-informatiesysteem en elektronische leeromgeving voor het voortgezet onderwijs, de productieomgeving aanpassen zonder dat de gebruikers daar last van hebben?

Om deze onderzoeksvraag voor Topicus te beantwoord is de volgende hoofdvraag opgesteld:

- Hoe kan het beste de sessie-data los worden getrokken van de individuele Kubernetes pods voor de applicatie Somtoday ELO en Somtoday Core.

3. Probleemstelling

Op dit moment worden bij productie onderhoud aan Somtoday alle gebruikers uitgelogd, ook bij kortdurend onderhoud zoals het toevoegen van extra verwerkingscapaciteit. Om de dienstverlening naar haar klanten te verbeteren wil Somtoday de productieomgeving kunnen onderhouden terwijl de gebruikers door kunnen werken zonder uitgelogd te worden.

Om de productieomgeving van Somtoday te onderhouden moet deze herstart worden. Het probleem hiermee is, is dat de gebruikersgegevens oftewel de sessie-data in het geheugen van de applicatie server wordt vastgehouden. En dit gaat verloren bij een crash of herstart. Als deze sessie-data behouden kan worden hoeven gebruikers van Somtoday hierbij niet uitgelogd te worden.

3.1 Huidige situatie

Tijdens mijn afstudeerstage werk ik aan Somtoday. Somtoday is opgebouwd uit verschillende applicaties die met elkaar samenwerken. Somtoday Core is de kern van het leerling administratiesysteem en Somtoday ELO is specifiek bedoeld om gebruikt te worden door leerlingen en ouders. Deze beiden applicaties maken gebruik van het Apache Wicket framework.

Deze onderdelen draaien in Kubernetes met verschillende instanties per applicatie-onderdeel. Zo een instantie noem je een pod.

3.2 Verwachting

Topicus.Education B.V. verwacht van dit afstudeerproject een Proof of Concept oftewel PoC, die als basis gebruikt kan worden voor het bewerken van de productie versie van Somtoday. Dit PoC moet kunnen voldoen aan de essentiële eisen gesteld in het, Requirements - Afstudeeropdracht Topicus[2] document. Als dit PoC is opgeleverd kan er ook onderzocht worden hoe dezelfde oplossing die is toegepast in het PoC ook toegepast zou kunnen worden in Somtoday.

4. Schets Somtoday

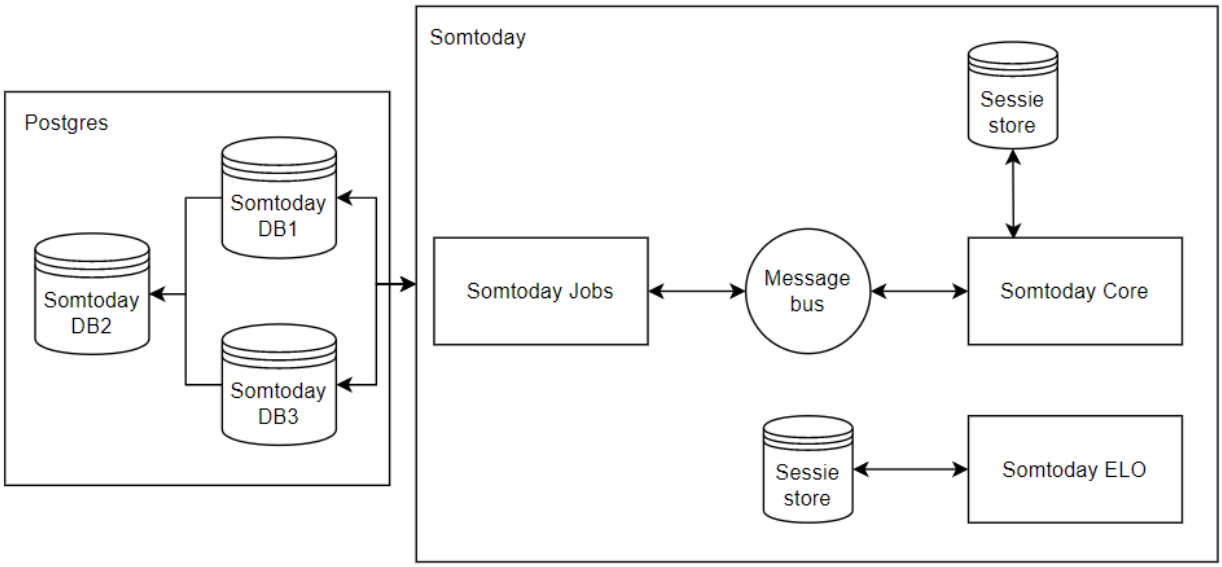
Somtoday is voor administreren, organiseren en leren. Het heeft in Nederland een marktaandeel van 35% waarbij het achterloopt op Magister, ook een elektronische leeromgeving met het grootste gedeelte van het resterende marktaandeel. Verder wordt Somtoday gebruikt door ongeveer 300.000 leerlingen en 30.000 docenten.

Somtoday bestaat uit een dynamisch landschap dat is opgebouwd uit verschillende applicaties die met elkaar samenwerken. Dit is dynamisch omdat hier steeds nieuwe services en applicaties aan worden toegevoegd.

Verder maakt Somtoday gebruik van Kubernetes. Kubernetes is een open-source systeem voor het automatiseren van implementatie, schaling en beheer van gecontaineriseerde applicaties [22]. Een draaiende instantie van een applicatie heet een pod. In het geval van Somtoday heeft elke instantie van de applicatie een eigen pod. En een applicatie kan meerdere pods bevatten.

4.1 Infrastructuur diagram Somtoday (Core en ELO) huidige situatie:

Somtoday bevat meer onderdelen dan hier weergegeven zijn. Deze onderdelen hebben echter weinig tot geen belang bij deze opdracht.



4.2 Somtoday ELO

Somtoday ELO is een Wicket applicatie uit het Somtoday landschap. En staat voor elektronische leeromgeving. Hierin worden onder andere de cijfers, roosters en persoonlijke informatie van leerlingen beheerd. Somtoday ELO is specifiek bedoeld om gebruikt te worden door leerlingen en ouders. Dit is een opslag van de Wicket SessionStore die lokaal op de applicatie pod in kubernetes wordt bijgehouden.[23] Dit is tijdens mijn afstudeerperiode toegevoegd om snelheden te verbeteren.

4.3 Somtoday Core

Somtoday Core is ook een Wicket applicatie uit het Somtoday landschap. Somtoday Core is de kern van het leerling administratiesysteem. Op het bovenstaande ‘Infrastructuur Diagram’ is een Sessie store zichtbaar. Dit is een opslag van de Wicket SessionStore die lokaal op de applicatie pod in kubernetes wordt bijgehouden.[32] Dit is tijdens mijn afstudeerperiode toegevoegd om snelheden te verbeteren.

4.4 Somtoday Jobs

Somtoday jobs is een los onderdeel van Somtoday en wordt gebruikt voor het uitvoeren van taken in somtoday. Dit is een recente toevoeging aan Somtoday en bevat momenteel maar 1 ‘job’. Dit is het herberekenen van cijfers. Hiervoor gebruikt het een ‘Message bus’ wat een NATS Messaging applicatie is.

4.5 Somtoday DB(‘s)

Somtoday maakt gebruik van 3 postgresql databases. Dit zijn DB1, DB2 en DB3. Hiervan zijn DB1 en DB3 de 2 databases die direct met Somtoday Core en ELO communiceren. En DB2 word als een ‘hot-standby’ database gebruikt. Deze database bevat de data van beiden DB1 en DB3 en dient als een backup.

5. Onderzoek

Om deze opdracht voor Topicus.Education B.V. uit te voeren is een Onderzoeksrapport [4] opgezet. In dit hoofdstuk wordt ingegaan op de bevindingen uit dit onderzoek. En hoe deze bevindingen tot stand zijn gekomen. Het volledige onderzoek is te vinden in het eerder genoemde Onderzoeksrapport.

5.1 Hoofdvraag

Het doel van dit onderzoeksrapport is om een een antwoord te vinden op de hoofdvraag:

- Hoe kan het beste de sessie-data los worden getrokken van de individuele Kubernetes pods voor de applicatie Somtoday ELO en Somtoday Core.

De hoofdvraag bestaat uit een aantal onderdelen:

- Somtoday ELO en Somtoday Core zijn de 2 applicaties waarvoor dit onderzoek wordt uitgevoerd.
- Kubernetes is het systeem waar Somtoday gebruik van maakt en pods zijn instanties van de applicaties in Kubernetes.
- sessie-data is data die de gebruiker nodig heeft om ingelogd te blijven.

Dus als de sessie data losgetrokken kan worden van individuele Kubernetes pods kunnen gebruikers ingelogd blijven en word hun sessie behouden. Ook als de achterliggende pod bijvoorbeeld verwijderd wordt en de gebruiker wordt overgezet naar een andere pod.

5.2 Deelvragen

Om de hoofdvraag van dit onderzoek te kunnen beantwoorden zijn twee deelvragen opgesteld.

Deze 2 deelvragen zijn voor 2 verschillende applicaties. Door de overeenkomsten tussen deze applicaties is er geen significant verschil in aanpak en worden er soortgelijke overwegingen gemaakt. Daarom worden deze 2 deelvragen samen beantwoord.

Deelvraag 1:

- Hoe kan de state van Somtoday ELO onafhankelijk van de applicatie pods worden gemaakt?

Deelvraag 2:

- Hoe kan de state van Somtoday Core onafhankelijk van de applicatie pods worden gemaakt?

5.2.1 Probleemstelling deelvraag 1 en 2:

Gebruikers van Somtoday Core en Somtoday ELO kunnen niet zonder hun state te verliezen overgezet worden naar andere Kubernetes pods. Hierdoor kan Somtoday niet op- of afschalen zonder de gebruikers uit te loggen.

5.2.2 Vooronderzoek:

De eerste stap van dit onderzoek is goed begrijpen wat het probleem is.

Somtoday Core + Somtoday ELO:
Kort samengevat, Somtoday Core en Somtoday ELO zijn onderdelen van het dynamische Somtoday landschap. Waar deze onderdelen voor gebruikt worden is all in dit document besproken.

Kubernetes:
'Kubernetes is een systeem dat containers (gecontaineriseerde applicaties) beheert, waarbij een container zou kunnen worden uitgelegd als een lichtgewicht virtuele machine. Om een applicatie te bouwen, moet je een aantal containers bouwen en vervolgens Kubernetes gebruiken om die containers te beheren. Het voordeel van Kubernetes is dat het containers automatisch kan maken en schalen en de opslag tussen alle containers kan beheren.' ([15] Eric-swildens, 2021)

Kubernetes pods zijn, *'een Kubernetes-abstractie die een groep van een of meer containers (zoals **Docker**) en enkele gedeelde bronnen(resources) voor die containers vertegenwoordigt.' 'Een Pod draait altijd op een Node. Een Node is een werkmachine in Kubernetes en kan een virtuele of een fysieke machine zijn, afhankelijk van het cluster. Een node kan meerdere pods bevatten'* ([16] Viewing Pods and Nodes, 2021)

'Een Kubernetes-cluster is een set van nodes die gecontaineriseerde toepassingen uitvoeren.' ([17] What is a Kubernetes cluster?, z.d.)

States:
Omdat de state van de gebruiker wordt bijgehouden op de Kubernetes pod zelf kunnen gebruikers niet zonder hun state te verliezen worden overgezet naar andere pods. Hierdoor kan de applicatie niet op of afschalen zonder de gebruikers uit te loggen. Nu worden er een aantal onderdelen van het Wicket framework die op de pod zelf opgeslagen dit zijn:

- IPageStore
    - *'De rol van IPageStore is om te bemiddelen bij het opslaan en laden van pagina's door de onderliggende IDataStore.'* ([18] Page Storage - Apache Wicket - Apache Software Foundation, 2020)
- IDataStore
    - *'IDataStore wordt gebruikt om Wicket-pagina's (als bytes) naar een persistente opslag zoals b.v. bestanden of databases.'* ([18] Page Storage - Apache Wicket - Apache Software Foundation, 2020)

5.3 Onderzoek deelvragen 1 en 2:

Voor dit onderzoek wordt een Proof of Concept(PoC) applicatie gemaakt. Het doel van dit PoC is om meerdere mogelijke oplossingen op het probleem uit te testen. En op die manier te kunnen bepalen wat de beste oplossing is.

5.3.1 Opzetten van een PoC applicatie:

Voor het opzetten van het PoC is Apache Wicket met Maven als bouwtool gekozen. Dit is gedaan omdat Somtoday hier ook gebruik van maakt. Dit bestaat uit 2 onderdelen.

- Het opzetten van een Apache Wicket applicatie.
- Het opzetten van een Apache Tomcat server. Tomcat is noodzakelijk om de applicatie lokaal te kunnen starten.

Opzetten van Docker:
Deze applicatie is vervolgens gedockerized met een Dockerfile. Voor de Dockerfile is Jetty als run-environment / servlet container gebruikt in plaats van Tomcat. De reden hiervoor is dat Tomcat niet heel gebruiksvriendelijk is in het geven van error messages. En er is voor Docker gekozen omdat Somtoday ook gebruik maakt van Docker. Later in dit onderzoek word dit omgeschreven naar WildFly omdat Somtoday daar gebruik van maakt, en niet van Jetty.

Opzetten van Kubernetes:
Nu de Wicket applicatie in Docker staat kan deze toegevoegd worden aan Kubernetes. Om Kubernetes lokaal te starten is Minikube gebruikt.

Het testbaar maken van de PageStore en DataStore:
Nu het PoC gebruik maakt van Kubernetes is de volgende stap het uitbreiden van het PoC zodat er getest kan worden of de gebruiker van het PoC inderdaad zijn state verliest als de pod waarop hij zit wordt afgeschaald(verwijderd) in Kubernetes. Hiervoor zijn:

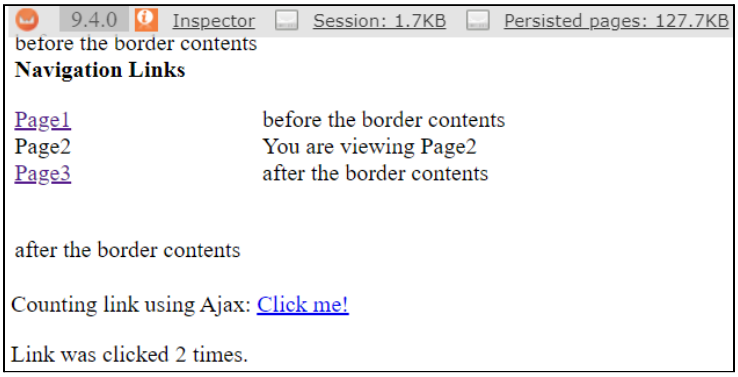
- Drie pagina's toegevoegd aan het PoC waar doorheen genavigeerd kan worden. Dit is gedaan om de IPageStore te kunnen testen.

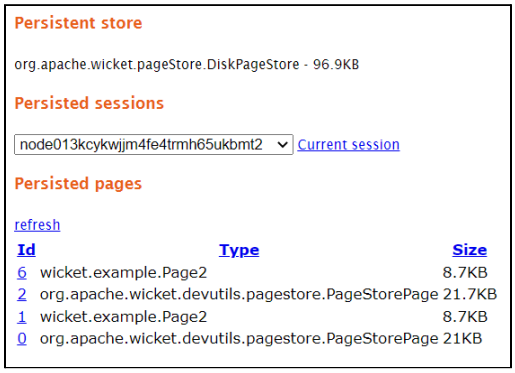
- Er is een teller element(counter) toegevoegd die bijhoudt hoe vaak er op een link geklikt is. Dit is gedaan om de IDataStore te kunnen testen.

- Ook is een debug bar toegevoegd aan het PoC via de 'Wicket Development Utilities' dependency. Dit voegt een component toe waarop de inhoud van de IPageStore en de sessie data op ingezien kan worden.

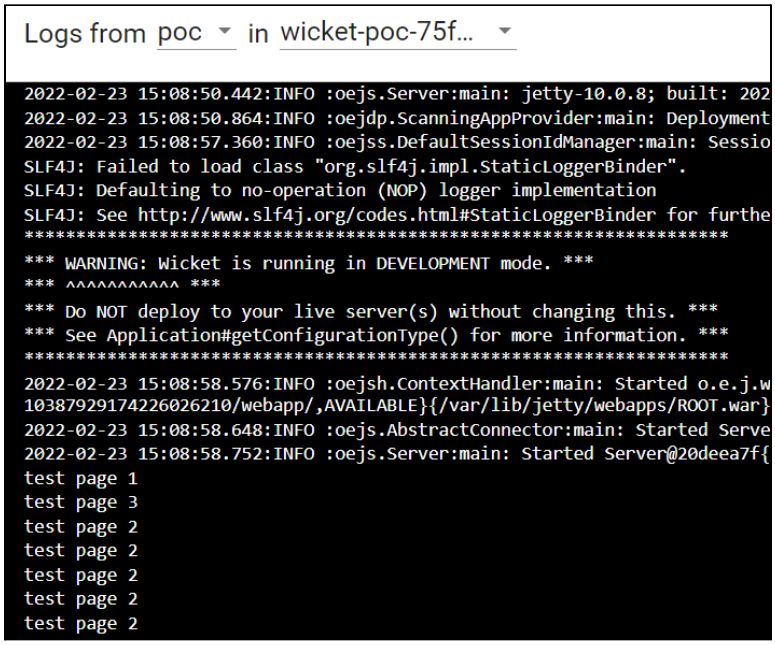
5.3.2 Probleemstelling aantonen en testen:

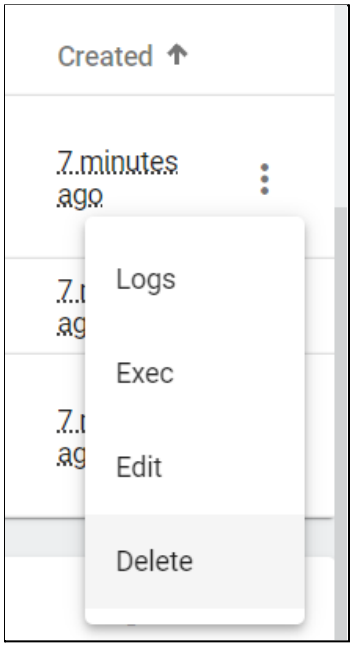
Nu deze 3 onderdelen toegevoegd zijn kan er getest worden of een gebruiker van het PoC zijn state verliest als de pod waar zijn sessie data in zit wordt afgeschaald. Aan de onderstaande foto is te zien dat de gebruiker Page 1, Page 2 en Page 3 heeft bezocht. Ook is te zien dat de gebruiker twee keer op ‘Click me!’ heeft gedrukt.



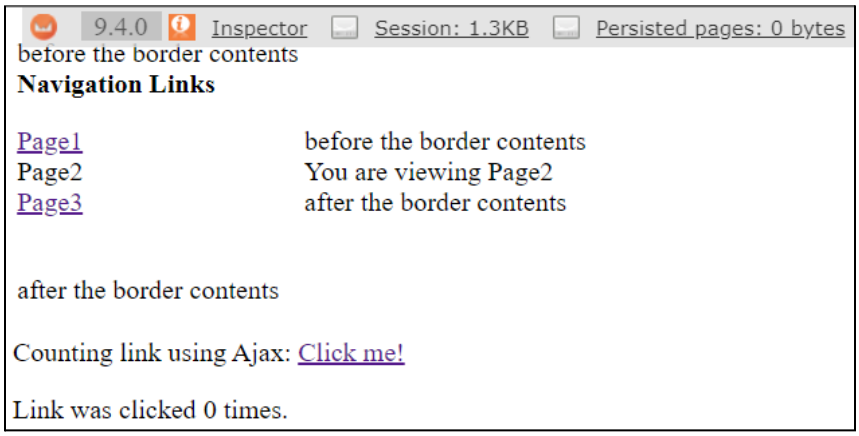


Door naar de Kubernetes logs te kijken is te zien dat de gebruiker op pod wicket-poc-75fc67cb9-2j2gl zit. Dus deze pod is de pod die we gaan afschalen.





Na het verversen van de webpagina is te zien dat de ‘Click me!’ teller weer op 0 staat. Ook heeft ‘Persisted pages’ weer 0 bytes en is dus leeg. Verder is er ook 0.4KB aan session data verloren gegaan.



5.4 AAD onderzoeksmethode:

Om te onderzoeken hoe het probleem van deze onderzoeksvraag het beste verholpen kan worden wordt de “Advanced Application Development Research approach” gebruikt. Deze methode bestaat uit 4 fases die doorlopen moeten worden:

- **Fase 0: Opstellen Longlist**
  Het maken van een lijst met mogelijke oplossingen die gebruikt kunnen worden voor het project.

  Uit deze longlist worden vervolgens de 3 (of meer) kansrijkste kandidaten geselecteerd.

- **Fase 1: Vergelijken Kandidaten**
  De 3 kansrijkste kandidaten worden verder vergeleken. De vergelijking gebeurt met een lijst met vooraf gedefinieerde criteria.

  De kandidaten worden op ieder criteria beoordeeld, naast het cijfer wordt ook een motivatie voor deze beoordeling gegeven.

  Uiteindelijk worden de 2 hoogst scorende kandidaten verder onderzocht.

- **Fase 2: Implementeren Proof of Concept**
  Met de 2 hoogst scorende oplossingen wordt een proof of concept ontwikkeld. In dit geval wordt het Proof of Concept die in de vorige casus is gemaakt ermee doorontwikkeld.

- **Fase 3: Bepalen keuze**
  Er wordt gereflecteerd op beide prototypes en er wordt gemotiveerd welke het best geschikt is en waarom.

5.4.1 Fase 0: Opstellen Longlist:

De onderstaande oplossingen zijn verkregen uit eigen onderzoek en als mogelijke technologieën / kandidaten die Topicus heeft aangeboden.

Voor de onderstaande technologie om in aanmerking te komen moeten deze aan een aantal vereisten criteria voldoen. Dit zijn:

- Het moet geschikt zijn voor een cloudomgeving
    - Een van de randvoorwaarden die zijn gesteld is *'Als systeembeheerder van Somtoday wil ik dat de opgeleverde oplossing niet kan verhinderen dat Somtoday later omgezet kan worden omgezet naar een cloud omgeving.' (*[2] Requirements - Afstudeeropdracht Topicus)
- Het moet snel zijn.
    - De opslag wordt gebruikt voor sessie data. Hierdoor zijn veelgebruikte datastores niet geschikt omdat sessie data constant wordt aangepast en langdurige opslag niet nodig is.
- Het moet te benaderen zijn vanuit een Java applicatie.
    - Somtoday en Apache Wicket zijn geschreven in Java.

Om een geschikte oplossing te vinden zijn er verschillende oplossings routes bekeken. Een aantal voorbeelden hiervan zijn:

- Docker volume:
    - Een Docker volume maakt gebruik van een bestandsstructuur. Hierdoor is het niet geschikt voor het beheren van sessie data.[25]
- In de Somtodays eigen Postgresql database:
    - Somtoday maakt gebruik van een Postgresql database. Deze database is echter niet snel genoeg voor het beheren van sessies. Dit heeft meerdere redenen. De eerste is dat sessie blobs veel groter zijn dan de normale enkele variabele die in de database worden geplaatst. De tweede reden is dat deze database niet gebruik maakt van werkgeheugen voor zijn opslag.
- Het alleen opslaan van sessie-data in Postgresql tijdens shutdown.
    - Het alleen opslaan van sessie-data tijdens een herstart in de al bestaande Postgresql database zou kunnen werken. Dit is omdat de lading op de database dan van hele korte duur is. Alleen loop je dan tegen het probleem aan dat de data niet opgeslagen wordt bij een crash. De reden hier is omdat crashes nooit te voorspellen zijn. En hierdoor voldoet deze oplossing niet aan de gestelde requirements.

Met deze redenen is er gekozen voor caching servers die gebruik maken van werkgeheugen voor het behouden van data. Hiervoor is een lijst met veelbelovende technologieën samengesteld. Niet alle technologieën in deze lijst zijn zelf uitgekozen, sommige zijn ook aangeboden door Topicus zelf als mogelijke oplossing.

De eerste stap van het AAD onderzoeksmodel is het opstellen van een longlist. Hiervoor zijn een aantal verschillende technologieën opgeschreven. Hierin staat als eerste opgeschreven wat de technologie inhoud en als tweede de voor en nadelen hiervan. Hieronder staat beschreven hoe dit is gedaan voor Redis:

Redis Remote Dictionary Server is een in-memory data structuur opslag, die wordt gebruikt als een gedistribueerde, in-memory key-value-database, cache en message broker. Redis ondersteunt verschillende soorten abstracte datastructuren, zoals strings, lists, maps, sets, bitmaps, streams en spatial indices.

Voordelen:

- Het wordt veel gebruikt in open source software projecten. [27]
- Het is open source software. [27]
- Geschikt voor een cloudomgeving via bijvoorbeeld Google cloud memorystore. [28]
- Redis is geschikt voor Kubernetes, dit maakt het goed schaalbaar. [29]
- Door gebruik van in-memory cache is Redis heel snel. [27]
- Redis wordt vaak gebruikt als session store. [27]
- Redis bied real-time analytics via Apache Kafka of NATS. [27]
- Redis samen met Memcached zijn de twee populairste in-memory key-value datastores. [27]
- Redis wordt gebruikt door bekende bedrijven zoals Twitter, Github, Pinterest, Snapchat, Craigslist en Stackoverflow. [31]

Nadelen:

- Redis alleen is niet geschikt voor langdurige opslag alleen als cache. [30]
- Redis is een datastructuurserver. Er is geen query language (alleen commando's). [30]
- Redis biedt alleen basisbeveiliging (in termen van toegangsrechten). [30]
- Redis is (meestal) single-threaded. [30]
- Redis biedt geen right out of the box compatibiliteit met de Java-programmeertaal. [29]

In het Onderzoeksrapport[4] zijn voor fase 0 de volgende technologieën onderzocht:

- Redis
    - Te vinden onder: ‘3.5.1.1 Redis’
- Memcached
    - Te vinden onder: ‘3.5.1.2 Memcached’
- Infinispan
    - Te vinden onder: ‘3.5.1.3 Infinispan’
- NATS
    - Te vinden onder: ‘3.5.1.4 NATS’
- Apache Ignite - GridGain
    - Te vinden onder: ‘3.5.1.5 Apache Ignite - GridGain’
- Hazelcast
    - Te vinden onder: ‘3.5.1.6 Hazelcast’

5.4.2 Fase 1: Vergelijken Kandidaten

Voor fase 1 zijn 4 kandidaten gekozen. Dit zijn Redis, Memcached, Hazelcast en Apache Ignite. Deze 4 kandidaten zijn gekozen omdat ze geen grote nadelen hadden. Alle voor en nadelen zijn te vinden in het onderzoeksrapport[4] onder ‘3.5.1 Fase 0: Opstellen Longlist:‘

NATS is voornamelijk afgevallen omdat de technologie NATS JetStream te nieuw is. Hierdoor wordt het nog maar heel weinig gebruikt en is er nog maar heel weinig informatie over te vinden. Dit maakt het ontwikkelen ermee veel lastiger.

Infinispan is afgevallen door een vergelijkbare reden. Er was weinig informatie over te vinden naast de Infinispan website zelf.

Van de 4 overgebleven kandidaten wordt een beoordelingstabel gemaakt. Hiermee worden de kandidaten beoordeeld en vallen de 2 laagste scorende kandidaten af.

Aan ieder criteria wordt vervolgens ook nog een weging van 1 t/m 5 gegeven.
Niet ieder criteria is namelijk even belangrijk sinds Topicus niet overal dezelfde waarde aan hecht. Deze criteria en hun wegingen zijn in afspraak met Topicus tot stand gekomen.

Criteria	Weging	Score 1 t/m 10
Installatie	1	-
Documentatie	3	-
Community	3	-
Uitbreidbaarheid	3	-
Prestaties	5	-
learning curve/Implementatie moeilijkheidsgraad	3	-
Kosten	2	-
Monitoring	3	-
Onderhoudbaarheid	3	–
Onderliggende technologie	-	-

Een volledige beschrijving van alle criterias en de motivatie achter alle gekozen scores zijn te vinden in het onderzoeksrapport[4] onder ‘3.5.2.1 Beoordelings-tabel‘

In het Onderzoeksrapport zijn voor fase 1 de volgende technologieën onderzocht:

- Redis
    - Te vinden onder: ‘3.5.2.2 Redis’ met een eindscore van 8.40
- Memcached
    - Te vinden onder: ‘3.5.2.3 Memcached’ met een eindscore van 7.08
- Apache Ignite - GridGain
    - Te vinden onder: ‘3.5.2.4 Apache Ignite - GridGain’ met een eindscore van 7.12
- Hazelcast
    - Te vinden onder: ‘3.5.2.5 Hazelcast’ met een eindscore van 7.84

Een van de technologieën die zijn onderzocht is Redis. Deze technologie wordt hier gebruikt als voorbeeld. De andere technologieën uit de shortlist zijn te vinden in het onderzoeksrapport op de hierboven genoemde locaties.

Redis		
Criteria	**Weging**	**Score 1 t/m 10**
Installatie	1	7
Documentatie	3	8
Community	3	9
Uitbreidbaarheid	3	8
Prestaties	5	7
learning curve/Implementatie moeilijkheidsgraad	3	8
Kosten	2	10
Monitoring	3	10
Onderhoudbaarheid	3	9
Onderliggende technologie	-	-
Totaal score	**19**	**8.40**

Motivatie:

Installatie:
Bij Redis moet er een server geïnstalleerd worden. Dit bestaat voor Windows machines uit 4 stappen. Het kan ook via Docker geïnstalleerd worden. Dit bestaat uit 1 stap. Vervolgens is het te benaderen via een Maven dependency. [32] [33]

Documentatie:
De documentatie bevat voorbeelden / tutorials, uitleg over de beschikbare commands, downloads, uitleg over troubleshooting / performance, en beschikbare plugins. En is over het algemeen heel uitgebreid. [34]

Community:
Redis is een veel gebruikte technologie die al voor een langere tijd bestaat (2009). Dit is ook te zien aan hoeveel er over te vinden is op forums. Op stackoverflow.com zijn er 22,300 getagde vragen over. Wat veruit het meeste is vergeleken de andere technologieen in de shortlist.

Uitbreidbaarheid:
Redis biedt een aantal ondersteunde modules die extra functionaliteiten bieden. Standaard heeft Redis ook een groot aantal data types beschikbaar. Redis heeft echter ‘out of the box’ weinig beveiligingsopties. [30]

Prestaties:
Redis is een in-memory store dit geeft het response times van enkele milliseconden. Er moet wel rekening mee gehouden worden dat Redis single threaded is. Dit betekend dat het maar 1 (virtuele) kern van een processor tegelijkertijd kan gebruiken.

Redis heeft een meegeleverde benchmark tool. In de onderstaande link zijn de prestaties te vinden tijdens een benchmark met 50 gesimuleerde gelijktijdige clients die tot 2 miljoen verzoeken per seconden uitvoeren. [35]

learning curve / Implementatie moeilijkheidsgraad:
Redis heeft een grote hoeveelheid features dat maakt het dus lastiger om mee te leren werken dan bijvoorbeeld Memcached wat beduidend minder features bezit.

Kosten:
Redis is open source software, hierdoor is het gratis. Er zijn echter wel veel betaalde cloud hosting services beschikbaar. Redis biedt ook een betaalde versie, Redis Enterprise. Redis Enterprise is een hosted service dat Redis db gebruikt verder heeft het meer functionaliteiten beschikbaar tegenover de gratis versie van Redis.

De gratis versie van Redis mag wel gebruikt worden voor commerciële doeleinden en biedt bijna alles dat Redis Enterprice ook biedt. [36]

Monitoring:
Bij topicus Onderwijs wordt Grafana 8.4.2 gebruikt voor monitoring. Grafana is een multi-platform analytics en dachboardtool. Redis heeft hier volledige integratie voor via een plugin. [37]

Onderhoudbaarheid:
Redis wordt door heel veel partijen gebruikt hierdoor is het van belang dat het uptodate blijft. Hierom brengt reddis elk jaar een nieuwe (volledige) versie uit. Verder wordt er ook een Wicketstuff wicket-redis-integratie dependency gebruikt. Deze wordt ook meerdere keren per jaar geupdate, vaak kort na dat Redis is geupdate. [38] [39]

Onderliggende technologie:
‘Redis is geschreven in de ANSI C-taal en werkt in de meeste POSIX-systemen zoals BSD, Linux, OS X zonder enige externe afhankelijkheden. OS X en Linux worden beschouwd als de twee besturingssystemen waarop Redis het meest is ontwikkeld en getest, terwijl Linux is gebruikt voor de implementatie ervan.’ [40]

5.4.2.1 Motivatie voor de gekozen technologieën.

Voor fase 2 van de AAD onderzoeksmethode worden er 2 prototypes gemaakt. De 2 technologieën die hiervoor gebruikt gaan worden zijn:

- Redis
    - De eerste technologie die is gekozen is Redis. Redis had de hoogste score van fase 1. Verder had het veruit de grootste community, heeft het hele duidelijke en uitgebreide documentatie en levert het prima prestaties. Het help ook mee dat het een industrie standaard is en dat Topicus het als een mogelijk oplossing heeft aangedragen.

- Hazelcast
    - De tweede technologie die is gekozen is Hazelcast. Hazelcast had na Redis de hoogste score. Het heeft hele duidelijke documentatie en ook al is het een nieuwere technologie wordt het toch in een korte tijd door veel developers opgepakt.

      Het gebruikt ook als enigste technologie uit de shortlist een gridstructuur voor dataopslag. Dit is een set netwerk-/geclusterde computers die hun RAM bundelen om gegevens te delen met andere toepassingen die in de cluster worden uitgevoerd. Dit maakt het Hazelcast veel schaalbaarder dan andere opties zolang er genoeg resources beschikbaar zijn. De vraag is of deze schaalbaarheid noodzakelijk is voor Somtoday.

5.4.3 Fase 2: Maken Prototypes

In deze fase worden met de 2 best presterende technologieën 2 verschillende prototypes ontwikkeld. Hiermee wordt vervolgens geverifieerd of het uitvoeren van de opdracht met die technologie mogelijk is.

In dit verslag word alleen ingegaan op het Redis PoC. De reden hiervoor is dat dit document anders te lang werd. De volledige uitwerking van het Hazelcast PoC is te vinden in het onderzoeksrapport[4] onder ‘3.5.4.3 Hazelcast’. En de volledige uitwerking van het Redis PoC is te vinden in het onderzoeksrapport onder ‘3.5.4.1 Redis’

Opdrachtomschrijving:
In het PoC kan momenteel de PageStore en de SessionStore van de huidige gebruiker in gezien worden. Als de pod waar die gebruiker op zit wordt verwijderd worden deze beiden vervolgens geleegd. Het doel is om het huidige PoC zo aan te passen dat dit niet meer gebeurd.

5.4.3.1 Proof of Concept met Redis:

Redis installeren op Docker:
De eerste technologie die toegepast gaat worden op het PoC is Redis. Hiervoor moet dit eerst geïnstalleerd worden. Voor de installatie hiervan zijn meerdere mogelijkheden. Voor dit project is Docker gekozen omdat Kubernetes ook in Docker staat. Dit word gedaan om de connectie met Redis vanaf het PoC te kunnen testen en om te leren hoe Redis werkt.

Vervolgens is de Redis Command Line Interface (CLI) ook lokaal geïnstalleerd. Dit is noodzakelijk voor het verbinden aan een bestaande redis instantie via CMD of PowerShell.

Redis compatibiliteit toevoegen aan Java:
Nu Redis is geïnstalleerd moet er een Maven dependency toegevoegd worden aan de PoC. Dit is gedaan omdat Redis geen out of the box compatibiliteit biedt met de Java-programmeertaal. Er zijn 2 keuzes om Java integratie toe te voegen, dit zijn:

- Jedis
    - Een Java-client voor Redis ontworpen voor prestaties en gebruiksgemak.
- Lettuce
    - Een schaalbare thread-safe Java-client voor Redis voor synchroon, asynchroon en reactief gebruik.

Voor dit project is gekozen voor Jedis. Dit is gedaan omdat de ‘wicketstuff-datastore-redis’ Maven dependency, die gebruikt word in een latere stap van dit project, afhankelijk is van Jedis.

Redis installeren op Kubernetes:
Om Redis schaalbaar te maken is Redis toegevoegd aan Kubernetes. Hiervoor is een Redis cluster toegevoegd aan Kubernetes. Dit houd in dat er meerdere instanties van Redis tegelijkertijd gebruikt kunnen worden.

PageStore migreren naar Redis:
Nu Redis aan Kubernetes is toegevoegd kan er begonnen worden met het lostrekken van de sessie data van de gebruiker en dit vervolgens migreren naar Redis. Deze sessie data bestaat uit 2 onderdelen zoals beschreven in ‘2.1.3 Vooronderzoek’.

- De PageStore die de webpaginas bevat als bytearray en pagina attributen.
- De Datastore die sessie data bevat (de SessionStore) en bij default ook de Pagestore zelf.

Het eerste wat is gedaan is het lostrekken van de PageStore. Hiervoor is een Maven dependency gebruikt van Wicketstuff (wicketstuff-datastore-redis). Deze dependency is een custom IPageManager die Redis gebruikt als backend in plaats van Wickets standaard PageStore. Om de wicketstuff-datastore-redis dependency te gebruiken moet de default Wicket PageManagerProvider worden overschreven naar die van wicketstuff-datastore-redis. Dit is gedaan in de ‘.init()’ methode van Wicket. In dit geval het ‘HelloWicketApplication.java’ bestand.

SessionStore migreren naar Redis:

Nu de Wicket PageStore is los getrokken naar Redis is de volgende stap om ditzelfde te doen met de Wicket SessionStore.

Hier is echter geen dependency voor beschikbaar dus er zal een eigen SessionStore implementatie gemaakt moeten worden worden die Redis als een backend gebruikt. Hiervoor is een bestaand project gebruikt[19]:

Dit project bevat meerdere java bestanden. Hier hebben we echter alleen maar 2 bestanden van nodig:

- RedisSessionStore.java
    - Dit is een class die gebruikt word in plaats van de Wicket SessionStore. Deze class bevat alle methoden die de Wicket ISessionStore interface nodig heeft.
- RedisCache.java
    - RedisCache wordt gebruikt door RedisSessionStore om sessie-data op te slaan in Redis.

Omdat het project waarvan de SessionStore wordt gebruikt niet is geupdate sinds 2015 zijn hier ook een aantal dingen op aangepast.

Load Balancer toevoegen aan het PoC:
Nu alle sessie-data van de gebruiker in Redis word opgeslagen zou dit het verwijderen van de kubernetes pod waar die gebruiker op zit, het moeten overleven. En deze sessie-data blijft ook bestaan in Redis als dit gebeurd. Er zijn echter nog wat problemen gevonden met het hergebruiken van deze sessie data.

Het eerste probleem is dat er geen load balancer is toegevoegd aan het PoC.

Voor het testen van het PoC worden 2 applicatie pod’s gebruikt met daarop de Wicket applicatie gemaakt in ‘5.3.2 Probleemstelling aantonen en testen’. Het niet hebben van een load balancer betekent dat een gebruiker niet ‘sticky’ is op een van die 2 pod’s. Hierdoor kunnen gebruikers spontaan worden overgezet naar een andere beschikbare applicatie pod (ze zijn dus niet sticky). En omdat gebruikers sessie dan nog op de vorige pod staat krijgt deze gebruiker hierdoor een nieuwe sessie aangewezen omdat de nieuwe pod hem niet herkent, en hierdoor raakt hij dus zijn oude sessie kwijt.

Om het eerste probleem te verhelpen is een load balancer met sticky sessies toegevoegd aan het PoC via Istio. Istio breidt Kubernetes uit met een programmeerbaar, applicatie bewust netwerk met behulp van een Envoy-serviceproxy. Hier is voor gekozen omdat Somtoday hier ook gebruik van maakt om ditzelfde probleem op te lossen.

Hergebruiken van Wicket sessies:

Nu we weten dat een gebruiker niet spontaan van pod kan veranderen, kan er begonnen worden aan het oplossen van het tweede probleem. Als een gebruiker op een nieuwe pod terecht komt herkent deze nieuwe pod niet zijn oude sessie. En hierdoor krijgt de gebruiker alsnog een nieuwe sessie aangewezen. Dus er moet een manier toegevoegd worden voor de Wicket applicatie in de pod om al bestaande sessies te kunnen herkennen en gebruiken.

Om dit probleem op te kunnen lossen moet eerst goed begrepen worden hoe sessies worden opgeslagen en hoe deze vervolgens worden opgehaald. Dit wordt gedaan in de bestanden ‘RedisSessionStore.java’ en ‘RedisDataStore.class’. Deze laatste is te vinden in de ‘org.wicketstuff.datastores.redis’ external library, een bestand dat wordt gedownload bij het binnenhalen van alle benodigde dependencies.

Om sessies te herkennen wordt in beide gevallen een ‘sessionIdentifier’ string voor gebruikt. Deze sessionIdentifier kan op verschillende manieren worden verkregen:

- Via het ‘httpSession’ java object zelf. Dit object wordt opgeslagen in Java’s eigen tijdelijke opslag.
- Indien het is ingesteld uit de uri. Dit is de link in een browser die word gebruikt om websites te bezoeken.
- Via een Cookie die tijdelijk is opgeslagen in de browser van een gebruiker.

Van deze 3 oplossingen is de laatste gekozen. De eerste oplossing werkt niet omdat het httpSession java object wordt verwijderd als Java wordt afgesloten. In dit geval gebeurt dit als een pod wordt verwijderd. Het ‘httpSession’ object is opgeslagen in de interne Java opslag (de on-heap / off-heap memory). En deze opslag wordt geleegd Java word afgesloten. Deze opslag is gekoppeld aan de pod zelf en kan deze niet worden gedeeld met andere pod’s.

De tweede optie is ook niet mogelijk omdat Somtoday zijn sessionsIdentifier niet en de uri zet.

Voor Wicket om de Session Identifier Cookie te kunnen hergebruiken en deze te koppelen aan een al bestaande sessie in Redis is er wat code toegevoegd aan ‘RedisSessionStore.java’. In dit bestand zit een methode die bij nieuwe connecties een nieuwe sessies aanmaakt voor de gebruiker. Hieraan is een methode toegevoegd die aan de hand van de Session Identifier Cookie eerst controleert of de nieuwe gebruiker al een bestaande sessie in Redis heeft staan.

Wat deze code is en wat het doet is te vinden in het onderzoeksrapport[4] onder het ‘Hergebruiken van sessies’ hoofdstuk van ‘3.5.4.1 Redis’.

Met deze aanpassingen kunnen Wicket sessies hergebruikt worden met Redis en is de probleemstelling opgelost.

5.5 WildFly sessies

Nadat het Redis PoC was aangepast om de Wicket sessies te kunnen hergebruiken bleek dat de sessie-data los trekken van de individuele Kubernetes pods nog een derde onderdeel bevatte. Dit is de sessie-data die in de applicatieserver WildFly wordt vastgehouden.

Dit werd ontdekt nadat het PoC was aangepast om de applicatieserver WildFly te gebruiken in plaats van Jetty omdat Somtoday hier ook gebruik van maakt. En hierna was er een teller toegevoegd aan het PoC die Jakarta Context and Dependency Injection (CDI) gebruikten. Deze teller maakte gebruik van de '@SessionScoped' interface. Dit houdt in dat de data van de teller persistent blijft zolang de sessie bestaat [20].

De data van deze teller wordt echter opgeslagen op de WildFly applicatieserver zelf. En omdat deze applicatieserver deel is van de pod die word afgesloten bij een herstart gaat deze data verloren. Ditzelfde probleem zou ook voorkomen op Jetty en is dus niet WildFly specifiek.

Somtoday maakt ook gebruik van deze interface. Daarom wordt er gekeken of deze data losgetrokken kan worden van de applicatie pods.

Om dit probleem op te lossen is ervoor gekozen om de huidige WildFly standalone applicatie server uit te wisselen voor een High Availability (HA) applicatie server met distributable web sessies:

- Door WildFly zo te configureren dat het (Kubernetes) cluster aware is kunnen distributable sessies worden toegepast. Deze sessies kunnen vervolgens in een externe database worden opgeslagen.

Als het PoC gebruik maakt van een HA WildFly applicatieserver worden sessies automatisch verdeeld over alle bestaande WildFly pods onder de zelfde namespace. Hierdoor gaan WildFly sessies niet verloren zolang een pod in hetzelfde cluster nog bestaat. En als een pod waar een gebruiker op zit crashed of herstart wordt de gebruiker hierdoor niet uitgelogd.

Dit is echter nog niet een complete oplossing omdat bij het updaten van Somtoday het complete cluster afgesloten wordt. Wat betekent dat er geen WildFly pod's meer beschikbaar zijn om de sessies over te nemen. Ook is het momenteel niet mogelijk om 'Rolling updates' te doen. De reden hiervoor is dat de huidige Postgresql database dan tegen versie problemen aanloopt.

Daarom is de volgende stap om te onderzoeken of de Wildfly sessies extern kunnen worden bijgehouden in Redis of Hazelcast.

5.5.1 WildFly sessies in Redis

Omdat Redis als de beste optie is gekozen wordt hier eerst onderzoek na gedaan. WildFly gebruikt Infinispan als Interne database en dat kan niet worden aangepast. Infinispan ondersteunde wel een plugin die de Infinispan cache kan overschrijven met Redis [41]

Deze plugin wordt echter niet meer ondersteund door Infinispan sinds 2016 en deze plugin is geschreven voor Infinispan 8. Daarom is deze plugin getest met Infinispan 8 in WildFly.

Dit is mogelijk door in het ‘standalone-full-ha.xml’ bestand het standaard infinispan 13 subsystem te overschrijven met die van Infinispan 8. Alleen heb ik het daarop ook niet werkend weten te krijgen. Dit was niet geheel onverwachts omdat tijdens mijn onderzoek op het internet al werd gewaarschuwd dat het aanpassen van de Infinispan cache naar Redis in WildFly niet (meer) mogelijk is.

Omdat WildFly gebruik moet maken van Infinispan en dit niet aan te passen is. En omdat Infinispan geen manier ondersteund om zijn cache op te slaan met een andere database. Het is daarom niet mogelijk om via de configuratie van WildFly de Infinispan cache los te trekken naar Redis.

5.5.2 WildFly sessies in Hazelcast

Omdat in het vorige hoofdstuk geconcludeerd is dat de WildFly cache niet losgetrokken kan worden naar Redis wordt er in dit hoofdstuk gekeken of het losgetrokken kan worden naar Hazelcast, de tweede technologie is in dit onderzoeksrapport was onderzocht.

Infinispan ondersteund geen plugin om de cache van Infinispan naar Hazelcast te verplaatsen. Hazelcast ondersteund wel een web-manager plugin[138][139]. Deze plugin wordt gebruikt voor het beheren van web-sessies in Hazelcast. Hierom is er onderzocht of deze webmanager ook gebruikt kan worden voor de cache van WildFly.

Hiervoor is het bestaande Hazelcast PoC eerst uitgebreid om gebruik te maken van een HA WildFly application server. Echter na het te hebben toegepast en correct te hebben geconfigureerd van de plugin, is geconcludeerd dat deze plugin hiervoor niet gebruikt kan worden. Deze plugin werkt wel voor het beheren van sessies zonder CDI.

5.5.3 WildFly sessies in een externe Infinispan database

Omdat WildFly momenteel alleen Infinispan integratie heeft is het lostrekken van de WildFly sessies ook uiteindelijk met Infinispan gedaan. Dit is dezelfde database die WildFly intern gebruikt voor het beheren van sessie-data. Hiervoor is onder andere een ‘Remote Cache’ toegevoegd aan WildFly’s ‘standalone-full-ha.xml’ configuratie bestand. Hoe dit precies is geconfigureerd is te vinden in het Onderzoeksrapport[4] onder ‘3.6.5 WildFly sessies in een extern Infinispan cluster’.

5.5.4 Het testen van WildFly persistent sessions:

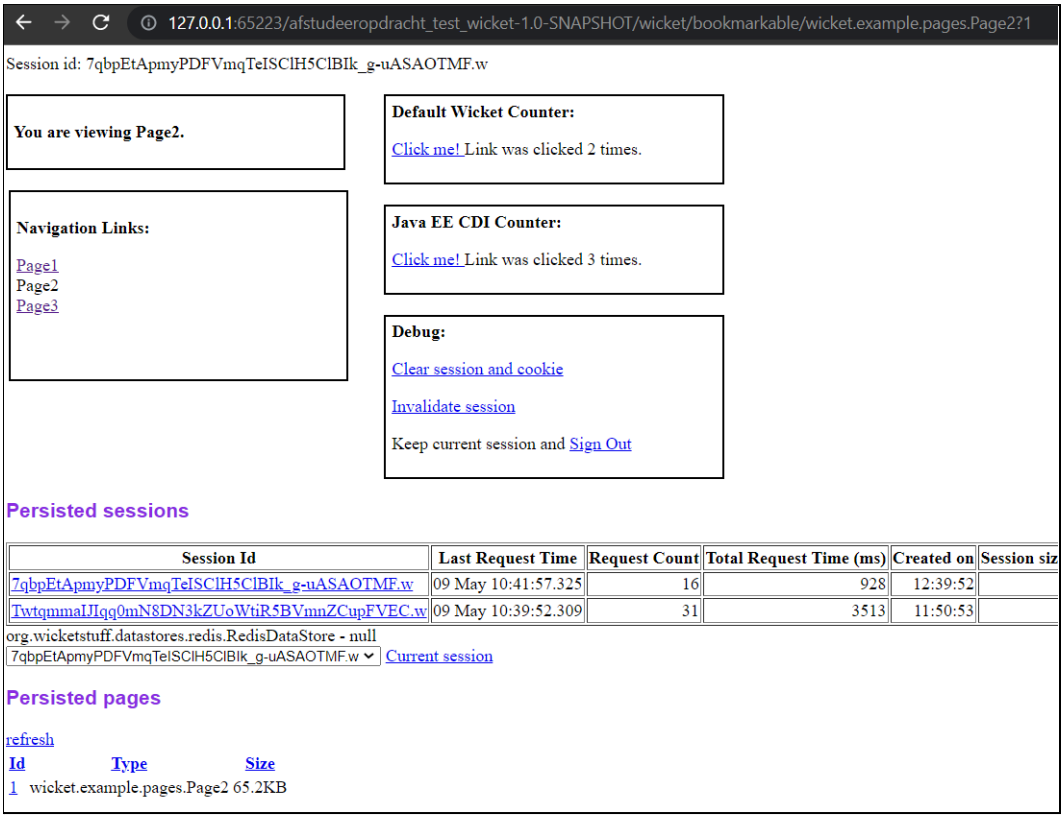
Nu WildFly is geconfigureerd om persistent sessions te gebruiken kan er getest worden of deze oplossing werkt. De volledige test is te in het Onderzoeksrapport[4] onder ‘Het testen van WildFly persistent sessions’.

Als het PoC is geopend en als hier op is ingelogd opent een scherm met de text ‘You are viewing Page1’ en 3 navigatie links. Druk op de link met de text ‘Page2’.

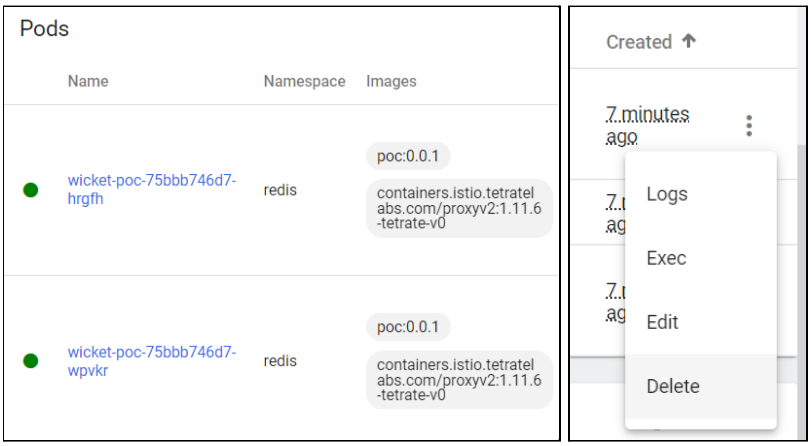
Dit opent de pagina ‘Page2’. Hierop zijn een aantal nieuwe elementen toegevoegd. Voor het testen zijn echter maar 2 elementen belangrijk. Dit zijn:

- Default Wicket Counter:
    - Dit is een standaard teller van Wicket. Hiermee wordt het behouden van Wicket sessies word getest.
- Java EE CDI Counter:
    - Dit is een teller die gebruik maakt van Jakarta Context and Dependency Injection (CDI) en maakt gebruik van de ‘@SessionScoped’ interface. Dit houd in dat de waarde van de teller bewaard blijft zolang de WildFly sessie bestaat. Hiermee wordt het behouden van de WildFly sessie getest.

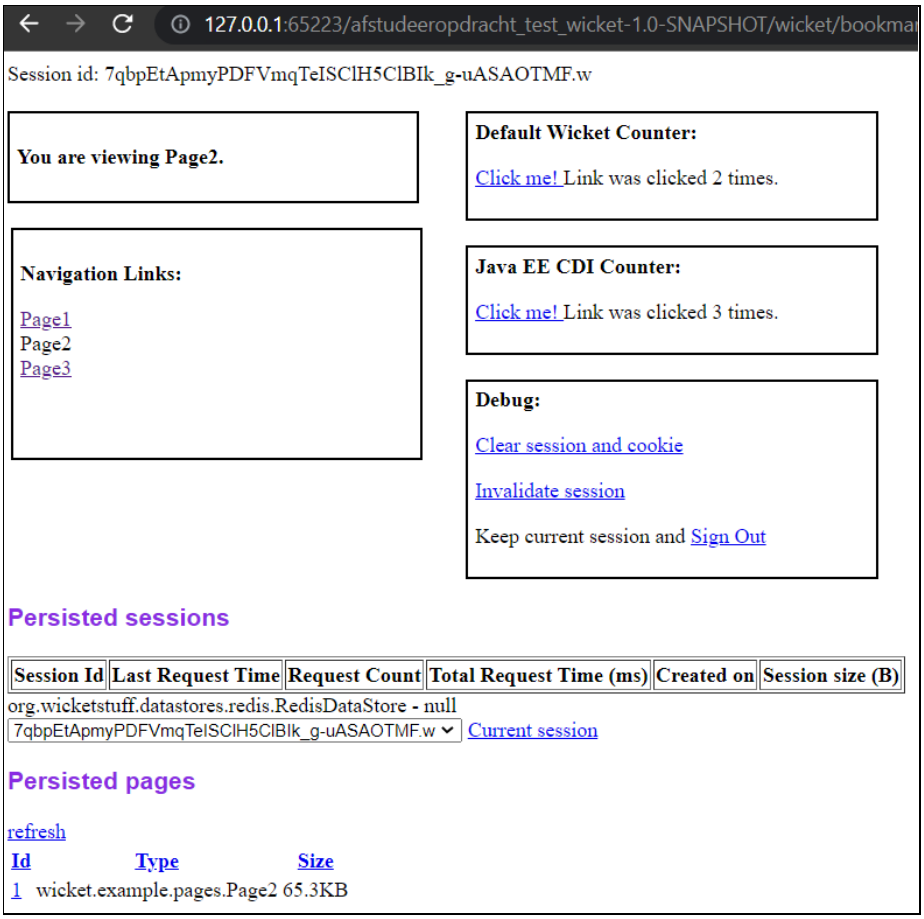
Vervolgens druk 2 keer op ‘Default Wicket Counter’ en 3 keer op de ‘Java EE CDI Counter’.



Hierna kunnen alle bestaande pods van de PoC applicatie worden herstart in het Kubernetes dashboard, tijdens deze test waren dit er 2:



Het starten van WildFly duurt 1 a 2 minuten dus hierop moet gewacht worden. Hierna kan de pagina worden herladen. Als dit is gedaan is te zien dat je niet wordt uitgelogd en dat beide tellers hun waardes hebben behouden.



5.5.5 Conclusie persistente WildFly sessies.

Het is mogelijk om WildFly sessie-data persistent te maken. Hiervoor kan alleen niet Redis of Hazelcast gebruikt worden.

Hiervoor moet WildFly eerst in High Availability mode gezet worden. Hierbij moet wel rekening gehouden worden dat Somtoday gebruik maakt van een standaard ‘Standalone’ WildFly configuratie.

Hierna kan een extern Infinispan cluster worden opgezet waarmee het PoC verbinding kan maken. En als dit cluster beschikbaar is kan WildFly zo geconfigureerd worden dat het zijn sessie-data daarin bijhoudt.

5.6 Conclusie deelvraag 1 en 2:

De deelvraag: ‘Hoe kan de state van Somtoday ELO en Somtoday Core onafhankelijk van de applicatie pods worden gemaakt?’ is te beantwoorden met het toepassen van een in-werkgeheugen sleutel-waarde dataopslag (in-memory, key-value, datastore).

Dit is een database die werkgeheugen gebruikt voor het beheren van data. Hiervoor is gekozen omdat werkgeheugen veel sneller is dan traditionele verwerkingsmethoden. En meer traditionele databases zijn vanwege hun verwerkingssnelheid niet geschikt voor het beheren van sessies.

Aan de hand van de AAD onderzoeksmethoden is bepaald dat de 2 meest geschikte kandidaten hiervoor de technologieen Redis en Hazelcast zijn.

Deze technologieën kunnen gebruikt worden voor het onafhankelijk maken en behouden van de state (sessie-data) van de gebruikers van een Apache Wicket applicatie. Dit bestaat uit de Wicket pageStore en de Wicket sessionStore. Dit kan worden gedaan door op maat implementaties van deze onderdelen toe te passen die een in-memory datastore gebruiken als backend. En de standaard wicket onderdelen hiermee te overschrijven.

Als de sessie data van gebruikers onafhankelijk van de applicatie pods zijn gemaakt kan door het hergebruiken van de sessie identificatie cookie die zich in de browser bevindt een gebruiker van deze zelfde sessie gebruik blijven maken. Ook als de pod waarop hij bezig was wordt afgeschaald.

Voor dit onderzoek zijn 2 Proof of Concepts (PoC’s) ontwikkeld. Een waarop de technologie Redis is toegepast. En een waarop de technologie Hazelcast is toegepast. Met beide technologieën kan deze deelvraag volledig worden beantwoord.

Voor beide technologieën moet echter ook de technologie Infinispan worden gebruikt voor het behouden van WildFly’s sessie-data.

5.7 Conclusie op de hoofdvraag:

Nu alle deelvragen zijn beantwoord kan er een antwoord gegeven op de hoofdvraag, dit was:

- Hoe kan het beste de sessie-data los worden getrokken van de individuele Kubernetes pods voor de applicatie Somtoday ELO en Somtoday Core.

Om de deelvragen te kunnen beantwoorden zijn er 2 Proof of Concepts (PoC's) ontwikkeld. Een waarop de technologie Redis is toegepast. En een waarop de technologie Hazelcast is toegepast. De hoofdvraag wordt beantwoord aan de hand van deze 2 PoC's.

Uit de 2 deelvragen was geconcludeerd dat het probleem die in de deelvragen werd gesteld te beantwoorden is met zowel Redis als Hazelcast. Maar welke van deze 2 technologieën is de betere keuze?

Hieruit Concludeer ik dat Redis de betere keuze is. De redenen hiervoor zijn:

- Redis is sneller en efficienter dan Hazelcast. Op een server met meer dan 30 beschikbare (virtuele) processoren is echter het tegenovergestelde hiervan waar. Voor een lading zoals die van SomToday heeft Redis er maar 2 tot 4 hiervan nodig. En sinds Somtoday al een groot marktaandeel heeft zou Redis eventuele groei van Somtoday goed aan moeten kunnen.

- Persoonlijk vond ik het fijner om met Redis te werken dan met Hazelcast. De voornaamste reden hiervoor was dat Redis betere documentatie had. Verder is het mij bijvoorbeeld niet gelukt om Hazelcast lokaal te benchmarken omdat de software hiervoor te verouderd was. Ook is het mij niet gelukt om mijn Hazelcast cluster te benaderen via de hazelcast-cli commandline omdat hier een netwerk service voor moet worden opgezet.

- Een voordeel van Hazelcast was de grafische gebruikersinterface (GUI). Echter heeft Redis ook een GUI beschikbaar die ik tijdens mijn onderzoek niet heb gebruikt, omdat ik hiervan destijds niet op de hoogte was. En ook omdat ik dit niet nodig heb gehad omdat ik alles via de redis-cli commandline kon doen. [21]

- Hazelcast werd mij bij het starten van dit project niet door Topicus aangeboden als mogelijke oplossing en Redis wel. Dit betekend dat ontwikkelaars bij Topicus Redis kennen, maar nog niet gebruiken. Hierom zou hun voorkeur eerder bij Redis liggen als oplossing dan Hazelcast. Dit is echter geen deal breaker als ik kan verantwoorden dat Hazelcast een betere keuze is. Echter beschouw ik zelf Redis ook als de betere keuze.

- Hazelcast word tegenover Redis veel minder gebruikt.

6. Procesmethode

Voor mijn afstudeerstage heb ik een planning[3] gemaakt. Hiervoor heb ik Projectlibre gebruikt, Hiermee kan je visueel (GANTT) planningen bijhouden. Ook heb ik voor de gehele duur van de afstudeerstage een urenregistratie[5] bijgehouden. En omdat ik met SCRUM heb gewerkt heb ik ook maandelijks een sprint retrospectives[6] gedaan.

6.1 Project fases

Opstartfase:
De eerste fase van mijn project was vooral gefocust op het begrijpen en analyseren van mijn opdracht. Met als doel een Plan van Aanpak[1] te kunnen schrijven. Hiervoor heb ik een Requirements analyse[2] en een gedetailleerde planning[3] gemaakt. Ook ben ik tijdens de opstartfase begonnen aan het Onderzoeksrapport[4].

De Opstartfase duurde ongeveer 25 werkdagen.

Uitvoeringsfase:
Tijdens de uitvoeringsfase heb ik mijzelf bezig gehouden met werken aan mijn opdracht. Het grootste gedeelte van deze fase is besteed het Onderzoeksrapport[4]. Hiervoor heb ik 2 PoC's gemaakt. Waarmee ik heb aangetoond hoe het probleem uit de probleemstelling verholpen kan worden. En ook welke technologieën hiervoor het best gebruikt kunnen worden.

Na dit onderzoek heb ik een Technisch ontwerp[9] geschreven over hoe mijn oplossing kan worden toegepast op SomToday ELO en Core. Vervolgens heb ik deze oplossing hierop toegepast. Ook ben ik in deze fase begonnen aan het concept van dit verslag.

De uitvoeringsfase duurde ongeveer 60 werkdagen.

Afrondingsfase:
In de laatste fase van mijn project ben ik bezig geweest met alles afronden. Dus het afmaken en overdragen van mijn afstudeeropdracht. Mijn eindpresentatie houden, mijn beoordelingsformulier laten invullen en dit document afronden.

Deze fase duurde ongeveer 25 werkdagen.

6.2 Organisatie

Deze opdracht had ook een organisatie element. Hieronder valt het op de hoogte houden van alle stakeholders. Om er achter te komen wie deze stakeholders precies zijn heb ik een stakeholderanalyse gedaan, te vinden in het Plan van Aanpak[1] onder ‘3.2 Stakeholder analyse’

Meetings:
Het eerste wat ik hiervoor heb gedaan is het inplannen van een vaste vergadering met mijn 2 bedrijfsbegeleiders. Het doel van deze vergadering is het bespreken van mijn progressie en het kunnen stellen van vragen. Hiervoor bereid ik vooraf altijd een agenda met punten die ik wil bespreken. Ook sloot ik de vergaderingen altijd af met het bespreken van mijn GANTT planning[3].

Ook heb ik een maandelijks vergaderingen gehad met mijn afstudeerdocent. Deze vergadering had hetzelfde doel als de wekelijkse vergaderingen met mijn bedrijfsbegeleiders.

Hiernaast heb ik dagelijkse stand-ups gehouden met het team waarin ik zat. Het doel van een stand-up bij Topicus is je team op de hoogte stellen wat je van plan bent te gaan doen die dag en optioneel ook wat je de vorige dag hebt gedaan. Op deze manier blijf je een beetje op de hoogte van waar je teamgenoten mee bezig zijn.

SCRUM:
Bij Topicus maken ze gebruik van Jira. Dit is een softwareprogramma waarmee op een Agile manier projecten bijgehouden kunnen worden. Hiervoor wordt een project opgedeeld in onderdelen genaamd user stories. Aan deze user stories worden vervolgens wegingen gegeven en geplaatst in een Jira board. Vanaf hier kunnen deze user stories worden opgepakt.

7. Kwaliteit

Omdat mijn opdracht vooral bestaat uit het uitvoeren van een onderzoek waarvoor ik veel configuratie bestanden heb opgeleverd, denk hierbij aan yaml, xml en ps1 bestanden. Heb ik naast gebruik te maken van de Topicus Coding standard, niet veel te maken gehad met de standaard waarborging van kwaliteit via bijvoorbeeld unit tests of functionele tests.

Wel wordt alles wat aan Somtoday wordt toegevoegd minimaal gecontroleerd door 2 developers bij Topicus. Deze developers moeten, voor dat er code toegevoegd kan worden aan de Somtoday codebase, de nieuwe code goedkeuren in de vorm van een vier ogen controle en het geven van een vinkje.

Een manier hoe ik de kwaliteit van mijn product heb gewaarborgd is via een onderzoek naar de lading van Somtoday en of mijn PoC hiermee overweg kan. Hiervoor heb ik een lading gesimuleerd die vergelijkbaar is met die van Somtoday. Het volledige onderzoek hiervan is te vinden in het Onderzoeksrapport[4] onder ‘Prestaties van het Redis cluster’

Om er achter te komen hoe een lading van Somtoday er uit ziet heb ik gekeken naar de hoeveelheid requests die zijn gestuurd in een van de drukste uren van die afgelopen maand. Uit dit onderzoek werd duidelijk dat:

- Op heel Somtoday zijn op piekmomenten maximaal 40000 clients (gebruikers) online. Deze gebruikers waren verdeeld over meerdere pods. En stuurde over deze periode van 45 minuten gemiddeld 2085 http requests per seconde. Hier moet er wel rekening mee gehouden worden dan dit over heel Somtoday is. En dat niet voor elke request bijvoorbeeld een nieuwe pagina geladen hoeft te worden.

- Tijdens deze periode werden er gemiddeld 354 requests per seconde gestuurd van Somtoday ELO en gemiddeld 833 requests per seconde gestuurd van Somtoday Core.

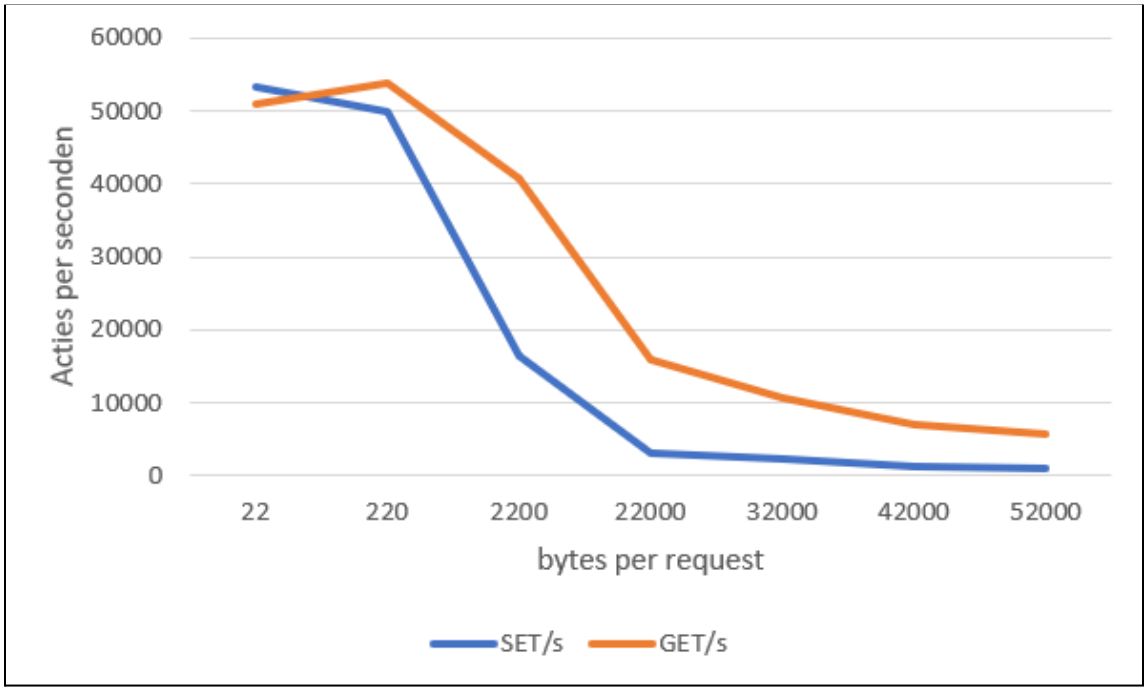
- Pagina’s groottes van Somtoday liggen tussen de 30kb en 70kb met enkele uitschieters rond de 200kb. Deze worden los opgeslagen. De pagestore zal dus steeds groter groeien (maximaal 20 pagina’s) maar http requests zullen nooit groter zijn dan de grote van een enkele pagina.

- De sessionstore van Somtoday ligt per sessie tussen de 5kb en 60kb en word steeds overschreven.

- Het interne cluster van Somtoday heeft 10 Gigabit internet switches/lijnen. Dit zit rond de 2 GB/s als het erg druk is. Dus netwerksnelheden zouden dus geen beperkende factor moeten zijn.
- Op de server van Somtoday is 1.5 tb aan werkgeheugen (RAM) beschikbaar. Wicket gebruikt al RAM voor de pageStore en sessieStore, dus dit zou hier geen impact op moeten hebben.

Met deze informatie heb ik een benchmark gedraaid op het Redis PoC met de redis-benchmark tool. De benchmark is gedraaid met 1, 2 en 4 processor kernen. De onderstaande benchmark was met 2 processorkernen uitgevoerd.

Redis-benchmark				
Parallelle clients	**bytes/request**	**Aantal requests**	**SET/seconden**	**GET/seconden**
10000	22	5000	53191	51020
10000	220	5000	50000	53763
10000	2200	5000	16447	40650
10000	22000	5000	3227	15974
10000	32000	5000	2351	10660
10000	42000	5000	1317	6896
10000	52000	5000	1042	5617



Het component van Somtoday dat het meeste data te verwerken heeft is Somtoday Core. Hier werden tijdens een drukke periode gemiddeld 833 requests per seconde gestuurd. Hiervan hoeft niet perse elke request het laden of plaatsen van een nieuwe pagina te zijn, wat de meest zware requests zijn.

In het uiterste geval zijn dit zo'n 30% tot 40% van de requests. Als de gemiddelde pagina grootte tussen de 30kb en 70kb liggen is dit 17500kb of tewel 17mb per seconde. Uit de resultaten van de 1 processorkern benchmark kan dit nogal krap zijn. Echter als Redis toegang krijgt tot 2 processorkernen, wat de snelheid bijna verdubbeld, zou dit ruim moeten kunnen. Dus kan geconcludeerd worden dat het PoC de Somtoday lading aan kan.

8. Eisen en requirements

Aan de start van dit project zijn er samen met Topicus een aantal requirements voor deze opdracht vastgelegd. Deze requirements zijn geschreven als user stories[14] en vervolgens geanalyseerd de aan de hand van de MoSCoW-methoden[13].

User Requirements:

Must have	Should have	Could have	Won't have

ID	Requirement	MoSCoW
U01	Als gebruiker van Somtoday Core wil ik er niks van merken als een node van Somtoday Core herstart.	Must have
U02	Als gebruiker van Somtoday ELO wil ik er niks van merken als een node van Somtoday ELO herstart.	Must have

System Requirements:

Must have	Should have	Could have	Won't have

ID	Requirement	MoSCoW
S01	Als systeembeheerder van Somtoday Core wil ik dat web-sessies ergens centraal worden opgeslagen.	Must have
S02	Als systeembeheerder van Somtoday Elo wil ik dat web-sessies ergens centraal worden opgeslagen.	Must have
S03	Als systeembeheerder van Somtoday Core wil ik dat webpagina's ergens centraal worden opgeslagen.	Must have
S04	Als systeembeheerder van Somtoday ELO wil ik dat webpagina’s ergens centraal worden opgeslagen.	Must have
S05	Als systeembeheerder van Somtoday Core wil ik dat gebruiker sessies van Somtoday Core een crash kunnen overleven.	Must have
S06	Als systeembeheerder van Somtoday ELO wil ik dat gebruiker sessies van Somtoday ELO een crash kunnen overleven.	Must have
S07	Als systeembeheerder van Somtoday Core wil ik dat gebruiker sessies van Somtoday Core een herstart kunnen overleven.	Must have
S08	Als systeembeheerder van Somtoday ELO wil ik dat gebruiker sessies van Somtoday ELO een herstart kunnen overleven.	Must have

S09	Als systeembeheerder van Somtoday wil ik dat de opgeleverde oplossing niet kan verhinderen dat Somtoday later omgezet kan worden naar een cloud omgeving.	Must have
S10	Als systeembeheerder van Somtoday ELO & Core wil ik dat tests worden afgehandeld via unit tests.	Could have
S11	Als systeembeheerder van Somtoday Core wil ik dat gebruikers van Somtoday Core onopgemerkt kunnen migreren naar een nieuwe node.	Could have
S12	Als systeembeheerder van Somtoday ELO wil ik dat gebruikers van Somtoday ELO onopgemerkt kunnen migreren naar een nieuwe node.	Could have

Business Requirements:

Must have	Should have	Could have	Won't have

ID	Requirement	MoSCoW
B01	Als Topicus wil ik dat Somtoday ELO geupdate kan worden zonder dat de gebruikers hier last van hebben.	Should have
B02	Als Topicus wil ik dat Somtoday Core geupdate kan worden zonder dat de gebruikers hier last van hebben.	Should have
B03	Als Topicus wil ik dat Somtoday ELO geherstart kan worden zonder dat de gebruikers hier last van hebben.	Should have
B04	Als Topicus wil ik dat Somtoday Core geherstart kan worden zonder dat de gebruikers hier last van hebben.	Should have

9. Schets Somtoday met persisted sessies:

Mijn opdracht is het onderzoeken hoe Somtoday de productieomgeving aan kan passen zonder dat de gebruikers daar last van hebben. Voor deze opdracht wordt het bestaande Somtoday landschap aangepast. Dit maakt het een brownfield project.

Op dit moment worden bij productie onderhoud aan Somtoday alle gebruikers uitgelogd, ook bij kortdurend onderhoud zoals het toevoegen van extra verwerkingscapaciteit. Om de dienstverlening te verbeteren naar haar klanten wil Somtoday de productieomgeving kunnen onderhouden terwijl de gebruikers door kunnen werken zonder uitgelogd te worden.

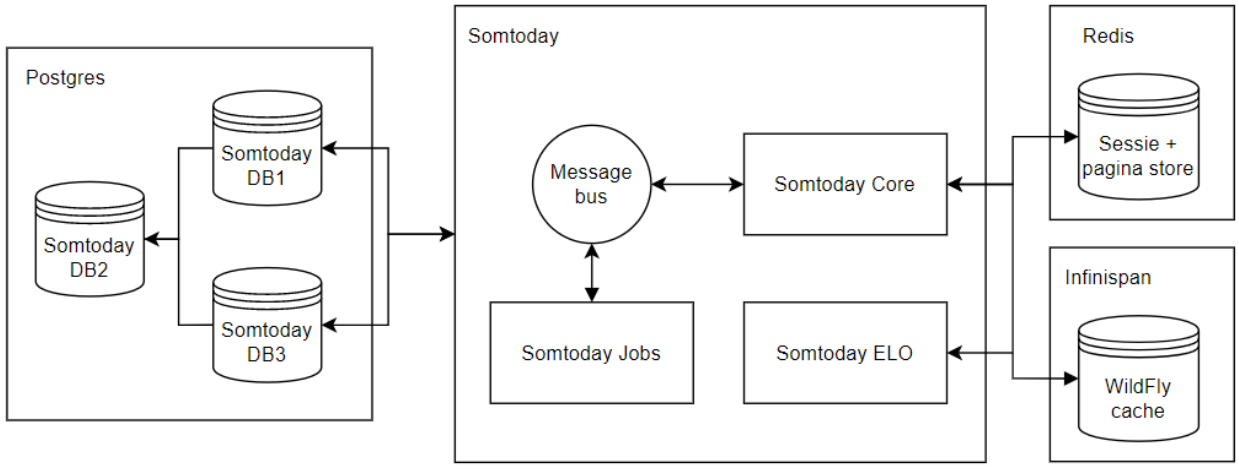
In het *Onderzoeksrapport[4]* is geconcludeerd dat de probleemstelling te beantwoorden is met het toepassen van een in-werkgeheugen dataopslag (in-memory datastore). Die gebruikt kan worden als een sessie-cache voor Apache Wicket en WildFly, het framework waar Somtoday in is geschreven en de applicatieserver van Somtoday.

Hiervoor wordt het cluster waar Somtoday op staat uitgebreid met 2 nieuwe databases. Dit zijn Redis en Infinispan.

Verder moet Somtoday worden aangepast om gebruik te maken van een Wicket PageStore en Wicket SessionStore in Redis.

Ook moet Somtoday zo geconfigureerd worden dat het zijn WildFly cache behoudt in de Infinispan database. Hiervoor moet Somtoday geconfigureerd worden om gebruik te maken van WildFly in High Availability mode in plaats van Standalone mode.

9.1 Infrastructuur diagram Somtoday met persisted sessies:



10. Implementatie en verklaring requirements

In dit hoofdstuk wordt een inzicht gegeven in hoe aan alle requirements wordt voldaan. Alle onderstaande requirements zijn te vinden in het *Requirements Afstudeeropdracht Topicus[2]* document.

Omdat deze opdracht bestaat het uitvoeren van een onderzoek waarbij het product is gerealiseerd aan de hand van het configureren van een bestaand framework en applicatieserver, bevat deze opdracht geen onderdelen zoals het uitwerken van een frontend, backend of API. Hierom word dit hoofdstuk over de implementatie van deze nieuwe functionaliteiten behandeld aan de hand van de gestelde requirements en wat hiervoor is toegevoegd.

S01 & S02 - Het centraal beheren van web-sessies.

Requirement(s):

- Als systeembeheerder van Somtoday Core wil ik dat Websessies ergens centraal worden opgeslagen.
- Als systeembeheerder van Somtoday ELO wil ik dat Websessies ergens centraal worden opgeslagen.

Onder de bovenstaande requirements valt het lostrekken van de web-sessies uit de interne opslag van Apache Wicket, dit is het framework waar Somtoday gebruik van maakt.

Websessies in Redis:
In het *Onderzoeksrapport[4]* is geconcludeerd dat hiervoor het beste een in-werkgeheugen data opslag gebruikt kan worden. Hierin kunnen vervolgens de web-sessies worden opgeslagen. De meest geschikte technologie hiervoor is Redis.

Redis heeft geen out of the box compatibiliteit met de Java-programmeertaal. Om dit toe te voegen wordt de Maven dependency Jedis gebruikt. Hiermee kan er via het RESP protocol[24] een connectie met Redis gelegd worden. Deze communicatie is beveiligd via een ingesteld wachtwoord.

Vervolgens is er een opmaat implementatie gemaakt van de wicket SessionStore die Redis als backend gebruikt waarmee de standaard Wicket sessionStore wordt overschreven. Hierdoor worden de web-sessies niet meer opgeslagen in Wicket zelf maar in Redis als base64 String.

Voor deze implementatie worden 2 nieuwe bestanden toegevoegd. Dit zijn:

- RedisSessionStore.java
    - Dit is de class die gebruikt wordt in plaats van de Wicket SessionStore. Deze class bevat alle methoden die de Wicket ISessionStore interface nodig heeft.
- RedisCache.java
    - RedisCache wordt gebruikt door RedisSessionStore om sessie-data op te slaan in Redis.

Redis in Somtoday:
Deze Redis dataopslag wordt vervolgens in hetzelfde Kubernetes cluster geplaatst waar Somtoday ook in staat. Om Redis schaalbaar te maken wordt het toegevoegd als StatefulSet. Dit is een Kubernetes storage service die bestaat uit 1 source volume en 2 (of meer) replica volumes. De 2 replica volumes zijn verantwoordelijk voor het opslaan van data en de source volume verantwoordelijk voor het ophalen van data uit de 2 replica volumes. Dit wordt gedaan om met de hoeveelheid data die Somtoday genereert om te kunnen gaan.

Om Redis toe te voegen aan het Somtoday worden de volgende bestanden gebruikt:

- redis-storage-class.yml
    - Een opslagklasse is gekoppeld aan een provisioner, een plug-in die schijfruimte kan reserveren of volumes kan kopen bij een cloudprovider.
- redis-persistent-volume.yml
    - Persistente volumes worden gebruikt om een opslagruimte te creëren. In dit geval een Redis-StatefulSet met drie pods (1 source en 2 replica's).
- redis-config.yml
    - Bevat configuratie informatie over de Redis-StatefulSet zoals het wachtwoord.
- redis-statefulset.yml
    - StatefulSet is een Kubernetes-object dat wordt gebruikt om stateful-toepassingen te deployen. In dit geval Redis.

S03 & S04 - Het centraal beheren van web pagina's.

Requirement(s):

- Als systeembeheerder van Somtoday Core wil ik dat webpagina's ergens centraal worden opgeslagen.
- Als systeembeheerder van Somtoday ELO wil ik dat webpagina's ergens centraal worden opgeslagen.

Onder de bovenstaande requirements vallen het lostrekken van de web pagina's uit de interne opslag van Apache Wicket, dit is het framework waar Somtoday gebruik van maakt.

Webpagina's in Redis:
In het *Onderzoeksrapport[4] is* onderzocht wat de beste oplossing hiervoor is. Net als bij de requirements S01 en S02 is geconcludeerd dat het opslaan hiervan in Redis de beste oplossing biedt.

Om dit toe te passen word de Maven dependency 'wicketstuff-datastore-redis' van Wicketstuff toegepast. Hiermee wordt een nieuwe DefaultPageManagerProvider Java class aangemaakt in Somtoday die gebruik maakt van een implementatie van de Wicket IPageStore die Redis als backend gebruikt. Door dit toe te passen in de installatiemethode van Wicket worden hiermee de webpagina's niet meer opgeslagen in Wicket zelf maar in Redis.

De Maven dependency 'wicketstuff-datastore-redis' voegt 3 bestanden toe:

- RedisDataStore.class
    - Dit bestand wordt gebruikt om de standaard Wicket IPageStore mee te overschrijven. Deze klasse bevat alle methodes die de standaard IPageStore gebruikt maar het gebruikt Redis als backend in plaats van Wicket.
- RedisSettings.class
    - Dit bestand bevat een aantal parameters die noodzakelijk zijn om met Redis te verbinden zoals de poort en hostnaam.
- IRedisSettings.class
    - Dit is de interface klasse van RedisSettings.

De Redis dataopslag die hiervoor wordt gebruikt is hetzelfde als die voor de requirements S01 en S02 en maakt ook gebruik van de Maven dependency Jedis.

De Wicketstuff-datastore-redis dependency is naast Jedis ook afhankelijk van 2 andere Maven dependencies die toegevoegd moeten worden aan Somtoday dit zijn:

- slf4j-api
- slf4j-simple
    - Deze 2 dependencies worden gebruikt om berichten uit te lezen in de terminal.

S05 & S06 & S07 & S08- Het overleven van een crash of herstart.

Requirement(s):

- Als systeembeheerder van Somtoday Core wil ik dat gebruiker sessies van Somtoday Core een crash kunnen overleven.
- Als systeembeheerder van Somtoday ELO wil ik dat gebruiker sessies van Somtoday Elo een crash kunnen overleven.
- Als systeembeheerder van Somtoday Core wil ik dat gebruiker sessies van Somtoday Core een herstart kunnen overleven.
- Als systeembeheerder van Somtoday ELO wil ik dat gebruiker sessies van Somtoday ELO een herstart kunnen overleven.

Alle bovenstaande requirements hebben te maken het het persisteren van gebruikerssessies.

Het identificeren van sessies:
Elke pod draait zijn eigen instantie van Wicket. In S01, S02, S03 en S04 zijn de web-sessies en webpagina's al losgetrokken van Wicket. Maar alleen dit is niet voldoende om ervoor te zorgen dat een gebruiker van Somtoday zijn sessie kan blijven gebruiken als zijn onderliggende pod word afgesloten. De applicatie moet hiervoor weten welke sessie-data bij welke gebruiker hoort. Anders krijgt de gebruiker op een nieuwe pod gewoon een nieuwe sessie, waardoor de gebruiker wordt uitgelogd.

Om dit te voorkomen wordt een sessie identificatie cookie gebruikt. Hiervoor kan de standaard JSESSIONID cookie gebruikt worden. Hiermee wordt het standaard Wicket sessie identificatie gedrag aangepast om eerst te controleren of een gebruiker al een JSESSIONID cookie in zijn browser heeft staan. Om dit toe te passen word de getSessionId methode van het RedisSessionStore.java bestand aangepast. Hoe dit precies wordt gedaan is te vinden in het Onderzoeksrapport[4] in het hoofdstuk ‘3.5.4.1 Redis’ onder ‘Hergebruiken van sessies:’.

Vervolgens kan hiermee een gebruiker aan zijn sessie gekoppeld worden. Ook als zijn onderliggende pod wordt afgesloten. Hierdoor wordt de gebruiker niet uitgelogd als zijn pod crashed of herstart.

WildFly's sessie-data:
Voor Somtoday is dit echter nog niet alles. Omdat Somtoday gebruik maakt van de applicatieserver WildFly bevat deze applicatie server ook sessie-data. En omdat elke pod een eigen instantie van WildFly draait gaat bij een herstart deze data verloren.

Voor deze requirement is het echter genoeg om WildFly in High Availability (HA) mode te zetten. Dit geeft de optie om distributed session aan te zetten waardoor WildFly pods hun sessies met elkaar kunnen delen. Hoe dit precies wordt gedaan is te vinden in het Onderzoeksrapport[4] in het hoofdstuk '3.6.2 WildFly persistent sessions'.

Om WildFly in HA mode te zetten moeten de volgende stappen ondernomen worden:

- Dockerfile
    - De huidige Dockerfile van Somtoday wordt aangepast om in plaats van WildFly in Standalone mode, WildFly in HA mode te gebruiken
- web.xml
    - Hierin is boven de </web-app> tag, de <distributable/> toegevoegd. Dit maakt websessie verdeelbaar onder pods die WildFly draaien.

Ook worden er een aantal bestanden toegevoegd:

- config-server.cli
    - Dit bestand bevat commando's die gebruik maken van de WildFly CLI en wordt gebruikt om de server te configureren tijdens het opstarten.
- beans.xml
    - Het beans.xml bestand is de 'bean-archief descriptor' voor CDI-toepassingen. Het kan worden gebruikt voor elke CDI-compatibele container, zoals die van WildFly.
- rbac.yml
    - Met dit bestand worden een ServiceAccount, ClusterRole en ClusterRoleBinding geconfigureerd in Kubernetes waar WildFly gebruik van maakt.
- standalone-full-ha.xml
    - Dit is het configuratie bestand van de HA WildFly server.

Hierdoor verliest een gebruiker niet zijn WildFly sessie-data zolang 1 pod van die applicatie nog bestaat. Dit in combinatie met de 'rolling updates' mogelijkheid die Kubernetes biedt waarbij gegarandeerd wordt dat een sessie nooit verloren gaat. Een rolling update werkt namelijk door de pods 1-voor-1 te vervangen en pas met de volgende verder te gaan zodra de vorige up en ready is.

U01 & U02 - Het herstarten van complete Kubernetes nodes zonder dat gebruikers hier last van hebben.

Requirement(s):

- Als gebruiker van Somtoday Core wil ik er niks van merken als een node van Somtoday Core herstart.
- Als gebruiker van Somtoday ELO wil ik er niks van merken als een node van Somtoday ELO herstart.

Het verschil tussen deze requirements en S05, S06, S07 en S08 is het kunnen herstarten van de hele applicatie node tegenover enkele pods. Hierdoor zijn tijdelijk alle pods van een applicatie afgesloten.

Aan deze requirement is al te voldoen door het extern trekken van de Wicket websessies en webpagina's naar Redis. Echter moet hiervoor wel de configuratie van de WildFly applicatieserver aangepast worden.

Om te voorkomen dat WildFly zijn sessie-data verliest als er geen pods beschikbaar zijn kan in WildFly een externe cache worden ingesteld met het 'standalone-full-ha.xml' bestand. Hiervoor moet een een distributable-web.xml bestand worden toegevoegd in de 'WEB-INF' folder. Dit bestand laat WildFly weten dat de sessie uit de externe cache gebruikt moet worden.

Voor deze externe cache moet Infinispan gebruikt worden. Dit is ook een in-werkgeheugen data opslag net als Redis.

Het opzetten hiervan is alleen niet makkelijk in Windows. De reden hiervoor is dat de traditionele manier van databases opzetten in Kubernetes via een yaml 'StatefulSet Operator' niet meer ondersteund is na Infinispan 9. Na deze versie van Infinispan moet dit gedaan worden via een 'Infinispan Operator'. Deze operator moet worden geïnstalleerd via de 'infinispan-cli kubectl plugin'. Deze plugin is echter alleen beschikbaar voor Linux en Darwin. En omdat WildFly gebruik maakt van Infinispan 13 kan er dus geen Infinispan Kubernetes cluster worden opgezet via 'yaml'. Ook kan WildFly niet worden aangepast hiervoor omdat remote clusters niet op lagere versies worden ondersteund.

Hierom is dit voor het opzetten hiervan met een omweg getest. Als deze oplossing op Somtoday wordt toegepast zal dit opgezet moeten worden met Linux.

Hoe Infinispan voor deze test opgezet is en hoe WildFly ingesteld kan worden om gebruik te maken van een externe cache is te vinden in het onderzoeksrapport[4] in het hoofdstuk '3.6.5 WildFly sessies in een extern Infinispan cluster' onder Het configureren van WildFly:

Als dit is ingesteld kan een complete applicatie node waar een gebruiker op zit worden herstart zonder dat deze gebruikers zijn sessie-data verliest en uitgelogd wordt.

B03 & B04 - Het onopgemerkt kunnen herstarten.

Requirement(s):

- Als Topicus wil ik dat Somtoday ELO geherstart kan worden zonder dat de gebruikers hier last van hebben.
- Als Topicus wil ik dat Somtoday Core geherstart kan worden zonder dat de gebruikers hier last van hebben.

Deze requirement gaat over wat de gebruiker ervaart als zijn achterliggende pod herstart. En deze requirement bevat 2 mogelijke verschillende situaties.

Situatie 1:
De eerste situatie is het herstarten van een gebruikers achterliggende pod terwijl er nog ander pods beschikbaar zijn in de node. Hierdoor zal de gebruiker worden verplaatst naar een andere al bestaande pod waar hij vervolgens door de voorgaande requirements zijn sessie kan blijven gebruiken.

In deze situatie zal de gebruiker er waarschijnlijk niks van merken. Het overschakelen naar een andere pod duurt enkele milliseconden.

Situatie 2:
De tweede situatie is het herstarten van een hele node. In deze situatie zullen tijdelijk alle pods afgesloten zijn. Hierdoor is de applicatie niet meer bereikbaar. De herstart tijd hiervan is de tijd die het kost voor WildFly om op te starten. Dit is 1 a 2 minuten.

In deze tijd zullen gebruikers van Somtoday een laadscherm of een ‘adres niet bereikbaar’ scherm zien. Hierna kunnen deze gebruikers gewoon verder zonder uitgelogd te worden of hun ingevulde data te verliezen. Aan deze opstarttijd is niks te doen, echter is het hierdoor niet helemaal ‘onopgemerkt’ te noemen.

S11 & S12 & B01 & B02- Het onopgemerkt kunnen updaten en migreren naar een nieuwe node.

Requirement(s):

- Als systeembeheerder van Somtoday Core wil ik dat gebruikers van Somtoday Core onopgemerkt kunnen migreren naar een nieuwe node.
- Als systeembeheerder van Somtoday ELO wil ik dat gebruikers van Somtoday ELO onopgemerkt kunnen migreren naar een nieuwe node.
- Als Topicus wil ik dat Somtoday ELO geupdate kan worden zonder dat de gebruikers hier last van hebben.
- Als Topicus wil ik dat Somtoday Core geupdate kan worden zonder dat de gebruikers hier last van hebben.

Deze requirements betekenen dat wanneer een pod uitgezet moet worden door bijvoorbeeld een update van Somtoday, dat de gebruikers hierop gewoon door kunnen werken.

Deze requirement is getest op het PoC dat is ontwikkeld voor het *Onderzoeksrapport[4].* Voor deze test is er wat code aangepast in het PoC en is deze herstart om de nieuwe aanpassingen toe te voegen.

Bij deze herstart wordt een nieuwe Kubernetes deployment aangemaakt, en wordt de oude verwijderd. Het PoC is zo geconfigureerd dat tijdens de deployment stap een random poort wordt gebruikt.

Het probleem hiermee is dat het poortnummer onderdeel is van de url die word gebruikt om het PoC te benaderen. Hierdoor moet dus het oude browservenster worden afgesloten.

Als nu het PoC geopend wordt via de nieuwe url hoeft dankzij de JSESSIONID cookie de gebruiker niet opnieuw in te loggen. Ook behoudt de gebruiker zijn WildFly sessie-data. Maar omdat het oude browservenster was afgesloten door de url verandering bevindt de gebruiker zich op een nieuwe pagina.

Op Somtoday zou dit echter geen probleem moeten zijn omdat Somtoday niet van url verandert bij een versie update. En hierdoor zou de gebruiker zich ook niet op een nieuwe pagina bevinden.

De applicatie is niet bereikbaar voor 1 a 2 minuten omdat WildFly moet herstarten. De gebruiker zal hier weinig van merken met uitzondering van die 1 a 2 minuten. Echter als de url waar die gebruiker zich op zit wordt aangepast zal hij hierdoor zich wel op een nieuwe pagina bevinden.

S09 - De oplossing moet geschikt zijn voor een cloud omgeving.
Requirement(s):

- Als systeembeheerder van Somtoday wil ik dat de opgeleverde oplossing niet kan verhinderen dat Somtoday later omgezet kan worden naar een cloud omgeving.

In het *Onderzoeksrapport[4] i*s onderzocht welke technologie het meest geschikt zou zijn om de hoofdvraag ‘Hoe kan het beste de sessie-data los worden getrokken van de individuele Kubernetes pods voor de applicatie Somtoday ELO en Somtoday Core’ te beantwoorden. Hierbij is voor elke technologie onderzocht of het geschikt is voor een cloud omgeving.

Omdat Redis en Infinispan beiden als clusters in Kubernetes gedraaid worden, kunnen deze technologieën ook als cluster in een cloud omgeving gedraaid worden.

S10 - Het implementeren van unit tests.
Requirement(s):

- Als systeembeheerder van Somtoday ELO & Core wil ik dat tests worden afgehandeld via unit tests.

Het schrijven van unit tests voor deze opdracht is lastig omdat alle functionaliteit gerelateerd is aan het herstarten van de applicatie. Het is wel een mogelijkheid om te testen of er verbinding gemaakt kan worden met de benodigde databases. En of een nieuwe sessie-data ook in deze databases terecht komt.

11. Resultaten en verantwoording gemaakte keuzes

Tijdens het uitvoeren van mijn afstudeeropdracht zijn er een aantal keuzes gemaakt. In dit hoofdstuk wordt hier dieper op ingegaan.

11.1 Opzet van het onderzoek:

Omdat mijn afstudeeropdracht een groot onderzoekend element bevat heb ik hiervoor een Onderzoeksrapport[4] opgezet. Hierin zijn een aantal keuzes gemaakt die bepalend waren voor de verloop van mijn onderzoek.

De eerste keuze die ik hierin heb gemaakt was het gebruik maken van de DOT onderzoeksmethode[7]. De reden dat ik hiervoor heb gekozen is omdat ik deze methode eerder heb gebruikt voor een onderzoeksrapport. Deze methode wordt gebruikt om een onderzoek te structureren en bevat nuttige sub methodes die tijdens het onderzoek kunnen worden gebruikt.

Een andere belangrijke keuze die ik in mijn onderzoeksrapport heb gemaakt is het gebruik maken van de Advanced Application Development (AAD) onderzoeksmethode. Dit is een onderzoeksmethode die ik heb geleerd te gebruiken tijdens mijn specialisatie. Deze methode bestaat uit het opzetten van een Longlist met meerdere technologieën waarmee een gesteld probleem verholpen kan worden. Deze technologieën worden vervolgens kort onderzocht of ze wel echt geschikt zijn.

Hieruit ontstaat een Shortlist met technologieën die verder onderzocht worden en aan de hand van een aantal criteria becijfert worden. Hierna wordt er met de 2 best scorende technologieën een PoC ontwikkeld. Dit is ook de reden waarom er 2 PoC’s zijn gemaakt. En vervolgens wordt er aan de hand van de PoC’s bepaalt welke technologie het beste geschikt is.

11.2 Sessie-data van de applicatie server:

Tijdens het uitbreiden van het gekozen PoC voor een presentatie had ik de applicatieserver Jetty vervangen voor WildFly. De reden hiervoor was omdat Somtoday ook WildFly gebruikt. En omdat WildFly gebruik kan maken van Jakarta Context and Dependency Injection[10] besloot ik te testen of dit werkte.

Hierdoor kwam ik erachter dat in WildFly ook sessie-data wordt bijgehouden die verloren gaat bij een herstart. Vervolgens heb ik onderzocht of dit probleem verholpen kan worden met een van de twee technologieën die ik had uitgewerkt. Echter kwam ik er achter dat beiden hiervoor niet zouden werken. En dat hiervoor alleen de technologie Infinispan gebruikt kan worden.

Na de oplossing met Infinispan te hebben uitgewerkt heb ik samen met mijn bedrijfsbegeleiders besloten om niet een eventueel 3de PoC te maken met alleen Infinispan, omdat ik hier niet meer genoeg tijd voor had. En met deze reden worden voor het uiteindelijke PoC 2 verschillende technologieën gebruikt.

12. Betrouwbaarheid en onderhoudbaarheid.

De uiteindelijke oplossing bevat 2 verschillende databases die gebruikt worden als caching servers. Dit zijn Infinispan en Redis. Ook wordt er bij de 2 PoC's die ik heb ontwikkeld een Istio load balancer gebruikt voor sticky sessions. Maar is dit allemaal wel goed te onderhouden?

Onderhoudbaarheid:
Tijdens het kiezen van Redis heb ik het beoordeeld op onderhoudbaarheid, Hiervoor scoorde het een 9. De reden hiervoor was:

- "Redis wordt door heel veel partijen gebruikt hierdoor is het van belang dat het uptodate blijft. Hierom brengt redis elk jaar een nieuwe (volledige) versie uit. Verder wordt er ook een Wicketstuff wicket-redis-integratie dependency gebruikt. Deze word ook meerdere keren per jaar geupdate, vaak kort nadat Redis is geupdate."

Ook wordt Infinispan gebruikt. Deze techologie viel af in de longlist fase in het onderzoeksrapport. Hiervoor heeft het nooit een cijfer gekregen voor onderhoudbaarheid. Echter brengt Infinispan ook jaarlijks nieuwe versies uit. Omdat de WildFly applicatieserver afhankelijk is van Infinispan wordt het ook door WildFly up-to-date gehouden. [26]

Ook is er voor beiden technologieen onderzocht of deze een naast een CLI benaderd kunnen worden met een dashboard. Voor Redis is dit Redis-insight en voor Infinispan is dit Infinispan Archive. Ook hebben beiden technologieën compatibiliteit met Promitheus. Dit is een oplossing die Topicus gebruikt voor het monitoren van applicaties.

Voor mijn 2 PoC's heb ik een Istio load balancer gebruikt voor sticky sessions. Deze loadbalancer was tijdens mijn onderzoek toegevoegd zodat ik geen onverwacht gedrag kreeg tijdens het testen en omdat dezelfde soort load balancer al wordt gebruikt in Somtoday. Echter nu het project is afgerond zijn sticky sessions niet meer nodig omdat de sessie-data niet meer afhankelijk is van de specifieke WildFly applicatieserver. Dit betekent ook dat Istio hiervoor niet meer nodig is op Somtoday, na het toepassen van mijn oplossing. En dat bespaard weer overhead en onderhoud.

Betrouwbaarheid:
Om er zeker van dat mijn oplossing goed te onderhouden en of ik alle nodige aspecten overwogen heb qua betrouwbaarheid heb ik mijn oplossing gepresenteerd aan het Topicus Infra team. Dit is het team binnen Topicus Onderwijs die het Kubernetes cluster onderhoud. En na afloop hiervan keer in de vraag wat er zou gebeuren als een van de caching servers uitvalt.

Beiden Redis en Infinispan bestaan uit een source server met meerdere replica servers. En beiden technologieën hebben hiermee de optie op een backup bij te houden van de data voor het geval een van de replica's uit valt. Echter als de source server uitvalt kunnen gebruikers niet meer hun sessie-data ophalen. Hierdoor zullen de gebruikers dus alsnog uitloggen. Met deze mogelijkheid moet dus rekening gehouden worden.

13. Conclusies en aanbevelingen

13.1 Conclusie op het proces

Tijdens dit project heb ik een planning [3] aangehouden. En aan het eind van elke week tijdens het progressie gesprekje met mijn bedrijfsbegeleiders rapporteerde ik ook over de voortgang hierop. Deze planning was echter wel meerdere keren aangepast tijdens de verloop van het project.

Initieel had ik geen goede inschatting gemaakt over hoe groot het onderzoekend gedeelte van deze opdracht was. Daarom had ik het originele tijdsbestek voor het schrijving van het onderzoeksrapport met 2 keer moeten verlengen. Een ander onderdeel dat onverwacht veel tijd heeft gekost was het onderzoeken hoe sessie-data losgetrokken kon worden van een WildFly applicatieserver. Ik kwam er namelijk pas achter dat dit noodzakelijk was toen ik mijn onderzoeksrapport al had afgerond.

Daarom had ik ook veel minder tijd voor het implementeren van mijn oplossing in Somtoday dan ik eigenlijk had ingepland. Dit is echter niet een heel groot probleem omdat het opleveren van een werkend PoC het meest belangrijke onderdeel is van deze opdracht. En het daadwerkelijk inbouwen in Somtoday meer een nice to have is. Tijdens het schrijven van deze conclusie is het implementeren hiervan ook nog niet afgerond.

13.2 Conclusie op het geleverde product:

Aan de hand van mijn Onderzoeksrapport[4] heb ik geconcludeerd dat de probleemstelling te beantwoorden is met het toepassen van een in-werkgeheugen dataopslag (in-memory datastore). Die gebruikt kan worden als een sessie-cache voor Apache Wicket, het framework waar Somtoday in is geschreven.

Dit is een database die werkgeheugen gebruikt voor het beheren van data. Hiervoor is gekozen omdat werkgeheugen veel sneller is dan traditionele verwerkingsmethodes. En de meer traditionele databases zijn vanwege hun verwerkingssnelheid niet geschikt voor het beheren van sessies. [11]

Ook hebben deze databases vaak betrouwbaarheid garanties op data (wanneer een transactie is gepleegd, staat deze gegarandeerd in de database). Dit is handig voor cruciale data maar dit soort checks maken het trager dan bijvoorbeeld Redis. [12]

Wicket sessie-data persistentie met Redis:
Aan de hand van de AAD onderzoeksmethoden is bepaald dat de meest geschikte technologie hiervoor Redis is.

Deze technologie kan gebruikt worden voor het onafhankelijk maken en behouden van sessie-data van de applicatie. Deze data bestaat uit de Wicket PageStore en de Wicket SessionStore. Dit kan worden gedaan door op maat implementaties van deze onderdelen toe te passen die Redis gebruiken als backend. En de standaard Wicket datastores hiermee te overschrijven.

Als de sessie data onafhankelijk van de applicatie pods is gemaakt kan door het hergebruiken van de sessie identificatie cookie een gebruiker van deze zelfde sessie gebruik blijven maken. Ook als de pod waarop hij bezig was wordt afgeschaald. Deze cookie bevindt zich in de browser van de gebruiker.

WildFly sessie-data persistentie met Infinispan:
Ook moet moet de Standalone WildFly applicatieserver van Somtoday worden uitgewisseld naar een High Availability WildFly applicatieserver. Deze applicatieserver moet vervolgens zo geconfigureerd worden dat het zijn sessie-data beheerd in een externe Infinispan database.

Advies voor mogelijk vervolgonderzoek:
Dit rapport bevat een onderdeel dat ik nog verder zou willen onderzoeken als ik meer tijd had. Dit is het gebruiken van 2 verschillende databases. Pas laat in het onderzoek bleek dat er ook sessie-data in de applicatieserver van Somtoday werd bijgehouden. En om deze sessie-data hiervan los te trekken kan alleen Infinispan worden gebruikt.

Daarom zou in een vervolgonderzoek gekeken kunnen worden of Infinispan ook geschikt is voor het beheren van de Wicket pageStore en sessionStore. De reden hiervoor is dat het gebruik van 2 verschillende databases een hoop complexiteit en onderhoud met zich mee brengt.

Eventueel zou WildFly ook uitgebreid kunnen worden via een externe module. Omdat WildFly ingesteld kan worden om persisted sessies te gebruiken via Infinispan, zou dit ook eventueel met Redis moeten kunnen. Hiervoor zou een module gemaakt kunnen worden die hetzelfde werkt als die van Infinispan. Maar dan moet alle Infinispan logica vervangen worden door Redis logica. Omdat WildFly open-source software is kan deze informatie gewoon verkregen worden.

14. Reflectie tot afstudeertraject - Rob van Heuven

Dit is een inhoudelijke reflectie met een focus op het uitgevoerde afstudeerproject.

Tijdens mijn afstudeertraject heb ik te maken gehad met veel voor mij nieuwe technologieën. Dit was voor mij niet geheel onverwachts omdat mij tijdens het intake gesprek bij Topicus al was verteld welke technologieën er gebruikt zouden worden. Een paar voorbeelden waarmee ik nog nooit eerder had gewerkt zijn Apache Wicket, Kubernetes, WildFly en databases zoals Redis.

Het was een vrij ingewikkelde opdracht, waarvoor ik onder andere een bestaand framework heb aangepast. Een opdracht die wat verder ging dan een normale opdracht waarvoor je een frontend, backend en API maakt. Dit is ook de voornaamste reden waarom ik in samenwerking met Topicus in mijn Definition of Done mijn doel als volgt geformuleerd heb.:

- ‘Mijn doel voor dit project is om te onderzoeken wat het probleem is en hoe dit het beste verholpen kan worden. Als dit is gedaan wil ik een werkend proof of concept opleveren die Topicus als basis kan gebruiken voor productie.’

Midden april dacht ik dit doel al te hebben bereikt. Ik had hiervoor al 2 proof of concepts gemaakt waarin ik de Wicket sessie-data had los getrokken. Een met Redis als uitgangspunt en een met Hazelcast als uitgangspunt. En ik had ook al mijn onderzoeksrapport afgerond. Omdat mijn afstudeerproject nog ongeveer 2 maanden duurde besloot ik te beginnen aan het plaatsen van mijn oplossing in Somtoday. Maar voordat ik hieraan kon beginnen wilde ik eerst mijn oplossing presenteren aan het team dat het Somtoday Kubernetes cluster en de infrastructuur hiervan onderhoud, voor wat feedback. Tijdens het uitbreiden van mijn PoC voor deze presentatie ontdekte ik dat het sessie-data lostrekken van de individuele Kubernetes pods nog een derde onderdeel bevatte. Dit is de sessie-data die in de applicatieserver WildFly wordt vastgehouden.

Hierom heb ik mijn onderzoeksrapport uitgebreid met een extra onderzoek om dit probleem op te lossen. Tijdens dit onderzoek kwam ik er alleen achter dat dit probleem niet opgelost kon worden met de technologieen Redis of Hazelcast. Hiervoor moest Infinispan gebruikt worden. Een technologie die in mijn onderzoek niet eens de shortlist had gehaald omdat het bijna niet gebruikt wordt. Dit is jammer want het maakt mijn oplossing complexer. Mijn PoC voldoet hierdoor wel aan alle gestelde requirements.

Tijdens het schrijven van deze reflectie ben ik nog bezig met het implementeren van mijn oplossing in het complexe Somtoday landschap. Als ik geen grote tegenslagen ondervindt zal mij dit in de komende maand nog wel moeten lukken.

Ik ben dus nog niet helemaal klaar maar ik vind dat ik best trots mag zijn op het tot nu toe geleverde resultaat.

15. Bronvermelding:

Deze bronnenlijst is geschreven volgens de IEEE-richtlijnen [8].

[1]R. van Heuven, "Plan van Aanpak", *Google Docs*, 2022. [Online]. Available: https://docs.google.com/document/d/1J49dKuV5jDDycFmAs_gQeG1CPczcV0wy-MS_9VxM1Hw/edit?usp=sharing. [Accessed: 28- Mar- 2022].

[2]R. van Heuven, "Requirements Afstudeeropdracht Topicus", *Google Docs*, 2022. [Online]. Available: https://docs.google.com/document/d/1Y7Pb-oAKC8s5Tn86_-ecJ0FRiLiqRfb7rhAbHIM8Woo/edit?usp=sharing. [Accessed: 28- Mar- 2022].

[3]R. van Heuven, "GANTT Planning - Afstudeeropdracht Topicus.pdf", *Google Docs*, 2022. [Online]. Available: https://drive.google.com/file/d/1kQkP82DmbWMG_DlgYM0JtYPCH7RKfYP6/view?usp=sharing. [Accessed: 28- Mar- 2022].

[4]R. van Heuven, "Onderzoeksrapport", *Google Docs*, 2022. [Online]. Available: https://docs.google.com/document/d/1fp1bNRiLq9XjDoFsLdFtd-YqERmnFYSoqFxgzdSOiUw/edit?usp=sharing. [Accessed: 28- Mar- 2022].

[5]R. van Heuven, "Urenregistratie", *Google Docs*, 2022. [Online]. Available: https://docs.google.com/document/d/1MSCAPfu6UqcG_DDGkW5cIe3JhNhh7hym1Gh3f4J44OY/edit?usp=sharing. [Accessed: 28- Mar- 2022].

[6]R. van Heuven, "Eind sprint reflecties", *Google Docs*, 2022. [Online]. Available: https://docs.google.com/document/d/1PkwX_XMSKHHn3FIX2c9s9T-fjCOTu-hSHFiZ0hmvxWA/edit?usp=sharing. [Accessed: 28- Mar- 2022].

[7]"Methods - ICT research methods", *Ictresearchmethods.nl*, 2022. [Online]. Available: https://ictresearchmethods.nl/Methods. [Accessed: 06- Apr- 2022].

[8]"Save Time and Improve your Marks with CiteThisForMe, The No. 1 Citation Tool", *Cite This For Me*, 2022. [Online]. Available: https://www.citethisforme.com/. [Accessed: 10- May- 2022].

[9]R. van Heuven, "Technisch Ontwerp", *Docs.google.com*, 2022. [Online]. Available: https://docs.google.com/document/d/1cRKZ8F4F0RvTwUpL3K8FfxA5ayJn2I2wvS1sUnZ4YGc/edit?usp=sharing. [Accessed: 11- May- 2022].

[10]C. Guindon, "Jakarta Context Dependency Injection 3.0 | The Eclipse Foundation", *Jakarta® EE: The New Home of Cloud Native Java*, 2022. [Online]. Available: https://jakarta.ee/specifications/cdi/3.0/. [Accessed: 24- May- 2022].

[11]"When to use a key/value store such as Redis instead/along side of a SQL database?", *Stack Overflow*, 2012. [Online]. Available: https://stackoverflow.com/questions/7535184/when-to-use-a-key-value-store-such-as-redis-instead-along-side-of-a-sql-database. [Accessed: 24- May- 2022].

[12]"Postgres Hstore vs. Redis - performance wise", *Stack Overflow*, 2012. [Online]. Available: https://stackoverflow.com/questions/9153157/postgres-hstore-vs-redis-performance-wise. [Accessed: 24- May- 2022].

[13]"MoSCoW – projectmanagementsite", *Projectmanagementsite.nl*, 2018. [Online]. Available: https://projectmanagementsite.nl/moscow/#.Yozu3KhBxPY. [Accessed: 24- May- 2022].

[14]"Wat is een User Story?", *Agile Scrum Group*. [Online]. Available: https://agilescrumgroup.nl/wat-is-een-user-story/. [Accessed: 24- May- 2022].

[15]"Kubernetes in SIMPLE WORDS | Nimbella.com® | Nimbella.com®", *Nimbella.com*, 2021. [Online]. Available: https://nimbella.com/blog/kubernetes-in-simple-words-explained-by-eric-swildens. [Accessed: 16- Feb- 2022].

[16]"Viewing Pods and Nodes", *Kubernetes*, 2021. [Online]. Available: https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/. [Accessed: 15- Feb- 2022].

[17]"What is Kubernetes Cluster? | VMware Glossary", *VMware*, 2022. [Online]. Available: https://www.vmware.com/topics/glossary/content/kubernetes-cluster.html. [Accessed: 16- Feb- 2022].

[18]"Page Storage - Apache Wicket - Apache Software Foundation", *Cwiki.apache.org*, 2020. [Online]. Available: https://cwiki.apache.org/confluence/display/wicket/page+storage#PageStorage-HttpSessionDataStore. [Accessed: 15- Feb- 2022].

[19]"GitHub - baholladay/WicketRedisSession: Use Redis server to allow for distributed sessions in Wicket", *GitHub*, 2015. [Online]. Available: https://github.com/baholladay/WicketRedisSession. [Accessed: 06- Apr- 2022].

[20]"SessionScoped (Java(TM) EE 8 Specification APIs)", *Javaee.github.io*, 2017. [Online]. Available: https://javaee.github.io/javaee-spec/javadocs/javax/enterprise/context/SessionScoped.html. [Accessed: 25- Apr- 2022]

[21]"RedisInsight | The Best Redis GUI", *Redis*, 2021. [Online]. Available: https://redis.com/redis-enterprise/redis-insight/. [Accessed: 11- Apr- 2022].

[22]"Production-Grade Container Orchestration", *Kubernetes*. [Online]. Available: https://kubernetes.io. [Accessed: 12- May- 2022].

[23]"FilePageStore (Wicket Parent 9.11.0-SNAPSHOT API)", Nightlies.apache.org. [Online]. Available: https://nightlies.apache.org/wicket/apidocs/9.x/org/apache/wicket/pageStore/FilePageStore.html. [Accessed: 26- May- 2022].

[24]"RESP protocol spec", *redis.io*, 2022. [Online]. Available: https://redis.io/docs/reference/protocol-spec/. [Accessed: 28- May- 2022].

[25]"Use volumes", *Docker Documentation*. [Online]. Available: https://docs.docker.com/storage/volumes/. [Accessed: 08- Jun- 2022].

[26]"Infinispan Archive", *Infinispan.org*. [Online]. Available: https://infinispan.org/download-archive. [Accessed: 09- Jun- 2022].

[27]"Redis: in-memory data store. How it works and why you should use it", *Amazon Web Services, Inc.*. [Online]. Available: https://aws.amazon.com/redis/. [Accessed: 25- Feb- 2022].

[28]"Memorystore: in-memory data store | Google Cloud", *Google Cloud*. [Online]. Available: https://cloud.google.com/memorystore. [Accessed: 06- Apr- 2022].

[29]"redis Tutorial - How to Connect to Redis in Java using Jedis", *Sodocumentation.net*. [Online]. Available: https://sodocumentation.net/redis/topic/9712/how-to-connect-to-redis-in-java-using-jedis. [Accessed: 02- Mar- 2022].

[30]"What is the disadvantage of just using Redis instead of an RDBMS?", *Stack Overflow*, 2012. [Online]. Available: https://stackoverflow.com/questions/10906246/what-is-the-disadvantage-of-just-using-redis-instead-of-an-rdbms. [Accessed: 25- Feb- 2022].

[31]"Why developers like Redis", *StackShare*. [Online]. Available: https://stackshare.io/redis. [Accessed: 06- Apr- 2022].

[32]"redis Tutorial => Installing and running Redis Server on Windows", *Riptutorial.com*. [Online]. Available: https://riptutorial.com/redis/example/29962/installing-and-running-redis-server-on-windows. [Accessed: 02- Mar- 2022].

[33]"Redis Quick Start – Redis.", *redis.io*. [Online]. Available: https://redis.io/topics/quickstart. [Accessed: 02- Mar- 2022].

[34]"Documentation.", *Redis*. [Online]. Available: https://redis.io/docs/. [Accessed: 06- Apr- 2022].

[35]"benchmarks", *Redis*. [Online]. Available: https://redis.io/docs/manual/optimization/benchmarks/. [Accessed: 06- Apr- 2022].

[36]"Redis: Open Source vs. Enterprise | MetricFire Blog", *Metricfire.com*. [Online]. Available: https://www.metricfire.com/blog/redis-open-source-vs-enterprise/. [Accessed: 06- Apr- 2022].

[37]"Redis", *Grafana Labs*. [Online]. Available: https://grafana.com/grafana/plugins/redis-datasource/. [Accessed: 06- Apr- 2022].

[38]"Redis release cycle.", *Redis*. [Online]. Available: https://redis.io/docs/about/releases/. [Accessed: 06- Apr- 2022].

[39]"Maven Repository: org.wicketstuff » wicketstuff-datastore-redis.", *mvnrepository*. [Online]. Available: https://mvnrepository.com/artifact/org.wicketstuff/wicketstuff-datastore-redis. [Accessed: 06- Apr- 2022]

[40]"What Is Redis and Why Is It so Popular?", *Eduonix Blog*, 2019. [Online]. Available: https://blog.eduonix.com/web-programming-tutorials/what-is-redis-and-why-is-it-so-popular/. [Accessed: 10- Mar- 2022].

[41]"GitHub - infinispan/infinispan-cachestore-redis: Infinispan Redis cache store", *GitHub*. [Online]. Available: https://github.com/infinispan/infinispan-cachestore-redis. [Accessed: 02- May- 2022].